

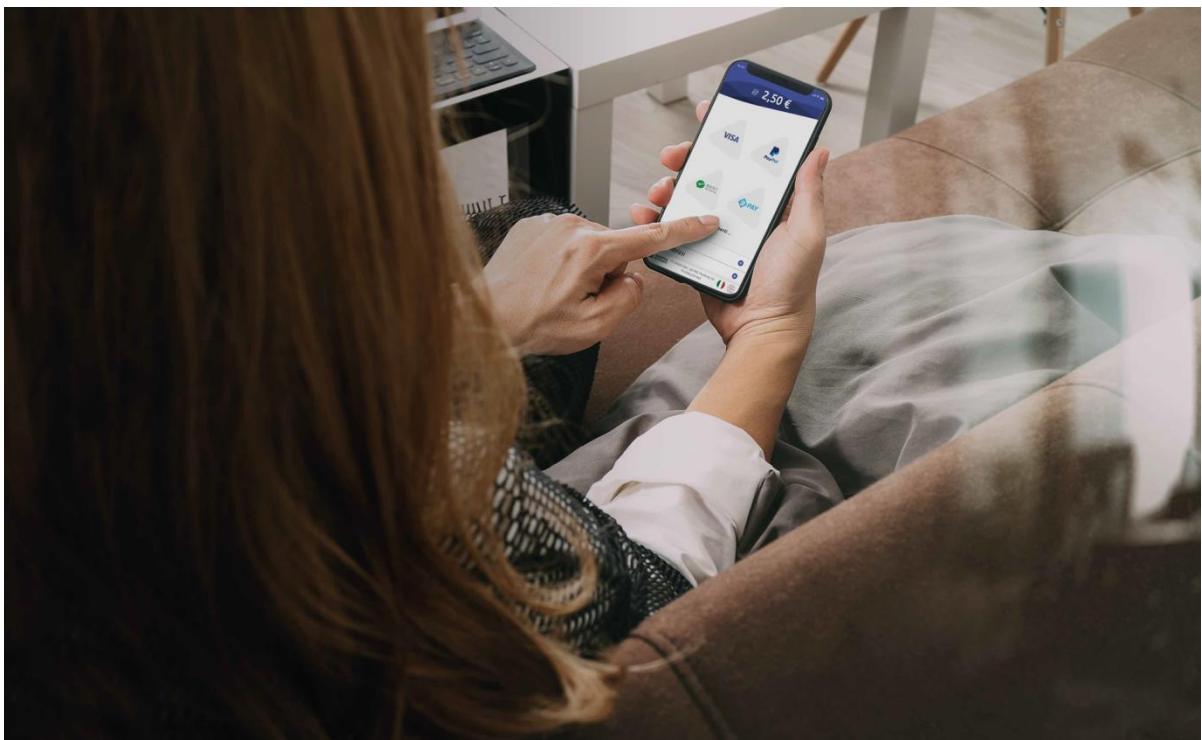


AXEPTA
BNP PARIBAS

AXEPTA BNP Paribas

pHey Integration Guide

v. 1.3.4





DOCUMENT VERSIONS

Version	Description of changes
1.0.0	First issue
1.0.1	Document outline revision
1.0.2	IT content revision
1.0.3	Business content revision
1.0.6	IT content revision
1.0.7	Document outline revision
1.0.8	Insertion of references to sections
1.0.9	Graphic revision
1.0.10	IT content revision (strings limited to 2048 bits, supported languages, removed the TID in <code>initPayment</code>)
1.0.11	IT content revision (body and response to server-to-server calls updated)
1.1.0	Added Android SDK and iOS SDK guides Added "Smart layout customization" section Error code revision Formatting revision
1.1.1	Typos in examples
1.1.2	Specified which licenses to use in the various Requests and redacted notification email
1.1.3	Detailed description on merchant-side PCI DSS Fixed typos on host
1.1.4	Typo in Android SDK in the "Implementation in an Activity" section, letter "f". <code>initPayment</code> , removed obligation of the <code>redirect_successUrl</code> , <code>redirect_failureUrl</code> and <code>callback_url</code> parameters
1.1.5	Smart integration, transaction outcome management
1.1.6	<code>redirect_successUrl</code> and <code>redirect_failureUrl</code> were changed from POST-type calls to REDIRECT. Easy integration, transaction outcome management
1.1.7	Fixed test PANs
1.1.8	Specified that the Access Token must be sent in Bearer mode Verifying a Transaction (<code>verify</code>) section inserted
1.1.9	Types corrected in "Smart integration" example
1.2.0	Document outline revision Tokenize function added to various Sections. Complete revision of In-App integrations (iOS and Android) Specification on <i>Access Tokens</i> <i>InitPayment</i> , description of <i>amount</i> when card is verified <i>InitPayment</i> , enumeratives added to address type "Making payments with 3DS" section. example modified Easy integration: compatible browsers, clarifications on <i>Verify</i> Smart integration: compatible browsers, <code>window.postMessage()</code> , clarifications on <i>Verify</i>



	<p>“Information for test environment” section, <i>enrStatus</i> e <i>authStatus</i></p> <p>“Information for test environment” section, test cases</p> <p>New “Information for release to Production” section</p>
1.2.1	Verify, <i>PaymentID</i> in input section
1.2.2	<p>References to APMs removed</p> <p>Notes added for the special case of the Credit of Confirm</p> <p>Added Par. “Making one-click payments”</p>
1.3.0	<p>Par. “Support information”, new email</p> <p><i>transaction_type</i> clarification in <i>initPayment</i></p> <p><i>shopID</i> clarification in <i>initPayment</i></p> <p><i>logo</i> parameter substituted with <i>image</i> parameter in <i>initPayment</i></p> <p>Par. “Smart Layout Management”, new layout “buttonless”</p> <p>SDK iOS, added Xcode functional requirements</p> <p>SDK Android, removed amount from <i>ApiService.paymentConfiguration()</i></p> <p>Par. “Making one-click payments”, complete review</p> <p>Par. “Deleting a tokenized card”, added paragraph</p> <p>Par. “Information for test environment”, added two test cases</p> <p>Par. “Testing 3DS 2.x”, added paragraph</p>
1.3.1	Par. “Recurring payments”, added paragraph
1.3.2	Par. “Recurring payments”, modified paragraph
1.3.3	<p>Par. “Payment Initialization (<i>initPayment</i>)”, removed <i>payInstrToken</i> and <i>payCardToken</i> in the examples</p> <p>Par. “Credit”, removed <i>payInstrToken</i> and <i>payCardToken</i> in the examples</p> <p>Par. “Void”, removed <i>payInstrToken</i> and <i>payCardToken</i> in the examples</p> <p>Par. “Confirm”, removed <i>payInstrToken</i> and <i>payCardToken</i> in the examples</p> <p>Par. “Payment Initialization (<i>initPayment</i>)”, added type values <i>SPEDIZIONE</i> and <i>FATTURAZIONE</i></p> <p>Par. “Payment execution (<i>execute</i>)”, <i>card_brand</i> is not mandatory</p> <p>Par. “Direct payment (<i>directPayment</i>)”, <i>card_brand</i> is not conditional</p> <p>Par. “Payment Initialization (<i>initPayment</i>)”, job to cancel payments.</p> <p>Par. “Making one-click payments”, case of <i>payCardToken null</i></p> <p>Par. “In-App checkout (SDK iOS)”, <i>itemId</i> value must be the <i>PaymentID</i> on the <i>initPayment</i></p>
1.3.4	<p>Par. “Payment outcomes”, added paragraph</p> <p>Par. “Easy checkout”, payment result moved to new Par. “Payment outcomes”</p> <p>Par. “Smart checkout”, payment result moved to new Par. “Payment outcomes”</p> <p>More paragraphs, <i>callback_url</i> must be in HTTPS</p> <p>More paragraphs, <i>redirect_successUrl</i> must be in HTTPS</p> <p>More paragraphs, <i>redirect_failureUrl</i> must be in HTTPS</p> <p>Par. “In-App checkout (SDK iOS)”, modified functional requirements</p> <p>Par. “In-App checkout (SDK iOS)”, modified SDK configuration</p> <p>Par. “In-App checkout (SDK Android)”, modified SDK configuration</p>



AXEPTA
BNP PARIBAS

Par. "In-App checkout (SDK Android)", modified example in
PaymentSelectorActivity Integration

Par. "Confirm with automatic void of the residual", added paragraph



CONTENTS

1	Introduction	8
1.1	Checkout solutions	8
1.2	Payment instruments	8
1.3	Additional services.....	8
1.3.1	Saving payment data	9
1.3.2	Payment notifications	9
1.3.3	Customer notifications	9
1.3.4	Easy Checkout personalization.....	10
1.3.5	Easy Checkout optional forms.....	10
1.4	Support information	11
2	Initializing a payment	12
2.1	Payment Initialization (initPayment)	12
2.1.1	Example of Java Unirest integration	17
2.1.2	Example of PHP http Request integration.....	18
2.1.3	Example of Node Request integration	20
3	Types of integration	22
4	API checkout.....	23
5	Easy checkout.....	24
6	Smart checkout	26
6.1	Smart layout management	28
6.2	Smart layout personalization.....	30
7	In-App checkout (SDK iOS)	32
7.1	Functional Requirements	32
7.2	Introduction	32
7.3	Adding SDK to the project	32
7.4	SDK Configuration.....	34
7.5	SDK implementation.....	36
7.5.1	Payment Context Integration	36
7.5.2	Widget integration	37
7.5.3	Integration by direct call on a specific payment method	41
7.5.4	Setting the Credit Card view in direct payments	41
8	In-App checkout (SDK Android)	42



8.1	Minimum Requirements.....	42
8.2	Adding dependency for SDK	42
8.3	SDK configuration	44
8.4	Graphic personalization.....	44
8.5	SDK Integration.....	45
8.6	PaymentSelectorActivity Integration	45
8.7	FragmentPayment Integration	47
8.7.1	Implementation in an Activity.....	47
8.7.2	Implementation in a Fragment	50
8.8	Direct call integration	50
8.9	Examples of the code.....	51
8.10	Examples of Layouts	53
9	Payment outcomes	54
10	After the payment	56
10.1	Credit	56
10.1.1	Example of Java Unirest	59
10.1.2	Example of PHP Http Request	59
10.1.3	Example of Node Request	60
10.2	Void	61
10.3	Confirm	64
10.3.1	Confirm with automatic void of the residual	66
10.4	Verifying a Transaction (verify).....	68
10.5	Making one-click payments.....	70
10.5.1	Phase 1 - Configuration	70
10.5.2	Phase 2 - Card tokenization.....	71
10.5.3	Phase 3 - Payment using token	73
10.6	Recurring payments (scheduled by the merchant)	75
10.6.1	Phase 1 - Configuration	75
10.6.2	Phase 2 - Card tokenization.....	75
10.6.3	Phase 3 - Payment using token	75
10.7	Deleting a tokenized card	77
11	Error codes	78
12	Testing environment information	79



12.1	Testing 3DS 2.x.....	81
13	How to move to Production environment	82
14	Merchant-side PCI data security information	83



AXEPTA
BNP PARIBAS

1 Introduction

This document presents useful information for the integration of a website with the Axepta e-commerce platform.

Websites offering e-commerce services usually provide a shopping cart with the summary of the products purchased and a button to complete the purchase through a payment request that can end on your website or through a redirection to a third-party site.

1.1 Checkout solutions

Axepta has several checkout solutions which are useful for optimizing conversions and increasing online sales. They can be easily integrated into your website:

- **Easy Checkout** – With the Easy solution your customers can finish their purchase on a personalized and optimized payment page for each device, which can be integrated with your website very quickly, without having to worry about the security of the payment that is guaranteed by Axepta
- **Smart Checkout** – Your customers can pay directly on your site and the transaction data will be handled by Axepta in secure mode thanks to Axepta's default UI components that can be easily integrated into your website with limited data security charges borne by you (details in the Merchant-side PCI data security information section)
- **In-App Checkout** – With the Smart solution, you can integrate our iOS and Android SDKs to have your customer complete their purchase directly in the App with the best mobile User Experience
- **API Checkout** – Axepta APIs can be used to integrate the payment request within your website (card data is managed by the merchant server)

1.2 Payment instruments

Axepta has several payment instruments that can be accepted with a single integration, including the VISA, VISA Electron, VPay, Mastercard, Maestro, American Express and Diners schemes.



1.3 Additional services

This integration guide has all the necessary information to integrate your website with the Axepta e-commerce platform.



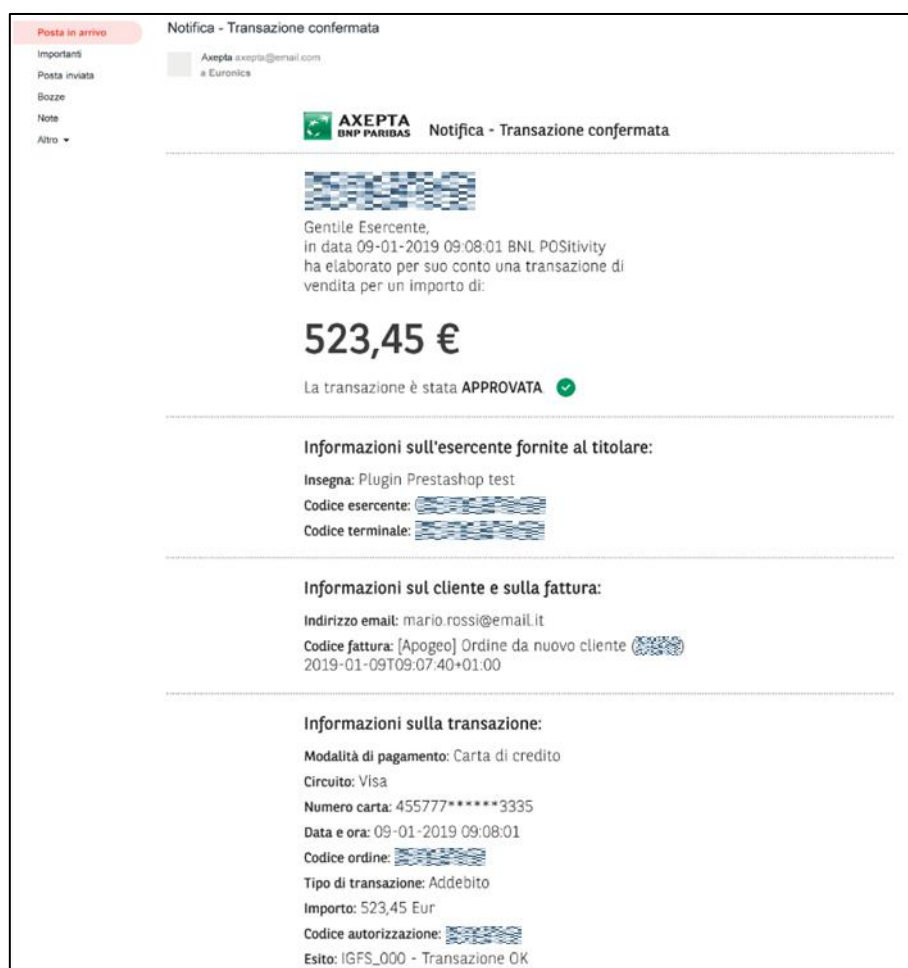
AXEPTA
BNP PARIBAS

1.3.1 Saving payment data

On request, by activating the “one-click” payments function, you can enable the service that lets you save payment data to facilitate future payments. For details on the calls to be made, see the “Making one-click payments” section.

1.3.2 Payment notifications

On request, you can enable a notification for each completed payment to receive via email, SMS.



You can ask our support team to modify the communications sent (see Section 1.4).

1.3.3 Customer notifications

On request, you can enable a customer notification for each completed payment to be sent via email or SMS.




AXEPTA
BNP PARIBAS

Posta in arrivo
Importanti
Posta inviata
Bozze
Note
Altro ▾

Conferma pagamento


Axepta axepta@email.com
a Mario Rossi

AXEPTA BNP PARIBAS Conferma pagamento

AXEPTA  STORE

523,45 €

Gentile Mario,
hai acquistato il seguente prodotto:

 Fotocamere reflex
NIKON - D3400 + AF-P DX 18-55MM
Colore: nero

Informazioni sull'acquisto:
Nome prodotto: NIKON - D3400 + AF-P DX 18-55MM
Data: 17 / 08 / 2019
Metodo di pagamento: Visa - **** * 1234
Esercente: Axepta store
Prezzo: 523,45 €

Per ulteriori informazioni [controlla il tuo acquisto.](#)

Cordiali saluti,
il team Axepta

You can ask our support team to change communications that have been sent (see section 1.4).

1.3.4 Easy Checkout personalization

Your payment page can be personalized with your logo and colours



1.3.5 Easy Checkout optional forms

Using the Easy Checkout solution, you can activate, on request, the following forms on the payment page:

- **Cart** - The cart summary can be displayed on the payment page, including the logo, quantity, description and unit price of each product purchased



- Addresses - The summary of shipping and billing addresses can be displayed, showing the recipient, address, postcode, city, province and state for each.

1.4 Support information

This integration guide has all the necessary information to integrate your website with the Axepta e-commerce platform.

For any further information, simply write to the email address:

ecommercesupport@axeptamail.com or contact us at **(+39) 060 070** selecting option 1 and then 4.



2 Initializing a payment

To make a payment using the Axepta Payment Gateway (pHey), a Merchant must be configured on Axepta side.

You must then be in possession of a valid key (**AccessToken**) for your user name and a user license. This Access Token can be obtained using your credentials at the following address <https://pay-test.axepta.it/access>. It is recommended that you keep the access token in a safe place and set this access token as a setup parameter on your e-commerce system, so that it can be replaced easily if the need arises. The access token lasts for 10 years but can be revoked at the specific request of the merchant or for Axepta needs.

Finally, you must possess an **API License Key**, which is indispensable for initializing the payment using server to server APIs. The licence is of the alphanumeric type (for example: "ce7e4f96-3fa1-4696-a669-80cfe2805411").

2.1 Payment Initialization (initPayment)

The payment initialization call is **required for every transaction that you want to carry out regardless of the integration method** the merchant wishes to use. This is a technical payment initialization call that **DOES NOT trigger any type of authorization flow**.

The specifications of the initialization call are below:

METHOD	POST
ENDPOINT	{{host server to server}}/api/v1/payment/initPayment
HEADERS	"Content-type": application/json "Authorization": Bearer <-- AccessToken --> "X-license-key": <-- API License Key -->
(BODY) PARAMETERS	{ "transaction_type": "PURCHASE", "transaction_timeout": "30000", "shopID": "shopIDprova12345", "currency": "EUR", "language": "IT", "amount": "84.58", "notifications": { "name": "Mario Rossi", "email": " test@test.it ", "smartphone": "3332233220" }



	<pre>}, "addresses": [{ "type": "SHIPPING", "addresseeName": "Mario Rossi", "streetAddress_1": "Via del Corso 1", "streetAddress_2": "C/O Axepta.", "zip": "00100", "city": "Rome", "provinceState": "RM", "country": "Italy" }, { "type": "BILLING", "addresseeName": "Michele Bianchi", "streetAddress_1": "Via del Corso 1", "zip": "00100", "city": "Rome", "provinceState": "RM", "country": "Italy" }], "addressesURI": "https://www.merchantSite.com/addressEdit", "products": [{ "logo": "https://logoRef.com/logo", "quantity": 1, "description": "Product1", "price": "49.75" }, { "logo": " https://logoRef.com/logo ", "quantity": 1, "description": "Product2", "price": "34.83" }], "redirect_successUrl": "https://www.merchantSite.it/success_redirect", "redirect_failureUrl": "https://www.merchantSite.it/failure_redirect", "callback_url": "https://www.merchantSite.it/callback", "additional": [{ "key": "Test", "value": "Value" }, { "key": "Test2", "value": "Value2" }] } }</pre>
RESPONSE	<pre>{ "code": 200, "message": "SUCCESSFULLY", }</pre>



	"paymentID": <-- payment ID --> }
ERROR RESPONSE	{ "code": 1118, "message": "The data are necessary." }

The *callback_url* parameter lets you receive the outcome of the transaction and the related data. Please see Par. "Payment outcomes".

The formats of the fields requested in **Input** are as follows:

Field Name	Format	Description	
transaction_type	Enumerative: [PURCHASE,AUTH,VERIFY]	Describes the desired type of transaction. PURCHASE means that the cardholder would be charged immediately, AUTH means preauthorization (an amount would be only blocked on the card but not charged), VERIFY means that the card would be only verified if it is valid (note that you must specify an amount greater than zero, even if this amount would not be used).	*1
transaction_timeout	Numeric format string	Indicates the maximum waiting time for the transaction, the value is expressed in milliseconds.	
payInstrToken	String	Token identifying the customer's wallet. For example, the user's e-mail or ID can be passed to Axepta. See Par. "Making one-click payments"	
payCardToken	String	Token identifying a tokenized card belonging to the wallet corresponding to "payInstrToken". It is not normally sent in input but is returned in output to the merchant after tokenizing. See from Par. "Making one-click payments"	
txIndicatorType	Enumerative: [UNSCHEDULED, RECURRENT, NOSHOW, DELAYCHARGE]	Indicator of the type of transaction to be used when paying with the wallet. See from Par. "Making one-click payments"	
tokenize	Boolean	Enables card tokenizing. See from Par. "Making one-click payments"	

¹ The fields indicated with an asterisk are mandatory



shopID	String (max 64)	Unique identifier of the transaction on merchant side. This string could be optionally passed in input, if needed for the merchant. Otherwise, a random string would be generated by Axepta. IMPORTANT: it is not possible to have two or more successfully transactions with the same shopID. It means that if a transaction fails, then the merchant may use again the same shopID. It means that if a transaction succeeds, then the merchant must NOT use again the same shopID.	
Currency	ISO 4217 format string, e.g. "EUR"	Currency to be used according to ISO 4217 format	*
Language	ISO 639-1 format string, possible values: IT, EN, FR, RU, JP, CN, NL, PL, ES, DE	Language to be used in the client implementation	*
Amount	String ¹	Amount, formatted with two mandatory decimal places, separated by a dot ".". For the "Verify" transaction_type, the amount should be set to at least "0.01" (uninfluential for authorization).	*
Notifications:			
name	String	Name of the Customer making the payment.	
email	e-mail format string	Customer email address for sending notifications via email.	
smartphone	String	Customer Smartphone number for sending notifications via SMS.	
Addresses:			
type	Enumerative: [SHIPPING,BILLING,SP EDIZIONE,FATTURAZI ONE]	Describes the address type.	
addresseeName	String	Address name.	
streetAddress_1	String	Street name.	
streetAddress_2	String	Additional field for Street name.	
zip	Postal code format string	Postal Code.	
city	String	City.	
provinceState	ISO 3166 format string, e.g. "RM"	Province.	
country	String	Country.	

¹ All the string fields support a maximum of 2048 characters, unless specified



addressesURI	URL format string	Merchant site URL for modification in shipping fields, if necessary.	
Products:			
image	String	URL of the product image.	
quantity	String	Product quantity.	
description	String	Product description.	
price	String	Product price.	
redirect_successUrl	URL format string	For the Easy integration, when the transaction has <u>successfully</u> concluded, the application performs a REDIRECT to this URL. URL must be in HTTPS. This URL is also used in some alternative payment methods, please see respective documents, if needed.	
redirect_failureUrl	URL format string	For the Easy integration, when the transaction has <u>incorrectly</u> concluded, the application performs a REDIRECT to this URL. URL must be in HTTPS. This URL is also used in some alternative payment methods, please see respective documents, if needed.	
callback_url	URL format string	Merchant callback URL to receive the outcome of the transaction asynchronously. URL must be in HTTPS	
Additional:		If you would like to add additional information to the transaction, you can specify this section.	
key	String	Additional information key.	
value	String	Additional information value.	



2.1.1 Example of Java Unirest integration

```
HttpResponse<String> response = Unirest.post("https://pay-test.axepta.it/api/v1/payment/initPayment")
    .header("Content-Type", "application/json")
    .header("x-license-key", "XXXXXXX-0ERMYE0-MP683C5-9G0Q976")
    .header("Authorization", "Bearer eyJhbGciOiJSUzI1NiIsInR5cCI6IkpzZW50L3N1bWU0ODYiLCJleHAiOiE4ODQzMzUzNzMsIm5iZil6MCwiaWF0IjoxNTY4OTc1NzYxLCJpc3MiOiJodHRwczovL3Nzby10ZXN0LmF4ZXB0YS5pZC9hdXR0L3JlYWxtcy9BeGVwdGEIMjBJbnRlcm5ldCIsInN1Yil6ImI1NDYzN2M3LWQyZTktNDkxNC04YmNkLWE0Y2MxNjliZDAzOCIsInR5cCI6IkpzZW50L3N1bWU0ODYiLCJhdXRoX3RpbWUiOiE1Njg5NzUzNzMsInNlc3Npb25fc3RhdGU0IjIN2E3MTg4MS01MGQ2LTQzN2Q0ODQyZS00MjAzNDcwYmY3NjliLCJhY3liOiIiwiaWwiOiI2NvcGU0IjVvcGVuaWQifQ.N_4xTX9FjGTMzFsc7fERvciU4RAdXqCgeMoaymjScGCCSabZSApG5a-ybeYEA5mC9hUWfwgYZSRLWTJhbnRz4vsc2vSwdR0xY_YlmbdC-y1IV-IQkCLyOOEOwdl65slc2fHAZlrBE3jVo6nV6ee81meEGQueueMQ4L1hWO1u73ZlWKLzd_5YpRFKFD8HsNPKODdJV6V2o1q2vqkJhfSOD9e3iJn_ehuqEs35m1dyiwUwFXXnMeq1aK1QiVDdKTPzG5_46XJ3ixTfVQ-3eBPqJcW3S3WYPy4wqtud85oPT6NYuXDh-VKuGg2A13_2TAeFxQqvgGEU-RUGJdFR0xTY0A")
    .header("cache-control", "no-cache")
    .body("{\"transaction_type\": \"PURCHASE\", \"currency\": \"EUR\", \"language\": \"IT\", \"amount\": \"84.58\", \"notifications\": {\"name\": \"Paolo Verdi\", \"email\": \"paoloverdi@axepta.it\", \"smartphone\": \"\", \"addresses\": [{\"type\": \"SHIPPING\", \"addresseeName\": \"Mario Bianchi\", \"streetAddress_1\": \"Via del Corso 1\", \"streetAddress_2\": \"C/O Axepta.\", \"zip\": \"00100\", \"city\": \"Roma\", \"provinceState\": \"RM\", \"country\": \"Italia\"}, {\"type\": \"BILLING\", \"addresseeName\": \"Francesco Bianchi\", \"streetAddress_1\": \"Via del Corso 1\", \"zip\": \"00100\", \"city\": \"Roma\", \"provinceState\": \"RM\", \"country\": \"Italia\"}], \"addressesURI\": \"https://www.shop.com\", \"products\": [{\"logo\": \"https://www.shop.com/01.jpg\", \"quantity\": 1, \"description\": \"Product\", \"price\": \"49.75\"}, {\"logo\": \"https://www.shop.com/02.jpg\", \"quantity\": 1, \"description\": \"Product description\", \"price\": \"34.83\"}], \"redirect_successUrl\": \"https://www.shop.com\", \"redirect_failureUrl\": \"https://www.shop.com\", \"callback_url\": \"https://www.shop.com\", \"additional\": [{\"key\": \"key_1\", \"value\": \"value_1\"}, {\"key\": \"key_2\", \"value\": \"value_2\"}] }")
    .asString();
```




```
"city": "Rome",
"provinceState": "RM",
"country": "Italy"
},
],
"addressesURI": "https://www.shop.com",
"products": [
{
"logo": "https://www.shop.com/01.jpg",
"quantity": 1,
"description": "Product",
"price": "49.75"
},
{
"logo": "https://www.shop.com/02.jpg",
"quantity": 1,
"description": "Product description",
"price": "34.83"
}
],
"redirect_successUrl": "https://www.shop.com",
"redirect_failureUrl": "https://www.shop.com",
"callback_url": "https://www.shop.com",
"additional": [{
"key": "key_1",
"value": "value_1"
},
{
"key": "key_2",
"value": "value_2"
}
]
}]);

try {
$response = $request->send();

echo $response->getBody();
} catch (HttpException $ex) {
echo $ex;
}
```



2.1.3 Example of Node Request integration

```
var request = require("request");

var options = { method: 'POST',
  url: 'https://pay-test.axepta.it/api/v1/payment/initPayment',
  headers:
    { cache-control: 'no-cache',
      Authorization: 'Bearer eyJhbGciOiJSUzI1NiIsInR5cCIgOiAiSldUiIiwia2lkIA6ICjkbklyWVZhRHFGbF9lUFQ4QlZ3bGE3bm8yN2lXaWNXd25fVkNPdWZTTURjIn0.eYjqdGkiOii4YTzkZjdZCOWZWlwLTRjNWMTOWIOMS1INjRmZjJmNWU5ODYiLCJleHAiOiE4ODQzMzUzNzMsIm5iZil6MCwiaWF0IjoxtNTY4OTc1NzYxLCJpc3MiOiJodHRwczovL3Nzby10ZXNOlmF4ZXBOYS5pdC9hdXRol3JIYWxtcy9BeGVwdGEIMjBJbnRlcm5ldClSnN1YiI6ImliNDYzN2M3LWQyZTkktNDkxNC04YmNkLWE0Y2MxNjliZDAZOCIsInR5cCI6IkllYXJlcilScmlmF6cCI6InBnLXBheW1lbnQtYXBPbWluZXQiLCJhdXRoX3RpbWUiOiE1Njg5NzUzNzMsInNlc3Npb25fc3RhdkUiOiIlN2E3MTg4MS01MGQ2LTQzN2QtODQyZS00MjAzNDcwYmY3NjliLCJhY3liOilwiwic2NvcGUiOiJvcGVuaWQifQ.N_4xTX9FjGTmZfsc7fERvciU4RAAdXqCgeMoaymjScGCCSabZSApG5a-ybeYTEA5mC9hUWFwgyszSRLWTJhbnRz4vsc2vSwdr0xY_YlmbdC-y1IV-lQkCLyOOEOwdl65slc2fHAZlrBE3jVo6nV6ee81meEGQueueMQ4L1hWO1u73ZIWKLzd_5YprFKFD8HsNPKODdJV6V2o1q2vqkJhfS0D9e3iJn_ehuqEs35m1dyiwUwFXxnMeq1aK1QiVDdKTpZG5_46XJ3ixTfVQ-3eBPqJCwS3WYPy4wqtftud85oPT6NYuXDh-VKuGg2A13_2TAeFxQvgGEU-RUGJdFRxxxxxx',
      'x-license-key': 'XXXXXXX-0ERMYE0-MP683C5-9G0Q976',
      'Content-Type': 'application/json' },
  body:
    { transaction_type: 'PURCHASE',
      currency: 'EUR',
      language: 'IT',
      amount: '84.58',
      notifications:
        { name: 'Paolo Verdi',
          email: 'paoloverdi@axepta.it',
          smartphone: '' },
      addresses:
        [ { type: 'SHIPPING',
            addresseeName: 'Mario Bianchi',
            streetAddress_1: 'Via del Corso 1',
            streetAddress_2: 'C/O Axepta',
            zip: '00100',
            city: 'Rome',
            provinceState: 'RM',
            country: 'Italy' },
          { type: 'BILLING',
            addresseeName: 'Francesco Bianchi',
            streetAddress_1: 'Via del Corso 1',
            zip: '00100',
            city: 'Rome',
            provinceState: 'RM',
            country: 'Italy' } ],
      addressesURI: 'https://www.shop.com',
      products:
        [ { logo: 'https://www.shop.com/01.jpg',
            quantity: 1,
            description: 'Product',
```



```
price: '49.75' },  
{ logo: 'https://www.shop.com/02.jpg',  
  quantity: 1,  
  description: 'Product description',  
  price: '34.83' } ],  
redirect_successUrl: 'https://www.shop.com',  
redirect_failureUrl: 'https://www.shop.com',  
callback_url: 'https://www.shop.com',  
additional:  
  [ { key: 'key_1', value: 'value_1' },  
    { key: 'key_2', value: 'value_2' } ] },  
json: true }];  
  
request(options, function (error, response, body) {  
  if (error) throw new Error(error);  
  
  console.log(body);  
});
```



AXEPTA
BNP PARIBAS

3 Types of integration

According to Business needs, the merchant can choose one of the following integration methods:

- API checkout
- Easy checkout
- Smart checkout
- In-App checkout (SDK iOS)
- In-App checkout (SDK Android)

These types of integration are described in the sections that follow.



AXEPTA
BNP PARIBAS

4 API checkout

Axepta APIs can be used to integrate the payment request within your website (the card data is managed by the merchant's server). This type of integration has the highest level of personalization but also the highest implementation complexity on the merchant side. For this reason, it is only applicable to special needs.



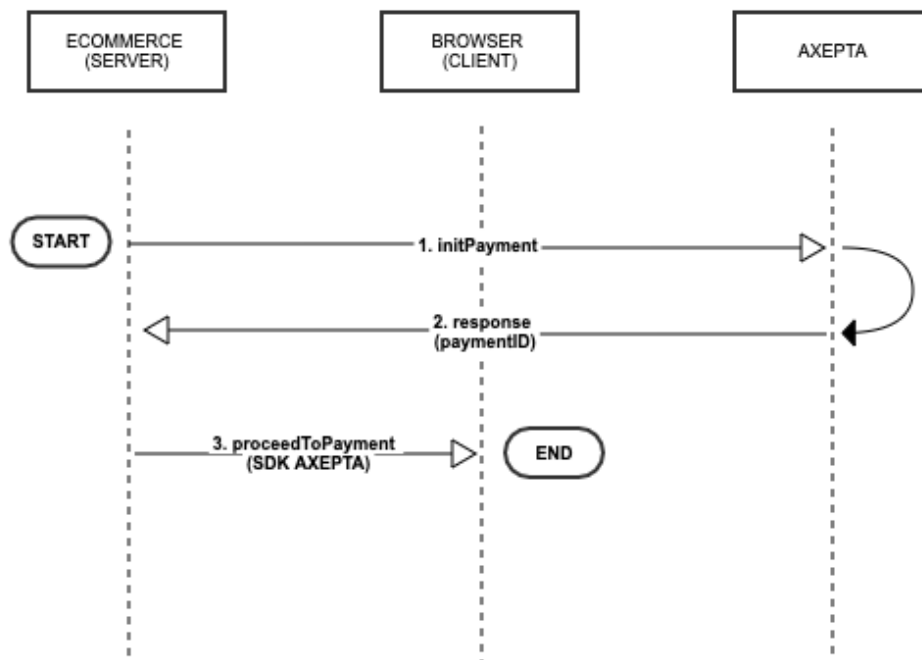
5 Easy checkout

Once configured, this integration, upon checkout, performs a redirect to the payment page provided by Axepta.

The compatible versions of the browsers are as follows:

- Chrome 51 May 2016
- Firefox 54 Jun 2017
- Edge 14 Aug 2016
- Safari 10 Sep 2016
- Opera 38 Jun 2016
- IE 11

The integration flow for performing an **Easy** integration is shown below:



- Invoke the *initPayment* service and retrieve the *PaymentID* parameter necessary for the subsequent calls;
- Initialize the *clientAxepta* with the *Easy license key* and the *PaymentID* (returned by *initPayment*) to invoke the *proceedToPayment* javascript function.

An example of the **Easy** integration is shown below:

```
<!DOCTYPE html>
<html>
<head>
  <title>SDK Redirect Integration Example</title>
  <meta name="viewport" content="initial-scale=1.0">
  <meta charset="utf-8">
</head>
```




AXEPTA
BNP PARIBAS

```
<body>
  <button type="button" onClick="axaptaClient.proceedToPayment('← PaymentId →')">Checkout</button>

  <script src="https://pay-test.axapta.it/sdk/axapta-pg-redirect.js"></script>
  <script type="text/javascript">
    let axaptaClient = new AxaptaSDKClient("https://pay-test.axapta.it", "← Easy type LICENSE Key →");
  </script>
</body>
</html>
```

Please see Par. “Payment outcomes” for explanations on how to get transaction results.



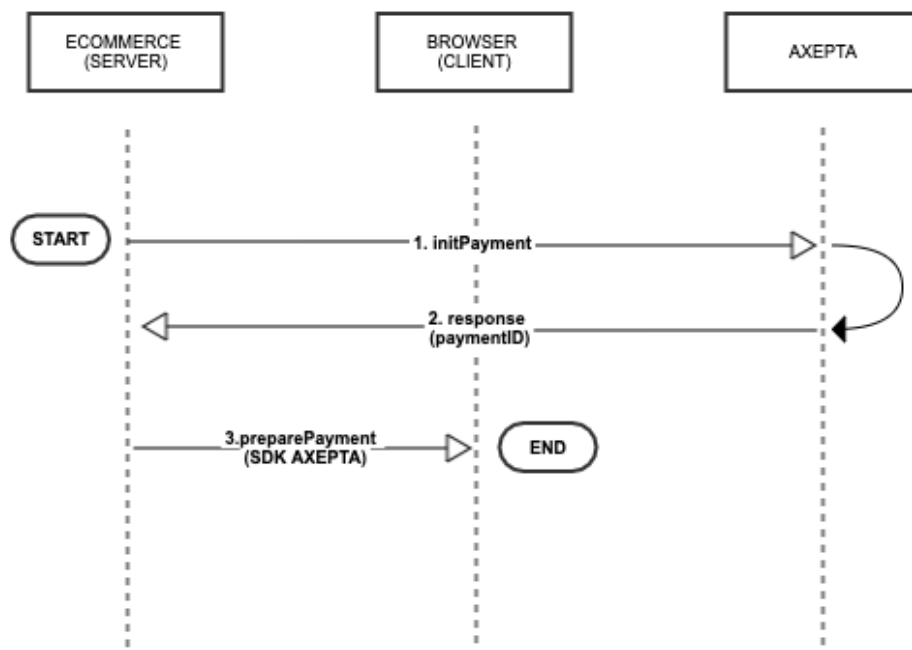
6 Smart checkout

This type of integration is a simple implementation of the card fields that can be integrated within your site. Unlike the Easy integration, the Smart integration requires that a div is created within your e-commerce site. In this case the fields of interest (for example, those of the card for the payment) will be displayed inside the div itself.

The compatible versions of the browsers are listed below:

- Chrome 51 May 2016
- Firefox 54 Jun 2017
- Edge 14 Aug 2016
- Safari 10 Sep 2016
- Opera 38 Jun 2016
- IE 11

The integration flow for performing a **Smart** integration is shown below:



- Invoke the *initPayment* service and retrieve the *PaymentID* parameter necessary for the subsequent calls;
- Initialize the *clientAxepta* with the *Smart license key* and the *PaymentID* (returned by *initPayment*) to invoke the *preparePayment* javascript function

An example of **Smart** integration is shown below:

```
<!DOCTYPE html>
<html>
<head>
<title>SDK JS Integration Example</title>
<meta name="viewport" content="initial-scale=1.0">
```



AXEPTA
BNP PARIBAS

```
<meta charset="utf-8">
</head>
<body>
  <!--EXAMPLE OF CHECKOUT BUTTON -->
  <button type="button"
onClick="axeptaClient.preparePayment('← paymentID →','inline')">Checkout</button>
  <!-- THE TAG WHERE THE HOSTED FORM WILL BE DISPLAYED -->
  <div id="my-axepta-sdk-pg"></div>
  <!-- THE SCRIPT TO BE RETRIEVED BY CDN -->
  <script src="https://pay.axepta.it/sdk/axepta-pg-sdk.js"></script>
  <!--INITIALIZATION OF CLIENT AND USE -->
  <script type="text/javascript">
    let axeptaClient = new AxeptaSDKClient("https://pay-test.axepta.it", "← Smart LICENSE →");
  </script>
</body>
</html>
```

Please see Par. “Payment outcomes” for explanations on how to get transaction results.




AXEPTA
BNP PARIBAS

6.1 Smart layout management

In this integration, you can define three different layouts using an optional parameter within the feature described above:


```
axeptaClient.preparePayment('←-PaymentID-→', '←- Layout ->');
```

If you do not enter the Layout parameter, the default layout will be displayed:

Nome	Cognome
1234 1234 1234 1234 	
MM/AA	CVV
PAY	

To obtain a layout with no name and surname, the **preparePayment()** call must contain a second parameter that is 'compact':

```
axeptaClient.preparePayment('←- PaymentID -->', 'compact');
```

1234 1234 1234 1234 	
MM/AA	CVV
PAY	



AXEPTA
BNP PARIBAS

If you would like an inline layout without the PAY button, the ***preparePayment()*** call must contain a second parameter that is *'inline'*:

axeptaClient.preparePayment('←- PaymentID -->', 'inline');


VISA	4557 7733 3333 3335	10/23	 111
-------------	---------------------	-------	---

and in the script, the form can be posted with the method ***axeptaClient.submit();***

```
<!-- ESEMPIO DI PULSANTE SUBMIT -->  
<button id="submit" type="button" onClick="axeptaClient.submit()">Submit</button>
```

If you would like a to control the button for proceed with payment, then the ***preparePayment()*** call must contain a second *'buttonless'* parameter:

axeptaClient.preparePayment('←- PaymentID -->', 'buttonless');

Nome	Cognome
1234 1234 1234 1234 	
MM/AA	CVV

and in the script, the form can be posted with the method ***axeptaClient.submit();***

```
<!-- ESEMPIO DI PULSANTE SUBMIT -->  
<button id="submit" type="button" onClick="axeptaClient.submit()">Submit</button>
```



6.2 Smart layout personalization

A number of **style classes** which you can modify to personalize the **Smart** layout are listed below:

Class	Description
axepta-sdk-textfield-outlined	Class that defines the text input style with Outline layout.
axepta-sdk-textfield-outlined:focus:invalid	Class that defines the text input style when in the focus state and the value entered is invalid.
axepta-sdk-textfield-outlined:focus:valid	Class that defines the text input style when in the focus state and the value entered is valid.
axepta-sdk-textfield-outlined:valid:not(:focus):not(:placeholder-shown)	Class that defines the text input style when not in the focus state, the value entered is valid and the Placeholder of the field is not displayed.
axepta-sdk-button-contained	Class that defines the style of the button that executes the payment.
axepta-sdk-button-contained:disabled	Class that defines the style of the button that executes the payment, when it is in the disabled state. In particular, it is in this state if the values in the fields are invalid or the fields are empty.
axepta-sdk-button-contained:not(:disabled)	Class that defines the style of the button that executes the payment, when it is not in the disabled state.



The table of the classes for personalizing the “**inline**” layout is shown below:

Class	Description
axepta-sdk-textfield-outlined-inline	Class that defines the container of the entire Widget
axepta-sdk-flex-row-inline	Defines the style of the line.
axepta-sdk-textfield-outlined-noBorder-Pan	Defines the borderless style of the Pan input.
axepta-sdk-input-add-on-inline	Manages the position of the Pan input for the parent.
axepta-sdk-flex-col-inline	Sets the Pan input as a column.
axepta-sdk-pan-icon-inline	Container of the pan icon.
axepta-sdk-input-add-on-item-inline	Manages the position of the Pan icon, for the parent.
axepta-sdk-flex-row-inline-cvv	Container for the expiry and cvv inputs.
axepta-sdk-flex-col-left-inline	Manages the position to the left of the expiry input, for the parent.
axepta-sdk-textfield-outlined-noBorder	Defines the borderless style of the expiry input.
axepta-sdk-pan-icon-inline-cvv	Container for the cvv input and its icon.
axepta-sdk-flex-col-right-inline	Manages the position to the right of the cvv input, for the parent.
axepta-sdk-textfield-outlined-noBorder-cvv	Defines the borderless style of the cvv input.
axepta-sdk-input-add-on-item-inline-cvv	Defines the position of the cvv icon for the parent.



7 In-App checkout (SDK iOS)

7.1 Functional Requirements

iOS version supported: from 11.0 to 14.2

Xcode 12

7.2 Introduction

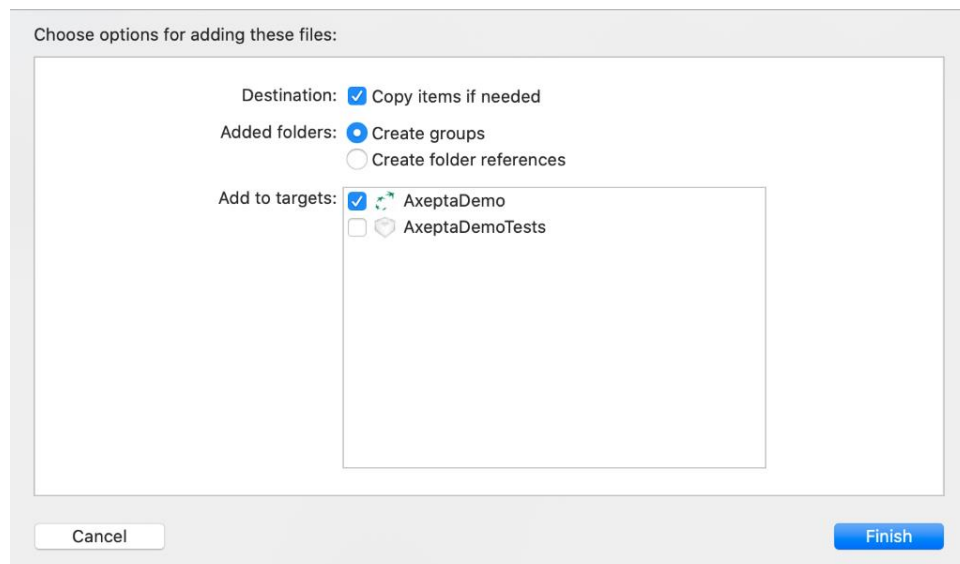
The AxeptaSDKClient framework allows payments to be made on the enabled schemes by:

- enabling a payment context managed completely by SDK
- integrating a graphic widget
- direct calls to make a payment on a determinate scheme

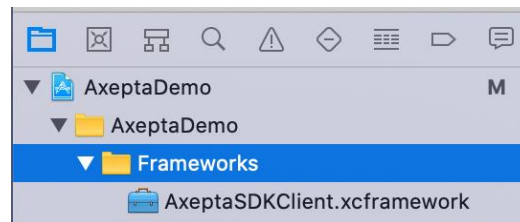
7.3 Adding SDK to the project

Follow the steps below to add SDK:

1. Open the project with Xcode and drag the AxeptaSDKClient.xcframework file onto the "project navigator", checking that the "copy items if needed" and "Add to targets" items are enabled:



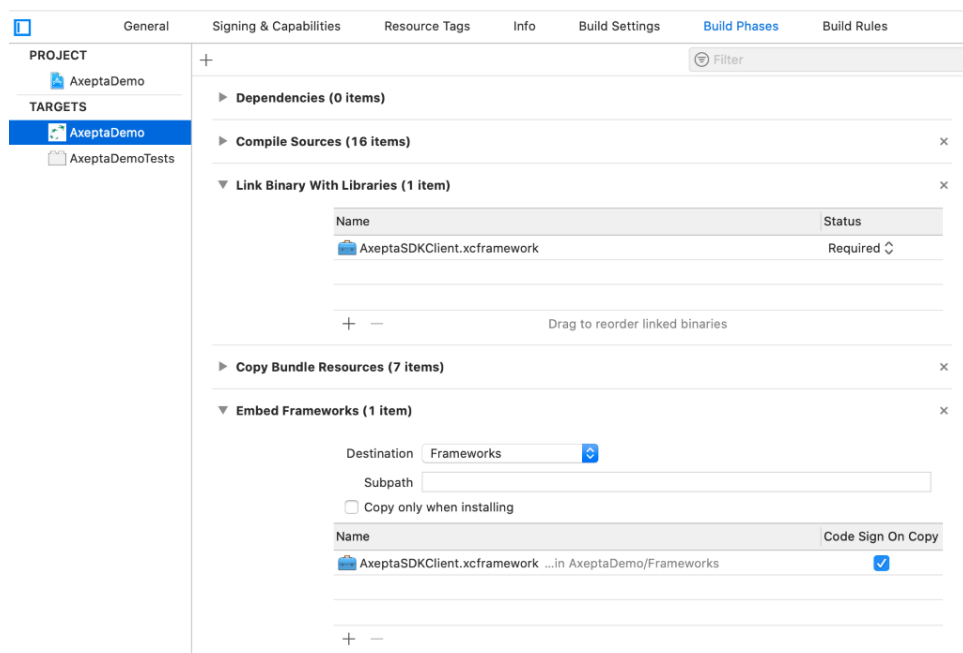
2. At this point the "project navigator" should display the framework:



3. Check that the framework is present on the “General” tab of the Target in the “Frameworks, Libraries and Embedded contents” section (with the “Embed & Sign” option enabled):

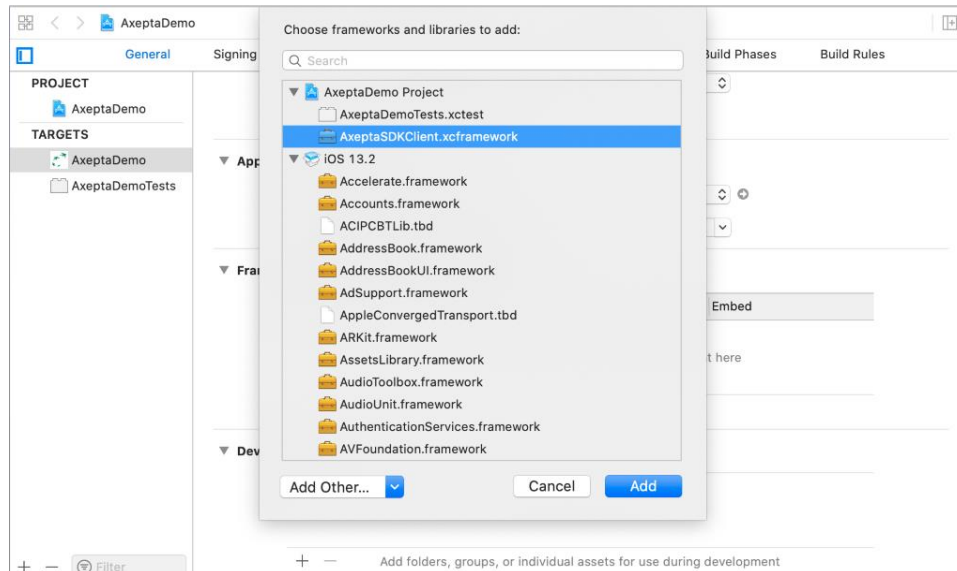


4. Check that the framework is present in the Build Phases, in the “Link Binary With Libraries” and “Embed Frameworks” sections:





5. If it is not present, select the + sign at the bottom of the “Frameworks, Libraries and Embedded contents” section and add it by hand, activating the “Embed & Sign” option if not enabled:



6. Check that the framework is present in the Build Phases as described in section 4.

7.4 SDK Configuration

Once the SDK has been added, an initial configuration is required.

1. In the AppDelegate file of the project, initialize the configuration by passing the endPoint for payments and the licenseKey to the SDK.
Optionally, it is possible to enable logs and print debug messages in console setting the "enableDebug" parameter to "true" (it defaults to false):

```
func application(_ application: UIApplication, didFinishLaunchingWithOptions launchOptions: [UIApplication.LaunchOptionsKey: Any]?) -> Bool {  
  
    Axepta.shared.configure(endPointUrl: "https://...", licenseKey: "XXXXXXX-XXXXXXX-XXXXXXX", enableDebug: true)  
    return true  
}
```

2. For debug purposes, it is possible to print in console the current installed SDK version number using

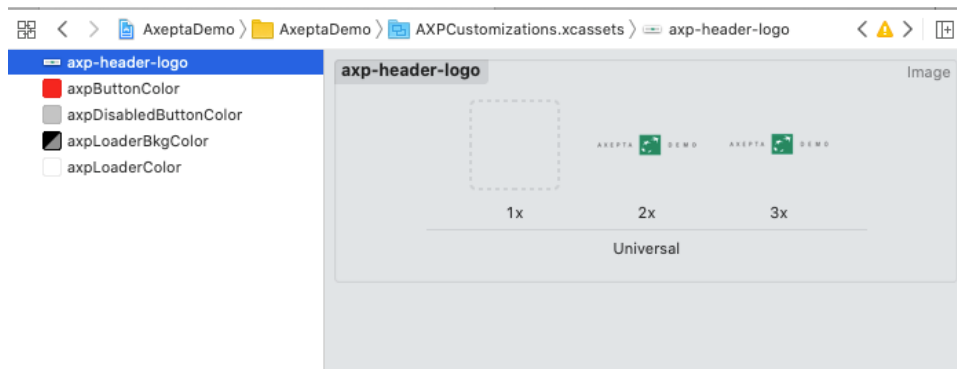


```
Import AxeptaSDKClient

. . .

Axepta.shared.showVersionNumber()
```

3. Drag the **AXPCustomizations.xcassets** folder into the project, always selecting the “Copy items if needed” and “Create groups” options. The Asset Catalog contains some Color Sets, used by the SDK for the graphic customizations and a logo, which will be used as the logo of the navigation bar on the screens managed by the SDK:



- **axpButtonColor:** is used to customize the colour of the payment and back buttons on the payment Widgets
- **axpDisabledButtonColor:** is used to customize the colour of the payment button where it is not enabled
- **axpLoaderBkgColor:** is used to customize the background colour of the loader shown by the SDK during the operations
- **axpLoaderColor:** is used to customize the colour of the loader shown by the SDK during the operations
- **axp-header-logo:** is used to customize the logo loaded by the navigation bar managed by the SDK. The image provided can be used as a template for formatting a custom image. If the logo is removed from the asset there will be no logo on the navigation bar. **N.B. The navigation bar only shows this logo using the integration methods by Payment Context or a direct call,**



setting the “present” parameter to true (as described in the next section).
If the “present” parameter is set to false, and the payment functions are loaded from a UIViewController inside a UINavigationController, the navigation bar used will be that of the application (and thus freely customizable).

7.5 SDK implementation

As mentioned in the introductory section, there are three different integration methods:

- Integration through the Payment Context: with this type of integration, the SDK will provide a selector to choose the payment method for every product, and will manage the subsequent payment phases autonomously;
- Integration through a Widget: some types of payment have a Widget embedded as a normal view to which an external payment button can be linked, or the default button can be used.
- Integration through a direct call: payments can be managed directly through a public interface that starts the payment on a specific scheme.

7.5.1 Payment Context Integration

This type of integration is used to display the list of payment methods enabled for every product, thus allowing the payment to be managed directly by SDK.

The Payment Context managed by the SDK can be linked to the tag on the object using the `createPaymentContext` function of the **Axepta** class by means of a Singleton.

Example of the code:

```
import AxeptaSDKClient
...

Axepta.shared.createPaymentContext("productId",    countryCode:    "IT",    present:
false)
```

The function accepts the following parameters in input:

- **itemId**: the ID of the payment, i.e. the output of the *initPayment* (see Par. Payment Initialization)
- **countryCode**: the ISO of the countryCode for the payment



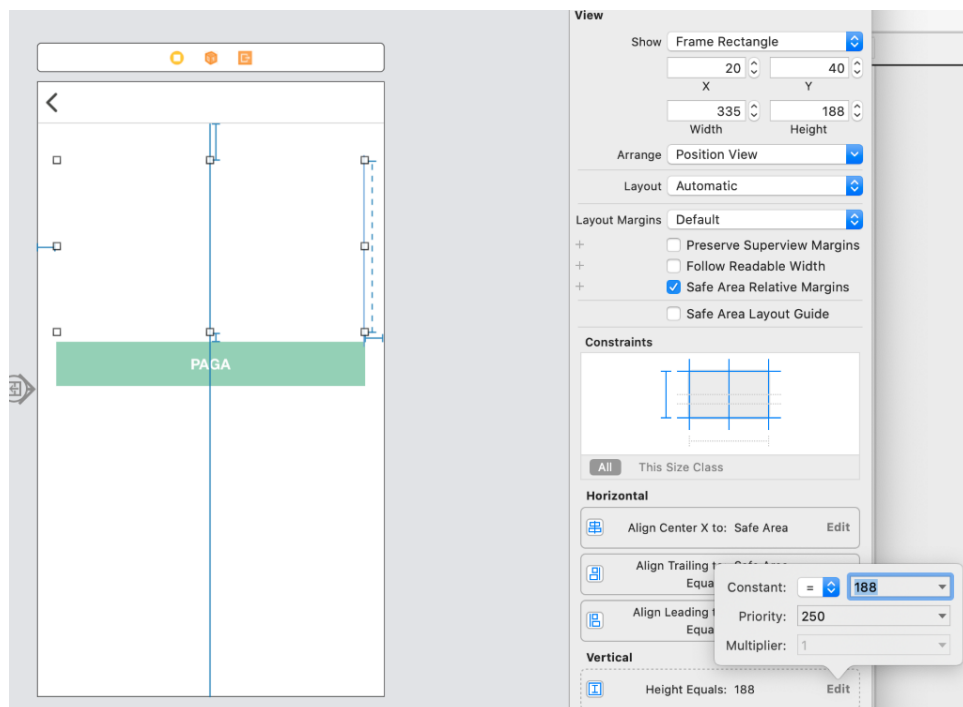
- **present:** a Boolean that enables the payment methods selector to be shown as modal (if set to true) or through a show on UINavigationController (if supported by the application, and setting this parameter to false).

7.5.2 Widget integration

There are three types of widget, all of the UIView type, which can be instantiated and embedded in any container UIView:

`AXPPhoneNumberWidget`, `AXPBankSelectorWidget`, `AXPCreditCardWidget`

A description of the characteristics and initialization methods of each of these widgets is provided below. Using, for example, a “widgetContainer” UIView in which to embed the widgets, a maximum height of 185px can be set on the container view with a low priority: in this way, the widget will automatically readapt according to the Mode set at the time of initialization (as will be described later on):



The *PaymentDelegate* protocol, common to all widgets, shows three functions that can be used to intercept the outcome of the transactions (any errors including “title” and “message” information, or conclusion of the process), and requires the declaration of the paymentID variable, which will be used by the widgets to make the payment:

```
public protocol PaymentDelegate {  
  
    var paymentID: String! { get set }  
  
    func onPaymentFinished()  
}
```



```
func onPaymentError(title:String?, message:String?)
func onPaymentCanceled()

}
```

Before any Widget can be initialized, the SDK must be imported:

```
import AxeptaSDKClient
```

AXPCreditCardWidget

Can be used for payments made using a Credit Card. This Widget in particular can be configured in 3 different versions, according to the mode with which it is initialized:

Form layout for .default mode: Two rows of text fields. The first row contains 'NOME' and 'COGNOME'. The second row contains 'NUMERO CARTA' with a card icon. The third row contains 'MM/AA' and 'CVV' with a card icon. Below the fields is a red button with a left arrow, the text 'PAGA', and a right arrow.

Mode: .default

Form layout for .compact mode: Two rows of text fields. The first row contains 'NUMERO CARTA' with a card icon. The second row contains 'MM/AA' and 'CVV' with a card icon. Below the fields is a red button with a left arrow, the text 'PAGA', and a right arrow.

Mode: .compact

Form layout for .inline mode: Two rows of text fields. The first row contains 'NOME' and 'COGNOME'. The second row contains 'NUMERO CARTA' with a card icon. The third row contains 'MM/AA' and 'CVV' with a card icon. Below the fields is a red button with the text 'PAGA' and a right arrow.

Mode: .inline

For this widget, the SDK offers three types of view; “**default**”, “**compact**” and “**inline**”, so as to satisfy the layout needs of the developer:

- “**default**” mode uses a height of 185px;
- “**compact**” mode uses a height of 142px;
- “**inline**” mode uses a height of 84px.

Initialization:

```
var paymentWidget = AXPCreditCardWidget(viewMode: .defaultMode,
                                          lang: "it",
                                          borderRadiusButton: 2,
                                          borderRadiusTextFields: 3,
                                          delegate: self)
```



- **viewMode:** is of the [Modality](#) type, an enum shown by the SDK, which can be set to the following values:
 - [.defaultMode](#)
 - [.compactMode](#)
 - [.inlineMode](#)
- **lang:** defines the language (according to the ISO639-1 standards): IT, EN, FR. If the parameter passed does not correspond to one of the languages managed by the widget, English will be used as the default language;
- **borderRadiusButton:** rounds the borders of the payment button;
- **borderRadiusTextField:** rounds the input fields;
- **delegate:** the object compliant with the *PaymentDelegate* protocol.

Using the **inline** layout (which has no payment button embedded in the component), the **payNow(itemId: String)** function can be activated directly, to manage the payment by customized interaction:

```
func payNow(itemId: String)
...
```

The itemId is the ID of the payment, i.e. the output of the *initPayment* (see Par. Payment Initialization), a required variable of the **PaymentDelegate** protocol.

Example of the code:

```
import UIKit
import AxeptaSDKClient

class PaymentViewController: UIViewController {

    @IBOutlet weak var widgetContainer:UIView!
    @IBOutlet weak var payButton: UIButton!

    var paymentWidget : AXPPaymentWidget?
    var itemToPay: Item?
    var paymentID: String!

    override func viewDidLoad() {
        super.viewDidLoad()

        payButton.addTarget(self, action: #selector(payBtnTapped(_)), for:
        .touchUpInside)
        payButton.clipsToBounds = true
        payButton.layer.cornerRadius = 4.0
        payButton.isHidden = SDKSettingsManager.shared.sdkModality != .inlineMode

        guard let itemToPay = itemToPay else { return }
```



```
paymentID = itemToPay.itemId

/** SDK Widget Initialization */
paymentWidget = AXPCreditCardWidget(
    viewModel                : .defaultMode,
    lang                     : itemToPay.language.lowercased(),
    borderRadiusButton       : 2,
    borderRadiusTextFields   : 3,
    delegate                 : self)

paymentWidget!.attachTo(widgetContainer)
/*****/
}

@IBAction func payBtnTapped(_ sender: Any) {
    // Action associated with the external payment button in .inline mode
    guard let itemToPay = itemToPay else { return }
    self.paymentWidget!.payNow(itemId: itemToPay.itemId)
}

}

extension PaymentViewController: PaymentDelegate {

    // MARK: Payment Delegate
    func onPaymentFinished() {
        print(#function)
        self.navigationController?.popViewController(animated: true)
    }

    func onPaymentError(title: String?, message: String?) {

        print("\(#function) error:\(String(describing: message))")
        DispatchQueue.main.async { [weak self] in
            guard let self = self else { return }

            if let message = messaggio, let title = titolo {
                let alert = UIAlertController.init(title: title, message: message,
preferredStyle: .alert)
                alert.addAction(UIAlertAction.init(title: "OK", style: .default,
handler: { [weak self] (action) in
                    self?.navigationController?.popViewController(animated: true)
                })
                self.present(alert, animated: true, completion: nil)
            }
        }
    }

    func onPaymentCanceled() {
        print(#function)
        self.navigationController?.popViewController(animated: true)
    }

}
```




7.5.3 Integration by direct call on a specific payment method

The last integration mode is the one that enables the use of the direct payment method from the `AXPPaymentManager` class:

```
public func executeDirectPaymentWith(itemId: String,
                                     circuit: Circuit,
                                     language: String,
                                     countryCode: String?,
                                     present: Bool = true)
```

- **itemId:** the ID of the payment, i.e. the output of the *initPayment* (see Par. Payment Initialization);
- **circuit:** an enum of the **Circuit** type, which can be set to: `.creditCard`, `.bancomatPay`, `.satispay`, `.aliPay`, `.weChat`, `.myBank`;¹
- **language:** defines the language (according to the ISO639-1 standards): IT, EN, FR. If the parameter passed does not correspond to any of the languages managed by the widget, English will be used as the default language;
- **countryCode:** the country code of the product to be paid (mandatory in payments on the Ali Pay, We Chat, Apple Pay)² payment method. The default setting is "IT";
- **present:** a Boolean, set by default to true, to determine whether the screen that includes the graphic widgets is to appear as modal (if set to true) or as a show on UINavigationController (if present in the application integrating the SDK, and if the value is set to false).

Using this method, the payment can be triggered directly by interaction with a button or from a gesture.

Example of the code:

```
AXPPaymentManager.shared.executeDirectPaymentWith(itemId: paymentId,
                                                    circuit: .creditCard,
                                                    language: "IT",
                                                    countryCode: "IT")
```

7.5.4 Setting the Credit Card view in direct payments

If you decide to use the credit card payment without initializing the graphic widget (that is, through the Payment Context or a direct call), the widget viewing method can be decided upon by passing the parameter directly to the singleton in the Acepta class.

Example of a code:

```
Acepta.shared.creditCardMode = .inlineMode
```

¹ If alternative payment methods have to be implemented, contact Acepta for support.

² If alternative payment methods have to be implemented, contact Acepta for support.



8 In-App checkout (SDK Android)

This SDK installation guide allows easy payment execution and a series of pre-set screens to meet different graphic needs, with the possibility of creating custom elements and the direct use of payment methods. The SDK naturally provides implementing developers the ability to capture **SUCCESS** and **FAILURE** events.

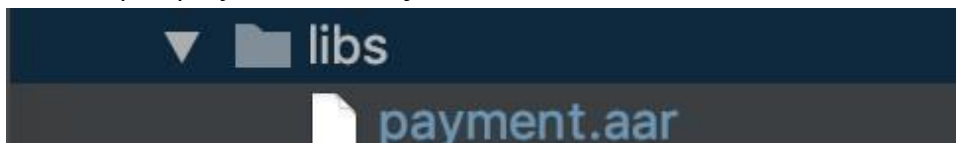
8.1 Minimum Requirements

To be able to integrate the SDK correctly, Android 6.0 (API 23) or a later version is required. The SDK has been made compatible with androidX, so the versions supported are API 23 and later.

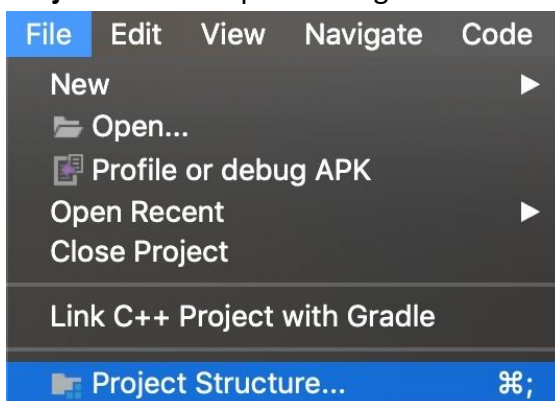
8.2 Adding dependency for SDK

Follow the steps below to add dependencies for the SDK:

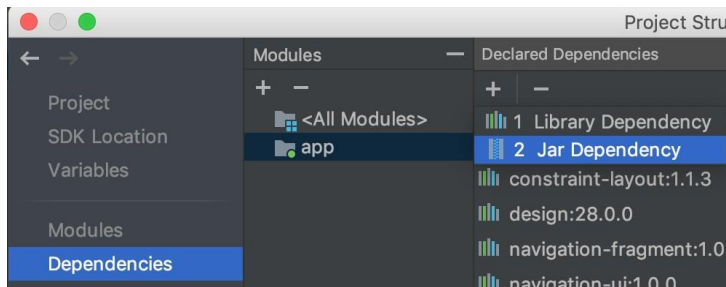
1. Take the **file.aar** of the SDK:
2. On the open project, enter **Project** view mode in **Android Studio**:



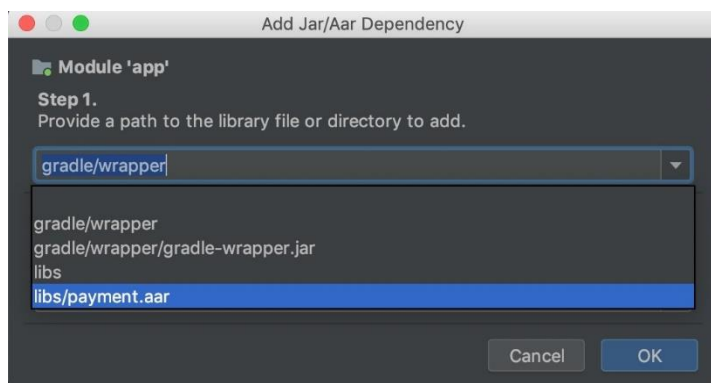
3. Locate the **libs** folder (within the app) and copy the file from the previous step into it:
4. It differs from one operating system to another, in the case of Mac OS X, open the **Project Structure** panel using the **File** button:



5. Select **Dependency**, under **Modules** select **app**, then press the **+** button and select **Jar Dependency**:



6. Select the item with the name of the SDK from the first drop-down menu. The second drop-down menu should be left with the default item, **implementation**, then press the **OK** button:



7. A new item for the library has now appeared, as can be seen from the image below. To finish correctly adding the dependency, press the **APPLY** button below and then press **OK**:



8. To verify that you have entered the dependency successfully, enter the **build.gradle(Module: App)** file and check that the implementation has been added correctly.



8.3 SDK configuration

Having added the dependency to the SDK, an initial configuration is necessary:

- Invoke the **ApiService.setEndPointAndLicence()** method, which requires two parameters:
 - **endpoint**, the endpoint to be contacted to make the executive call to make the payment. Parameter type → **String**
 - **licence**, the x-licence-api provided (of the In-App SDK Android) type. Parameter type → **String**
 - **enableDebug**, enable logs and print debug messages in console (default is false)
- Invoke the **ApiService.paymentConfiguration()** method, this method requires in input:
 - **paymentId**, the ID of the payment received from the init call made previously. Parameter type → **String**
 - **countryCode**, ISO code of the countryCode of the payment → **String**
- For debug purposes, it is possible to print in console the current installed SDK version number using the showVersionSdk function

Utils.showVersionSDK()

8.4 Graphic personalization

The colours of buttons, backgrounds and radii of buttons or text fields can be personalized. To be able to customize these parameters, you will have to enter the same keys used in the SDK containing the desired value in the color.xml file (for the colours), dimen.xml (for the size), drawable folder (for the images) of your project:

- Colours:
 - **background_color**: used to change the background colour of widgets
 - **button_background_default_color**: used to personalize the background colour of the buttons
 - **btn_color_enable**: background colour of the button when it is enabled
 - **btn_color_disable**: background colour of the button disabled
 - **background_loader**: background colour of the screen of the loader
 - **input_field_strokes_color**: colour of the borders of the editable fields
 - **input_field_background_color**: background colour of the EditText
 - **button_pay_text_color**: colour of the payment button text
 - **loader_color**: colour of the progress bar of the loader screen



- **item_selector_color**: background colour of the selector item
- Size
 - **default_edit_text_radius**: radius of editable field borders
 - **default_button_border_radius**: radius of button borders
- Drawable:
 - **logo**: logo loaded from the toolbar managed by the SDK. The logo will only be visible in the PaymetSelectorActivity integration mode. If no logo is set, the navigation bar will appear blank

8.5 SDK Integration

There are three types of SDK integration:

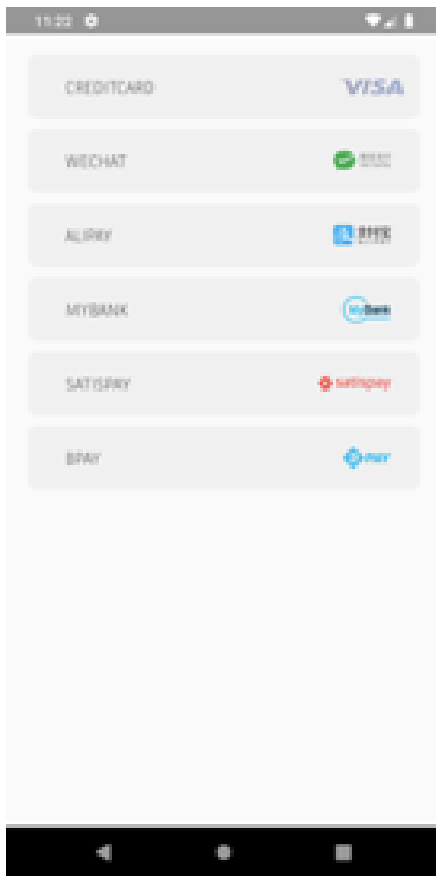
- Integration by **PaymentSelectorActivity**: in this case, the SELECTOR mode will be displayed, enabling the user to choose the type of payment and the subsequent phases of the payment will be managed automatically by the SDK.
- Integration by **FragmentPayment**: when the type of circuit to be used to make the payment is passed, the View for the selected mode to which an external button can be linked for the payment will be displayed or the default view can be used.
- Integration by direct calls: methods for starting payments directly on a specific scheme have been shown

8.6 PaymentSelectorActivity Integration

This type of integration enables the list of enabled payment methods for every product to be viewed. Once the payment method has been selected, the subsequent phases of the payment will be managed directly by SDK.



AXEPTA
BNP PARIBAS



To be able to integrate this mode, an intent will have to be executed in **PaymentSelectorActivity**

Java

```
Intent intent=new Intent(context,PaymentSelectorActivity.class)
startActivity(intent)
```

Kotlin

```
val intent = Intent(context, PaymentSelectorActivity::class.java)
startActivity(intent)
```



8.7 FragmentPayment Integration

8.7.1 Implementation in an Activity

The implementation of the elements for the effective use of this SDK requires a few steps both on the Java side and on the xml resources (layout) side).

Here are the steps necessary for the correct implementation within an Activity:

1. Instantiate a **ResultExecutePaymentCallback** object, which will be our callback.

```
private FragmentPayment.ResultExecutePaymentCallback resultExecutePaymentCallback;  
...
```

2. Create a method, such as *initCallBackSdkPayment*. Then override the methods needed to capture the **SUCCESS** and **FAILURE** outcomes that will be generated by the SDK.

```
Java  
private void initCallBackSdkPayment() {  
    resultExecutePaymentCallback = new FragmentPayment.ResultExecutePaymentCallback() {  
        @Override  
        public void onExecuteSuccess() {  
            Toast.makeText(getApplicationContext(), "Success", Toast.LENGTH_LONG).show();  
        }  
        @Override  
        public void onExecuteFailure() {  
            Toast.makeText(getApplicationContext(), "Failure", Toast.LENGTH_LONG).show();  
        }  
    };  
}  
...  
  
Kotlin  
val resultExecutePaymentCallback = object : ResultExecutePaymentCallback {  
    override fun onExecuteSuccess() {  
    }  
    override fun onExecuteFailure(reason: String?) {  
    }  
}
```



3. Invoke the *initCallBackSdkPayment* method immediately after **setContentView** within the onCreate activity.

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    initCallBackSdkPayment();
    ...
}
```

4. Create a **FragmentPayment** (a **Fragment**) object instance, which will process all the information entered into it by the user and if positive, proceed with the payment, otherwise it will return an error signal. Below are the elements that make up the **FragmentPayment** object constructor.

- a. **paymentType**: accepted values [**CREDIT_CARD** - **SATISPAY** - **BANCOMATPAY** - **MY_BANK** - **WECHAT** - **ALIPAY**]¹. The payment method indicated will be shown according to the value shown. Parameter type -> **PaymentType**
- b. **viewType**, accepted values [**DEFAULT** - **COMPACT** - **INLINE**]. This will generate a different view according to the value indicated. The **INLINE** view allows use of the direct call integration mode, thus giving the possibility of using custom components. Parameter type -> **ViewType**.
- c. **resultExecutePaymentCallback** returns the callback of the *executeCall()* method (defined in the Custom button – Payment section)

```
Java

final FragmentPayment = new FragmentPayment(

    FragmentPayment.PaymentType.CREDIT_CARD,

    ViewType.COMPACT,

    resultExecutePaymentCallback);
```

¹ If alternative payments method have to be implemented, contact Axepta for support.



```
...  
  
Kotlin  
  
val fragmentPayment = FragmentPayment(  
  
    FragmentPayment.PaymentType.CREDIT_CARD,  
  
    ViewType.COMPACT,  
  
    resultExecutePaymentCallback, buttonStateCallback  
  
)
```

5. **Add the Fragment:** The following image shows a sequence of standard commands for adding a **Fragment** within a layout

```
FragmentManager = getSupportFragmentManager();  
  
FragmentManager.beginTransaction().beginTransaction();  
  
fragmentTransaction.add(R.id.container,fragmentPayment).addToBackStack(null).commit();  
  
...
```

6. in this case a **FrameLayout**

```
<FrameLayout  
  
    android:id="@+id/container"  
  
    android:layout_width="match_parent"  
  
    android:layout_height="wrap_content"  
  
    android:layout_gravity="center_vertical|center_horizontal">  
  
</FrameLayout>
```



8.7.2 Implementation in a Fragment

For implementation within a Fragment, all the explanations above are valid. It is best to implement this by overriding the `onViewCreated` method and entering all of the instructions indicated within the latter. The only change to be made is on point “5” of section 2.12.7.1. The code you need to create a childFragment to integrate `FragmentPayment` within your Fragment is indicated below:

```
FragmentTransaction transaction = getChildFragmentManager().beginTransaction();
transaction.replace(R.id.container, fragmentPayment).commit();
```

8.8 Direct call integration

The possibility of including additional graphic elements, for example, a button that respects all the UI and UX lines that are being followed in your project, has been left.

The appropriate method will have to be invoked according to the payment method:

- **CREDIT_CARD**: the **`executeCall()`** method will have to be invoked; it is a public method that can be invoked using the **FragmentPayment** object. Below is an example image showing how to implement an **Activity** side button and how to create the respective **onClickListener** to allow the method indicated above to be invoked. On the XML layout side, there will be a personalized button below the **FrameLayout** created previously:

```
Button customPayButton = findViewById(R.id.custom_pay_button);
customPayButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        fragmentPayment.executeCall();
    }
});
...
```



8.9 Examples of the code

FragmentPayment integration

```
public class MainActivity extends AppCompatActivity {
    private FragmentPayment.ResultExecutePaymentCallback resultExecutePaymentCallback;

    @Override
    protected void onCreate(Bundle savedInstanceState) {

        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        initCallBackSdkPayment();

        final FragmentPayment fragmentPayment = new FragmentPayment(
            FragmentPayment.PaymentType.CREDIT_CARD,
            ViewType.COMPACT,
            resultExecutePaymentCallback
        );

        FragmentManager fragmentManager = getSupportFragmentManager();
        FragmentTransaction fragmentTransaction = fragmentManager.beginTransaction();
        fragmentTransaction.add(R.id.container, fragmentPayment).addToBackStack(null).commit();
        ;
        ApiService.setEndPointAndLicence("endpoint", "x-licence-key");
        ApiService.paymentConfiguration("paymentID", "countryCode", "amount");

        Button customPayButton = findViewById(R.id.custom_pay_button);
        customPayButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                fragmentPayment.executeCall();
            }
        });
    }

    private void initCallBackSdkPayment() {
        resultExecutePaymentCallback = new FragmentPayment.ResultExecutePaymentCallback() {
            @Override
            public void onExecuteSuccess() {
                Toast.makeText(getApplicationContext(), "Success",
                    Toast.LENGTH_LONG).show();
            }

            @Override
            public void onExecuteFailure() {
                Toast.makeText(getApplicationContext(), "Failure",
                    Toast.LENGTH_LONG).show();
            }
        }
    }
}
```



```
};  
}  
}
```

PaymentSelectorActivity integration

```
public class MainActivity extends AppCompatActivity {  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_main);  
  
        ApiService.setEndPointAndLicence("endpoint", "x-licence-key");  
        ApiService.paymentConfiguration("paymentID", "countryCode", "amount");  
  
        Intent intent=new Intent(context,PaymentSelectorActivity.class)  
        startActivity(intent)  
  
    }  
}
```



AXEPTA
BNP PARIBAS

8.10 Examples of Layouts

CreditCard

Default

<input type="text" value="Nome"/>	<input type="text" value="Cognome"/>
<input type="text" value="1234 1234 1234 1234"/>	
<input type="text" value="MM/AA"/>	<input type="text" value="CVV"/>
<input type="button" value="PAGA"/>	

Compact

<input type="text" value="1234 1234 1234 1234"/>	
<input type="text" value="MM/AA"/>	<input type="text" value="CVV"/>
<input type="button" value="PAGA"/>	

Inline

<input type="text" value="1234 1234 1234 1234"/>	
<input type="text" value="MM/AA"/>	<input type="text" value="CVV"/>



9 Payment outcomes

After payment execution, as indicated in the *InitPayment* call, the outcome of the transaction is returned with two methods:

- **Frontend:** frontend outcome depends of the integration type
 - **Easy checkout:** after processing the payment, Axepta performs a REDIRECT to the url specified by the merchant in *redirect_successUrl* (or in *redirect_failureUrl* if the transaction has failed).
 - **Smart checkout and API checkout:** the render of the outcome will trigger an event on the web page with the iframe, which indicates the end of the transaction. The event is triggered with a script that invokes the `window.postMessage()` method.
That is:
`window.top.postMessage('axepta_SUCCESS_message', '*');`
if the transaction is concluded successfully,
`or window.top.postMessage('axepta_FAILURE_message', '*');`
if the transaction fails.
This event can be intercepted by a listener on the web page. In any case, the details on the transaction will only be on the backend.
 - **In-App checkout (Android) and In-App checkout (iOS):** please see respective paragraphs.
- **Backend:** Axepta makes a server to server call to the url specified by the merchant in *callback_url* parameter of the *initPayment* call

An example of the parameters returned is shown below (format JSON):

```
{
  "integration_type": "WIZARD",
  "integration_name": "Easy",
  "mid": "BNLP TEST ALIAS",
  "paymentId":
  "76b5806696f823837a030cbb2c708c6108ce79961bf0d123519a055d75bdb6a1",
  "instrument": "CREDITCARD",
  "operation_type": "PAYMENT",
  "amount": "0.01",
  "currency": "EUR",
  "language": "IT",
  "transaction_type": "PURCHASE",
  "addresses": [],
  "products": [],
  "notification": {
    "area_code": "+39",
    "name": "",
    "email": "",
    "smartphone": "+39 "
  },
}
```



```
"additional": [],  
"callback_url": "https://webhook.site/45cf1cbd-0885-4845-980c-  
9e750e104b01",  
"transactionAt": "2020-11-10T12:03:09.409Z",  
"card_brand_desc": "Visa",  
"service_type_desc": "Debit Card",  
"product_priority_code": "",  
"shopID": "jSf9ppMWy7ZNvw943iBVRcXcHKsQFVwq",  
"tid": "08000001",  
"transaction_status": "PG_000",  
"payInstrToken": null,  
"payCardToken": null,  
"maskedPan": "411111*****1111",  
"brand": "VISA",  
"transactionID": "3087001610827263",  
"card_expiration": "1023",  
"authCode": "125996",  
"xid": "MDAzMTU2NzQ5MzYxMjkzNzY3OTA=",  
"transaction_code": "PG_01010",  
"description_status": "TRANSAZIONE OK"  
}
```

IMPORTANT:

- *Verify* is a server to server call that can be used ONLY AFTER the callback, as an additional check of the status of the transaction
- If the callback is not received, this means that the customer has decided not to pay or has not managed to pay or, though less likely, that an error has occurred in processing the payment and that, AT THE SAME TIME, this error has not been received on the callback. These two cases can be managed with the merchant-side *Verify*, but it should be after a long period of time (for example, an hour), in which, for example, the operation can be closed and the purchase indicated as “failed”.
- it is available a feature that, by default, if callback is not acknowledged (i.e. 200 OK by the merchant), then callback is resended again after 1 hour for a maximum of 24 times if a 200 OK is not received by Axepta. “1 hour” can be modified and “24 times” can be modified, if needed, contacting ecommerce support.
- The system has a recurring job. This job analyses any payment in pending status and manage to cancel it, if needed. In this case, a callback is sent to the merchant endpoint defined for that payment in order to notify that the payment is canceled. By default, the job is scheduled each 30 minutes and it takes payment older than 30 minutes, but those values can be modified if needed, contacting ecommerce support.



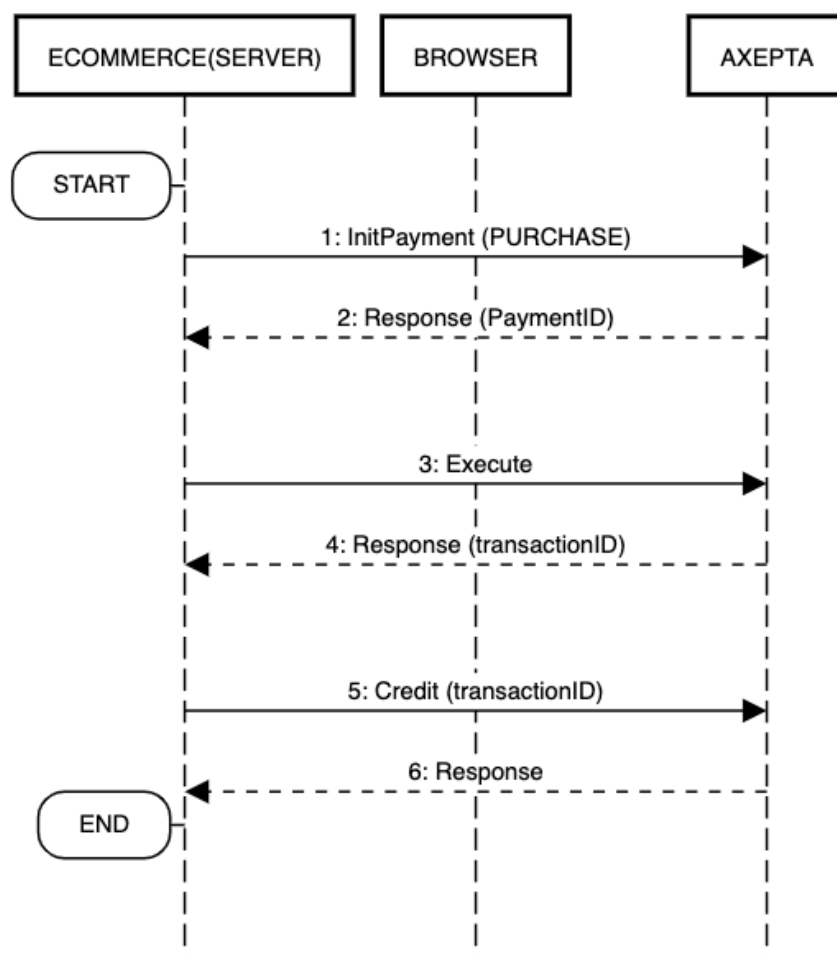
10 After the payment

This section describes the possible server to server calls that may be useful to the merchant after making the payment.

10.1 Credit

For a successful **PURCHASE** transaction, you can return an amount in order to return part or the entire amount paid into your wallet.

The payment ID to be included in input is that of the transaction already performed and on which the return is to be made.



- Initialize a PURCHASE type payment
- Perform an *execute*¹ to make the payment and a *transactionID* field will be returned in the response

¹ It is intended in general that the payment be made using one of the integration methods provided by Axepta: API, Easy, Smart, In-App.



- Perform a *Credit* using the *transactionID*, to return a credit of a certain amount on a payment¹

The specifications of the Return call for **PURCHASE** type transactions are shown below:

METHOD	POST
ENDPOINT	{{host server to server}}/api/v1/payments/credit
HEADERS	"Content-type": application/json "Authorization": Bearer <-- AccessToken --> "x-license-key": <-- API License Key -->
(BODY) PARAMETERS	{ "paymentId": "< -- payment ID -->", "transactionID": "3079887950480832", "amount": "2.36" }
RESPONSE	{ "maskedPan": "411111*****1111", "brand": "VISA", "authCode": "113973", "card_expiration": "1023", "mid": "a", "amount": "10.01", "currency": "EUR", "tid": "08000001", "paymentId": "<--PAYMENTID-->", "transactionAt": "2019-11-29T15:56:22.561Z", "operation_type": "CREDIT", "shopID": "hUiCdUtkvpNel9nM16v7NhtMXKfA4zx6", "transaction_status": "PG_000", "transaction_code": "01010", "description_status": "TRANSACTION OK", "transactionID": "3079905910425225", "pendingAmount": "5455" }
RESPONSE KO	{ "mid": "a", "amount": "10.01", }

¹ A return (Credit) of a transaction charged previously (Confirm) can also be performed. For further details, see the relevant section. Note that the amount to be returned must be consistent with the amount of the Purchase or Confirm performed previously.



	<pre>"currency": "EUR", "tid": "08000001", "paymentId": "<--PAYMENTID-->", "transactionAt": "2019-11-29T15:56:22.561Z", "operation_type": "CREDIT", "shopID": "hUiCdUtkvpNeI9nM16v7NhtMXKfA4zx6", "transaction_status": "PG_001", "transaction_code": "00001", "description_status": "Generic error." }</pre>
ERROR RESPONSE	<pre>{ "code": 1118, "message": "The data are necessary." }</pre>



10.1.1 Example of Java Unirest

```
HttpResponse<String> response = Unirest.post("https://pay-test.axepta.it/api/v1/payments/credit")
    .header("Content-Type", "application/json")
    .header("x-license-key", "XXXXXXX-0ERMYE0-MP683C5-9G0Q976")
    .header("cache-control", "no-cache")
    .body("{\n\t\"paymentId\":\n\t\"c5121109fd86460de50c33526ab7cc07:20454330610d34d156698b12b070cef7be4b948219ffbf1a15cff30ab32a94f93\",\n\t\"transactionID\": \"3079887950480832\",\n\t\"amount\": \"1.00\"\n}");
    .asString();
```

10.1.2 Example of PHP Http Request

```
<?php

$request = new HttpRequest();
$request->setUrl('https://pay-test.axepta.it/api/v1/payments/credit');
$request->setMethod(HTTP_METH_POST);

$request->setHeaders(array(
    'cache-control' => 'no-cache',
    'x-license-key' => 'XXXXXXX-0ERMYE0-MP683C5-9G0Q976',
    'Content-Type' => 'application/json'
));

$request->setBody('{
    "paymentId":
    "c5121109fd86460de50c33526ab7cc07:20454330610d34d156698b12b070cef7be4b948219ffbf1a15cff30ab32a94f93",
    "transactionID": "3079887950480832",
    "amount": "1.00"
}');

try {
    $response = $request->send();

    echo $response->getBody();
} catch (HttpException $ex) {
    echo $ex;
}
```



10.1.3 Example of Node Request

```
var request = require("request");

var options = { method: 'POST',
  url: 'https://pay-test.axepta.it/api/v1/payments/credit',
  headers:
    { 'cache-control': 'no-cache',
      'x-license-key': 'XXXXXXX-0ERMYE0-MP683C5-9G0Q976',
      'Content-Type': 'application/json' },
  body:
    {
      paymentId:
        'c5121109fd86460de50c33526ab7cc07:20454330610d34d156698b12b070cef7be4b948219ffbf1a15cff30ab32a94f99',
      transactionID: '3079887950480832',
      amount: '1.00' },
  json: true };

request(options, function (error, response, body) {
  if (error) throw new Error(error);

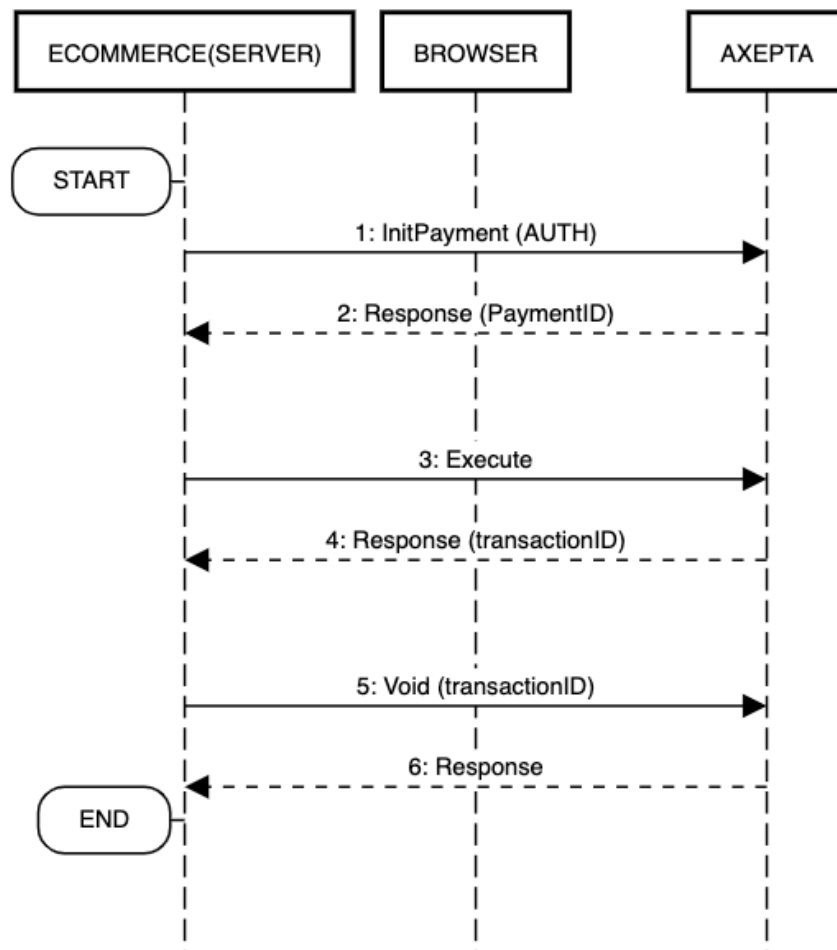
  console.log(body);
});
```



10.2Void

For a successful **AUTH** transaction, you can return an amount in order to return part or the entire amount paid into your wallet.

The payment ID to be entered in input is that of the transaction already performed and on which the return is to be made.



- Initialize an AUTH payment
- Perform an *execute*¹ to make the authorization and a *transactionID* field will be returned in the response
- Perform a *Void* using the *transactionID*, to return a credit of a certain amount on an already authorized payment²

¹ It is intended in general that the payment be made using one of the integration methods provided by Axepta: API, Easy, Smart, In-App.

² Note that the amount to be returned must be consistent with the remaining preauthorized amount, for example, if a partial Confirm has been performed previously



The specifications of the Return call for an **AUTH** transaction are shown below:

METHOD	POST
ENDPOINT	{{host server to server}}/api/v1/payments/void
HEADERS	"Content-type": application/json "Authorization": Bearer <-- AccessToken --> "x-license-key": <-- API License Key -->
(BODY) PARAMETERS	{ "paymentId": "< -- Payment ID -->", "transactionID": "3079887950480832", "amount": "2.36" }
RESPONSE	{ "maskedPan": "411111*****1111", "brand": "VISA", "authCode": "727958", "card_expiration": "1023", "mid": "a", "amount": "0.01", "currency": "EUR", "tid": "08000001", "paymentId": "<--PAYMENTID-->", "transactionAt": "2019-11-29T16:01:45.686Z", "operation_type": "VOID", "shopID": "dQUoilwIBi3LRs4nrnbGelWjwfj4Zuhd", "transaction_status": "PG_000", "transaction_code": "01010", "description_status": "TRANSACTION OK" }
RESPONSE KO	{ "mid": "a", "amount": "10.01", "currency": "EUR", "tid": "08000001", "paymentId": "<--PAYMENTID-->", "transactionAt": "2019-11-29T16:01:45.686Z", "operation_type": "VOID", "shopID": "dQUoilwIBi3LRs4nrnbGelWjwfj4Zuhd", "transaction_status": "PG_001", "transaction_code": "00001", "description_status": "Generic error." }



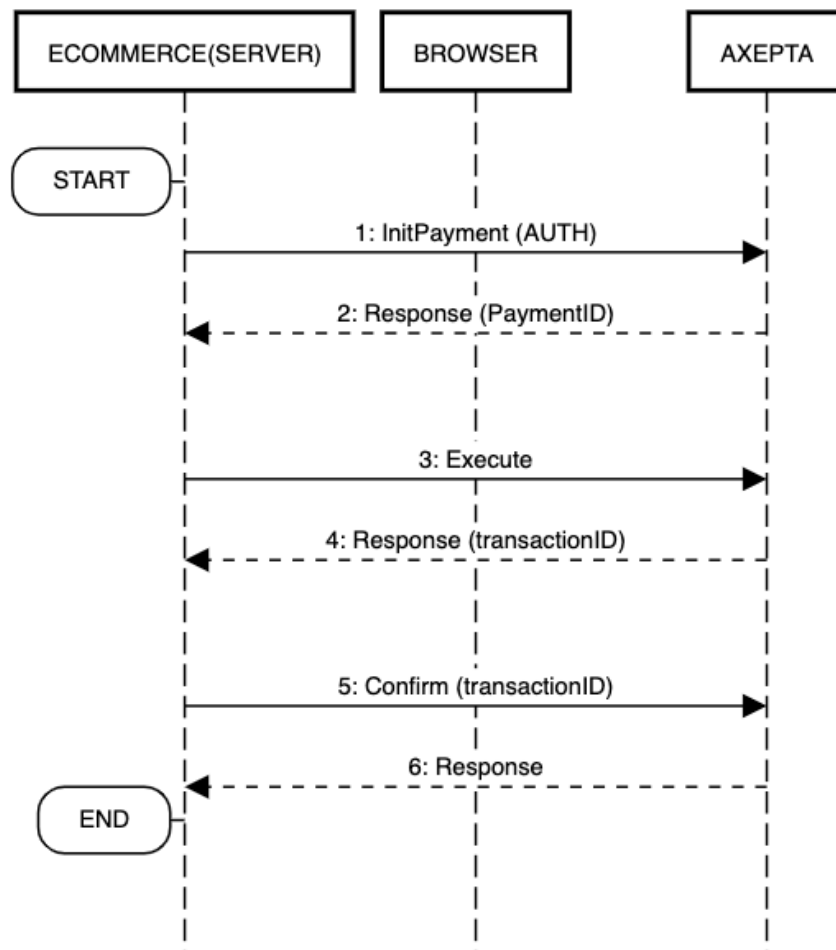
	}
ERROR RESPONSE	{ "code": 1118, "message": "The data are necessary." }



10.3 Confirm

For an **AUTH** transaction, you can confirm a specific amount in order to make the payment for a part or the entire amount.

The payment ID to be entered in input is that of the transaction already performed and on which the confirmation is to be made.



- Initialize an AUTH payment
- Perform an *execute*¹ to make the authorization and a *transactionID* field will be returned in the response
- Perform a *Confirm* using the *transactionID*, to perform the credit of a certain amount on an already authorized payment²

¹ It is intended in general that the payment be made using one of the integration methods provided by Axepta: API, Easy, Smart, In-App.

² Note that the amount to be confirmed must be consistent with the remaining preauthorized amount, for example, if a partial Void has been performed previously.



The specifications of the Confirm call for an **AUTH** transaction are shown below:

METHOD	POST
ENDPOINT	{{host server to server}}/api/v1/payments/confirm
HEADERS	"Content-type": application/json "Authorization": Bearer <-- AccessToken --> "x-license-key": <-- API License Key -->
(BODY) PARAMETERS	{ "paymentId": "< – payment ID -->", "transactionID": "3079887950480832", "amount": "2.36" }
RESPONSE	{ "maskedPan": "411111*****1111", "brand": "VISA", "authCode": "727958", "card_expiration": "1023", "mid": "a", "amount": "1.01", "currency": "EUR", "tid": "08000001", "paymentId": "<--PAYMENTID-->", "transactionAt": "2019-11-29T16:03:36.279Z", "operation_type": "CONFIRM", "shopID": "dQUoilwIBi3LRs4nrnbGelWjwfj4Zuhd", "transaction_status": "PG_000", "transaction_code": "01010", "description_status": "TRANSACTION OK", "transactionID": "3079905990598769", "pendingAmount": "8357" }
RESPONSE KO	{ "mid": "a", "amount": "10.01", "currency": "EUR", "tid": "08000001", "paymentId": "<-PAYMENTID->", "transactionAt": "2019-11-29T16:03:36.279Z", "operation_type": " CONFIRM ", "shopID": " dQUoilwIBi3LRs4nrnbGelWjwfj4Zuhd", "transaction_status": "PG_001", }



	<pre>"transaction_code": "00001", "description_status": "Generic error." }</pre>
ERROR RESPONSE	<pre>{ "code": 1118, "message": "The data are necessary." }</pre>

IMPORTANT: a return (*Credit*) of a previously charged transaction (*Confirm*) can also be made, in fact, the latter is as though it has become a *Purchase*. In this case, note must be taken of the *transactionID* returned by the *Confirm* and used for the *Credit*. Example:

First transaction (*Auth*):

- output
 - TransactionID=aaa

Confirm of the *Auth*:

- Input
 - TransactionID=aaa
- Output
 - TransactionID=bbb

Credit of the *Confirm*:

- Input
 - TransactionID=bbb

10.3.1 Confirm with automatic void of the residual

It is possible to partly confirm amount, and void the residual, using “voidEnabled” attribute set to true.

METHOD	POST
ENDPOINT	{{host server to server}}/api/v1/payments/confirm
HEADERS	<pre>"Content-type": application/json "Authorization": Bearer <-- AccessToken --> "x-license-key": <-- API License Key --></pre>
(BODY)	{



PARAMETERS VOID	<pre>"paymentId": "< – payment ID -->", "transactionID": "3079887950480832", "amount": "2.36", "voidEnabled": true }</pre>
RESPONSE VOID	<pre>{ "maskedPan": "411111*****1111", "brand": "VISA", "authCode": "727958", "card_expiration": "1023", "mid": "a", "amount": "1.01", "currency": "EUR", "tid": "08000001", "paymentId": "<--PAYMENTID-->", "transactionAt": "2019-11-29T16:03:36.279Z", "operation_type": "CONFIRM", "shopID": "dQUoilwIBi3LRs4nrnbGelWjwfj4Zuhd", "transaction_status": "PG_000", "transaction_code": "PG_01010", "description_status": "TRANSACTION OK", "transactionID": "3079905990598769", "voidedAmount": "8357" }</pre>
RESPONSE KO	<pre>{ "mid": "a", "amount": "10.01", "currency": "EUR", "tid": "08000001", "paymentId": "<-PAYMENTID->", "transactionAt": "2019-11-29T16:03:36.279Z", "operation_type": "CONFIRM", "shopID": "dQUoilwIBi3LRs4nrnbGelWjwfj4Zuhd", "transaction_status": "PG_001", "transaction_code": "PG_00001", "description_status": "Generic error." }</pre>
ERROR RESPONSE	<pre>{ "code": 1118, "message": "The data are necessary." }</pre>

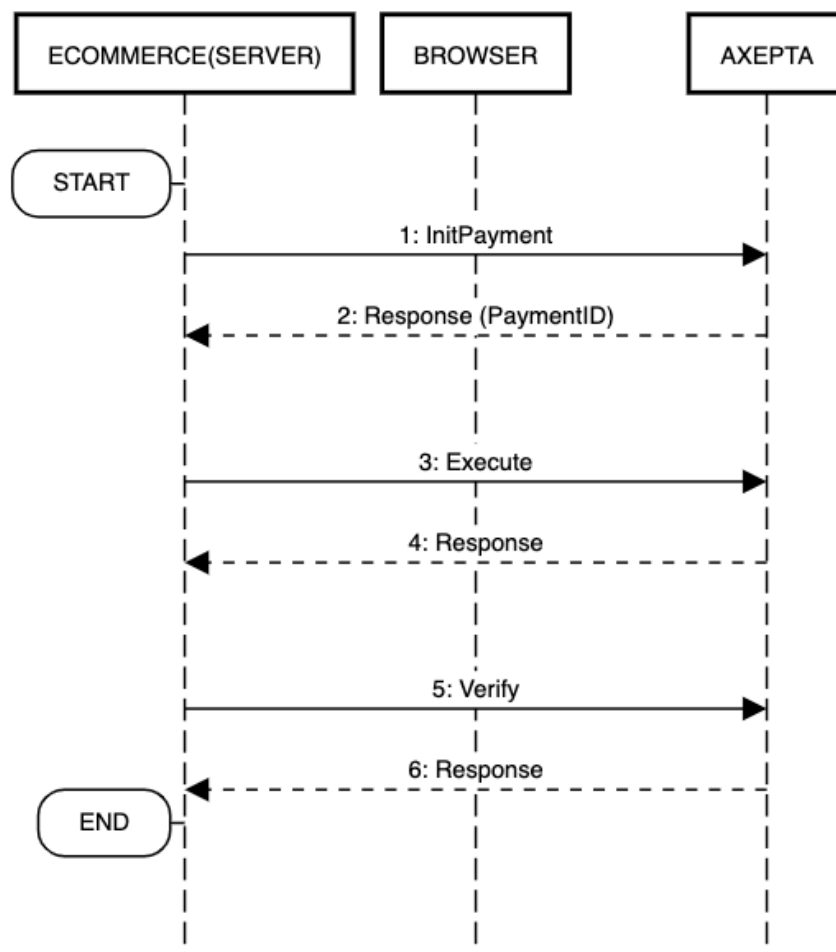
Where *voidedAmount* is the amount of the void operation, expressed in decimal.



10.4 Verifying a Transaction (verify)

In Server-To-Server mode, this method is used to verify the status of a payment. **The outcome of the transaction is indicated in the callback** (as explained in previous sections), **Verify is only to be used in some very special cases.**

The Verify function requires the payment ID (i.e. *PaymentID*) to be included in the parameters of the endpoint.



- Initialize a payment
- Perform an *execute*¹ to make the payment
- Perform a *Verify*, which will respond with the status of the payment.

¹ It is intended in general that the payment be made using one of the integration methods provided by Axepta: API, Easy, Smart, In-App.



The specifications of the payment verification call are shown below:

METHOD	GET
ENDPOINT	{{host}}/api/v1/payments/verify/{{PaymentID}}
HEADERS	"Content-type": application/json "Authorization": Bearer <-- AccessToken --> "x-license-key": <-- API License Key -->
RESPONSE	{ "maskedPan": "455777*****3335", "brand": "VISA", "authCode": "317676", "card_expiration": "1023", "mid": "012", "currency": "EUR", "tid": "08000001", "paymentId": "a27ca5d4ee424e2086c57d647fef12429af2f57c0943a082519a055d75bdb6a1", "transactionAt": "2020-02-13T12:44:16.315Z", "operation_type": "VERIFY", "shopID": "1jEyAMdH9hdj2jiAMCHxYysNOoATuW0I", "transaction_status": "PG_000", "transaction_code": "01010", "description_status": "TRANSACTION OK", "transactionID": "3077722880769169" }
RESPONSE KO	{ "mid": "a", "amount": "10.01", "currency": "EUR", "tid": "08000001", "paymentId": "<-PAYMENTID->", "transactionAt": "2020-02-13T12:44:16.315Z", "operation_type": "VERIFY", "shopID": "1jEyAMdH9hdj2jiAMCHxYysNOoATuW0I", "transaction_status": "PG_001", "transaction_code": "00001", "description_status": "Generic error." }
ERROR RESPONSE	{ "code": 1118, "message": "The data are necessary." }



10.5 Making one-click payments

One-click payments is an optional service that it could be activated on request. This service allows to store card data and use them for successive payments (without insert them again in the form).

The service consists of three phases:

1. Configuration
2. Card tokenization
3. Payment using token

10.5.1 Phase 1 - Configuration

Axepta will activate the tokenization feature on merchant request.

Merchant has to decide if wants:

- **Explicit** tokenization (only for Easy, Smart, In-app checkouts): in this scenario, it is the card holder that explicitly checks the corresponding checkbox in order to get the card tokenized for future payments.

A screenshot of a web form for explicit tokenization configuration. The form contains input fields for 'Nome' and 'Cognome', a card number field with the placeholder '1234 1234 1234 1234', and fields for 'MM/AA' and 'CVV'. Below these fields is a checkbox labeled 'Salva i dati di pagamento per i prossimi acquisti'. At the bottom, there is a large button labeled 'Paga 0,01 €' with left and right arrows. Below the button, a small text line reads: 'Cliccando sul tasto Paga dichiari di aver preso visione dell'[informativa sulla privacy](#)'.

A screenshot of a web form for implicit tokenization configuration. It features a card number field with the placeholder '1234 1234 1234 1234', fields for 'MM/AA' and 'CVV', and a checkbox labeled 'Salva i dati di pagamento per i prossimi acquisti'.

- **Implicit** tokenization: in this scenario, the card tokenization it is totally choiced by the merchant. The card holder could be advised by a message of the merchant in the ecommerce page or by a message in Axepta payment form (this message is configured by Axepta following merchant requirement). Implicit tokenization is also the only possible case for API Checkout.



AXEPTA
BNP PARIBAS

10.5.2 Phase 2 - Card tokenization

There are some parameters of the *initPayment* method, needed necessary to understand regarding tokenization:

- **“tokenize”** is a boolean (true/false) parameter. It is the “consensus” from card holder in order to tokenize the card. The usage of this parameter is the following:
 - *tokenize=true* in input of input of the *initPayment* if you are configured as **implicit** tokenization (this is valid for all types of checkouts)
 - No need to use *tokenize* parameter in case of **explicit** tokenization, because they will take the card holder’s consensus from the flag in the form (this is valid Easy, Smart or In-app checkouts, in fact for the API checkout the explicit tokenization does not have sense, because card holder gives the consensus always on the merchant website)
- **“payInstrToken”** is the name of the user wallet and it is also the ID connecting successive transactions. For example, it could be the customer ID on merchant side or customer email. It is in input of the *initPayment* and in output of the payment (i.e. execute and/or callback). It could be used in input for the first transaction, otherwise it would be randomly generated by Axepta. For successive transactions, it must be used in input on *initPayment*, otherwise first and successive transactions would not be connected each others. In successive transactions, it allows to show stored cards of the card holder/user.
- **“payCardToken”** is the card token. It is in input of the *initPayment* and in output of the payment (i.e. execute and/or callback). If the transaction is a “first transaction”, then *payCardToken* would be an output parameter generated by Axepta. If you are submitting a successive transaction in the “one-click payments”, then you can optionally use *payCardToken* as an input parameter. If you do, you are forcing the system to pay with that tokenized card instead of let the cardholder to choose.
- **“txIndicatorType”** is the type of tokenized transaction. It is an input of *initPayment*. In the case of a first transaction *txIndicatorType* **would not be specified**. In the case



of a successive transaction of type “one-click payments”, then *txIndicatorType*=UNSCHEDULED.

Following, there is an example of the *initPayment* of a first transaction (implicit tokenization case):

METHOD	POST
ENDPOINT	{{host server to server}}/api/v1/payment/initPayment
HEADERS	“Content-type”: application/json “Authorization”: Bearer <-- AccessToken --> “X-license-key”: <-- API License Key -->
(BODY) PARAMETERS	{ "transaction_type": "PURCHASE", "transaction_timeout": "30000", "tokenize": true, "payInstrToken": "test@test.it", "shopID": "shopIDprova12345", "currency": "EUR", "language": "IT", "amount": "84.58", }
RESPONSE	{ "code": 200, "message": "SUCCESSFULLY", "paymentID": <-- Payment ID --> }

Then, the card holder would execute the payment. This action could be done in different ways, it depends on which type of checkout is used by the merchant (Easy, Smart, etc)¹.

In addition, if implicit tokenization is disabled, it is up to the card holder the choice to tokenize the card or not.

If the transaction:

- is successfully, then the card is tokenized and VERIFIED. In this case, the user could make one-click payments without insert card data again.

¹ If you are using API Checkout, then you must specify “tokenize=true” in the *execute* method.



- is failed, then the card would be temporary tokenized as PENDING. In this case, merchant would receive *payInstrToken* but the user needs to insert again card data for future payments. In this case merchant will not receive *payCardToken*.

Followig, there is an example of success transaction (tokenized card), where *payCardToken* is randomly generated:

RESPONSE	<pre>{ "mid": "merchantID", "instrument": "CREDITCARD", "operation_type": "PAYMENT", "isHTML": false, "transactionAt": "2019-11-29T15:17:19.373Z", "tid": "08000001", "shopID": "AfqhuojN7LCJw6UstZMVoPwo2QGNX8N7", "transaction_status": "PG_000", "token": "nloS0bPqZq8F27wcH4a5LNoOd2XVM55v", "maskedPan": "411111*****1111", "brand": "VISA", "transactionID": "3079905680585024", "payInstrToken": "test@test.it", "payCardToken": "6UstZMVo6UstZPwo2QGNX", "authCode": "288380", "xid": "MDAzMzMzODI4MzMzMjIzOTU2Nzc=", "transaction_code": "01010", "description_status": "TRANSAZIONE OK" }</pre>
----------	---

In case of explicit tokenization, if the end user did not accept to save card data, then the output of the callback would contain "**payCardToken**": null

10.5.3 Phase 3 - Payment using token

Assuming the first payment and tokenizing are successfully, then the merchant could initiate a "one-click" payment. In order to do that, merchant has to call *initPayment* method with:

- *payInstrToken* of the user
- *payCardToken*, optionally if you want to force the payment with a tokenized card of the user instead to let the user choose which card on Axepta form
- *txIndicatorType=UNSCHEDULED*

METHOD	POST



ENDPOINT	{{host server to server}}/api/v1/payment/initPayment
HEADERS	"Content-type": application/json "Authorization": Bearer <-- AccessToken --> "X-license-key": <-- API License Key -->
(BODY) PARAMETERS	{ "transaction_type": "PURCHASE", "transaction_timeout": "30000", "payInstrToken": "test@test.it", "payCardToken": "6UstZMVo6UstZPwo2QGNX", "txIndicatorType": "UNSCHEDULED", "shopID": "shopIDprova12345", "currency": "EUR", "language": "IT", "amount": "84.58", }
RESPONSE	{ "code": 200, "message": "SUCCESSFULLY", "paymentID": <-- Payment ID --> }

Then, the card holder would execute the payment. This action could be done in different ways, it depends on which type of checkout is used by the merchant (Easy, Smart, etc)¹.

¹ If you are using API Checkout, then you must specify *payInstrToken* and *payCardToken* in the *execute* method. See Cap. API checkout



10.6 Recurring payments (scheduled by the merchant)

Recurring payments is an optional service that it could be activated on request. This service allows to store card data and use them for successive payments initiated by the merchant in certain period of times (for example a membership subscription each month).

In particular, here we describe the case in which the merchant wants to schedule the transactions.

Before to read this paragraph, you need to read carefully the Par. "Making one-click payments" in order to understand the tokenization topic.

The service consists of three phases:

1. Configuration
2. Card tokenization
3. Payment using token

10.6.1 Phase 1 - Configuration

Please read respective paragraph of Par. "Making one-click payments".

10.6.2 Phase 2 - Card tokenization

Please read respective paragraph of Par. "Making one-click payments". There is not any difference.

10.6.3 Phase 3 - Payment using token

Assuming the first payment and tokenizing are successfully, then the merchant could initiate a "recurring payment". In order to do that, merchant has to call *initPayment* method with:

- *payInstrToken* of the user
- *payCardToken*, optionally if you want to force the payment with a tokenized card of the user instead to let the user choose which card on Axepta form
- *txIndicatorType=RECURRENT*

METHOD	POST
ENDPOINT	{{host server to server}}/api/v1/payment/initPayment
HEADERS	"Content-type": application/json "Authorization": Bearer <-- AccessToken --> "X-license-key": <-- API License Key -->



(BODY) PARAMETERS	<pre>{ "transaction_type": "PURCHASE", "transaction_timeout": "30000", "payInstrToken": "test@test.it", "payCardToken": "6UstZMVo6UstZPwo2QGNX", "txIndicatorType": "RECURRENT", "shopID": "shopIDprova12345", "currency": "EUR", "language": "IT", "amount": "84.58", }</pre>
RESPONSE	<pre>{ "code": 200, "message": "SUCCESSFULLY", "paymentID": <-- Payment ID --> }</pre>

Then, the merchant would execute the payment. This action must be done with the API call called *execute*.



10.7 Deleting a tokenized card

There are different ways to delete a tokenized card:

1. By the card holder. Merchant makes an *initPayment* specifying *payInstrToken* parameter. Then, merchant calls a payment form (either Easy, Smart or In-app checkout) and shows it to the cardholder. If cardholder has previously tokenized one or more cards, then he will view a form like this

In this case, cardholder can delete the tokenized card(s) using the corresponding button (X)

2. By the merchant. Merchant can use a server to server method in order to delete a tokenized card on behalf of the cardholder. Then, Merchant is responsible of deleting a tokenized card. The specifications of the call are shown below:

METHOD	DELETE
ENDPOINT	{{host}}/api/v1/payments/card/←payInstrToken→/←payCardToken→
HEADERS	"Content-type": application/json "Authorization": Bearer <-- AccessToken --> "x-license-key": <-- licenza d'uso -->
RESPONSE	{ "code": "200" }



AXEPTA
BNP PARIBAS

11 Error codes

Call responses for status 200, i.e. those which were successful, were described previously.
For error codes, see the document:
"AXEPTA_ListaCodiciEsitoTransazione".



12 Testing environment information

General information for the test environment are provided below. Specific information for the test merchant is communicated separately by Axepta.

Reference HOST:

Checkout	Testing
API	https://pay-test.axepta.it
Easy	https://pay-test.axepta.it/sdk
Smart	https://pay-test.axepta.it/sdk
In-App (SDK iOS)	https://pay-test.axepta.it
In-App (SDK Android)	https://pay-test.axepta.it

The cards used for the tests are indicated below.

N.B. Where specified, the correct expiry date and/or CVV code must be indicated otherwise the transaction will fail.

PAN	Scheme	Expiry	Cvv	Enrstatus	Authstatus	Authorization outcome
455777333333335	Visa	-	-	Y	Y	OK
455777222222229	Visa	-	-	Y	A	OK
411111111111111	Visa	10/2023	-	N	-	OK
455500000000001	Visa	-	-	N	-	OK
4111111112225555	Visa	-	-	N	-	OK
401151444444116	VisaElectron	-	-	N	-	OK
401151999222222	VisaElectron	-	-	N	-	OK
400500000000007	VisaDebit	10/2023	-	N	-	OK
4005004455555556	VisaDebit	-	-	N	-	OK
543013222222226	Mastercard	-	-	N	-	OK
589353544444429	Mastercard	-	-	N	-	OK
5790640100000005	Mastercard	-	-	N	-	OK
5430131234567891	Mastercard	-	-	Y	N	KO
5548535889622125	Mastercard	-	-	N	-	OK
540117222222227	Mastercard	-	-	Y	Y	OK
543013222222226	Mastercard	-	-	N	-	OK
5548536000000126	Mastercard	-	-	Y	Y	OK
526492111111115	MastercardDebit	-	555	N	-	OK
5545910000000019	MastercardDebit	-	-	Y	N	KO
5020639451965933	Maestro	-	-	Y	Y	OK
5893535596092423	Maestro	-	-	Y	A	KO



where:

- *enrStatus* represents the status of registration of the card to the 3D Secure service
 - Y – Authentication available;
 - N – Holder not registered with the service;
 - U – Authentication not possible;
 - E – Error.
- *AuthStatus* represents the authorization outcome of the card with the 3D Secure service
 - Y – Authenticated;
 - A – Attempted authentication;
 - N – Holder not authenticated;
 - U – Authentication not possible.

The testing environment includes some test cases through which the behaviour of the solution can be simulated when some conditions that would invalidate the transaction occur. They respond to the following Pan / Amount combinations:

PAN	Amount	Expiry	Outcome	Description
455777333333335 540117222222227 455777222222229 5548536000000126 5020639451965933	101,00	any	PG_01045	Authorization denied
	102,00	any	PG_01058	Incorrect merchant code
	103,00	any	PG_01057	Invalid card
	104,00	any	PG_01086	Holder not enabled for this operation
	105,00	any	PG_01089	Frequency limit exceeded
	106,00	any	PG_01180	Stolen card
	107,00	any	PG_01038	Format error
	108,00	any	PG_01080	Contact issuer
	109,00	any	PG_01078	Suspected fraud
543013222222226	103,00	any	PG_01060	Insufficient funds
411111111111111	104,00	any	PG_01086	Holder not enabled for this operation
401151999222222	105,00	any	PG_01089	Frequency limit exceeded
400500000000007				
526492111111115				
540117222222227	> 200	any	PG_01060	Insufficient funds
526492111111115				
411111111111111	Any different from above	Any different from 10/23	PG_01018	Card expired
400500000000007				



12.1 Testing 3DS 2.x

If merchant is enabled to 3DS 2.x, then it needed to make some specific tests. In particular you can follow the below table in order to test some 3DS 2.x scenarios.

PAN	Scheme	Expiry	cvv	transStatus	Authorization outcome
4012001036275556	VISA	10/23	any	U	UNABLE
5255999999999992	MC	10/23	any	U	UNABLE
4012001038443335	VISA	10/23	any	Y	RISK BASED AUTHENTICATION (FRICTIONLESS)
5453010000073866	MC	10/23	any	Y	RISK BASED AUTHENTICATION (FRICTIONLESS)
5256103270096532	MC	10/23	any	Y	RISK BASED AUTHENTICATION (FRICTIONLESS)
4824983270096509	VISA	10/23	any	Y	RISK BASED AUTHENTICATION (FRICTIONLESS)
4012001036853337	VISA	10/23	any	N	REFUSED BY THE PAYMENT GATEWAY
5453010000073684	MC	10/23	any	N	REFUSED BY THE PAYMENT GATEWAY
4012001037141112	VISA	10/23	any	C	FULL 3DSecure (CHALLENGE) Fill with 111111
5453010000073205	MC	10/23	any	C	FULL 3DSecure (CHALLENGE) Fill with 111111



13 How to move to Production environment

The following activities must be carried out starting from the time at which Axepta and the merchant agreed on the release to Production:

- Change all the endpoints from <https://pay-test.axepta.it> to <https://pay.axepta.it>, i.e. follow the information given in the table below

Checkout	Testing	Production
API	https://pay-test.axepta.it	https://pay.axepta.it
Easy	https://pay-test.axepta.it/sdk	https://pay.axepta.it/sdk
Smart	https://pay-test.axepta.it/sdk	https://pay.axepta.it/sdk
In-App (SDK iOS)	https://pay-test.axepta.it	https://pay.axepta.it
In-App (SDK Android)	https://pay-test.axepta.it	https://pay.axepta.it

- Generate another Access Token, to be used in the Production environment
 - By accessing the URL <https://pay.axepta.it/access> from a browser session in “incognito mode”
 - Using the Username and Password provided by Axepta
 - Use this new Access Token for the Production environment
- Replace all the License Keys (i.e. the Server to Server key and each of the special keys for the single integration method chosen).



14 Merchant-side PCI data security information

The PCI DSS standard defines the compliance requirements that merchants must fulfil. The table below lists these requirements (Self Assessment Questionnaire SAQ and Report on Compliance RoC) for each type of technical integration solution that the merchant decides to implement.

Product	Technical solution	Description	Compliance (≤6,000,000 transactions/year)	Compliance (> 6,000,000 transactions/year)
Easy Checkout	Redirect SDK	The user is redirected to the AXEPTA payment page and the card data can be entered on the same page	SAQ A	RoC ^A
Smart Checkout	JS SDK	The card data entry form is presented by the merchant and the data is transmitted directly to the payment gateway without interaction with the merchant's server	SAQ A-EP	RoC ^{A-EP}
In-App Checkout	Mobile SDK	SDK for the integration of payment services on mobile applications	SAQ A	RoC ^A
API Checkout	Server to server	Card data is entered on the merchant page and the card data is managed by the merchant's server	SAQ D	RoC

Where, in principle, the criteria are as follows:

- SAQ A / RoC^A = the entire payment page is managed by Axepta
- SAQ A-EP / RoC^{A-EP} = The merchant's site does not historicize, process or transmit card data, but controls how the data is collected
- SAQ D / RoC = the merchant's site historicizes, processes or transmits card data



AXEPTA
BNP PARIBAS

AXEPTA BNP Paribas

pHey – CMS Plugins integration guide

v.1.1.0



Document versions

Version	Description of changes
1.0.0	First issue
1.1.0	CMS Compatibility Par. "From Test to Production", added paragraphs Other test cases in Par. "Credit Cards for demonstration use" Par. "Testing 3DS 2.x", added paragraph Par. "Credit, Void and Confirm", added paragraph



CONTENTS

1	Introduction	5
2	Service information	5
3	WooCommerce Plugin	6
3.1	Installation.....	6
3.2	Activation.....	6
3.3	Configuration.....	8
3.4	From Test to Production.....	11
4	PrestaShop 1.6.x Plugin	13
4.1	Installation.....	13
4.2	Activation.....	13
4.3	Configuration.....	14
4.4	From Test to Production.....	16
5	PrestaShop 1.7.x Plugin	17
5.1	Installation.....	17
5.2	Activation.....	17
5.3	Configuration.....	17
5.4	From Test to Production.....	20
6	Magento 1.9.x Plugin.....	21
6.1	Installation.....	21
6.2	Configuration.....	21
6.3	Assistance request.....	24
6.4	From Test to Production.....	25
7	Magento 2.x Plugin.....	26
7.1	Installation.....	26
7.2	Activation.....	27
7.3	Configuration.....	27
7.4	Upgrading	29
7.5	Assistance request.....	30
7.6	From Test to Production.....	31
8	Credit cards for demonstration use.....	32
8.1	Testing 3DS 2.x	34
9	Credit, Void and Confirm	35
10	Merchant side PCI data security information	36



AXEPTA
BNP PARIBAS



AXEPTA
BNP PARIBAS

1 Introduction

This document describes how to install and configure the e-commerce plugins released by Axepta. Plugins are software packages that facilitate integration with Axepta's pHey solution.

2 Service information

This integration guide contains all the information necessary for the integration of your website with Axepta's e-commerce platform.

For any further information, simply write to us at:

ecommercesupport@axeptamail.com or contact us at (+39) 060 070 by selecting option 1 and then 4.

3 WooCommerce Plugin

Compatibility	WordPress 5.3.2 and higher / Woocommerce 3.9.1 and higher Other technology requirements are specified by the CMS itself, depending on the version.
----------------------	---

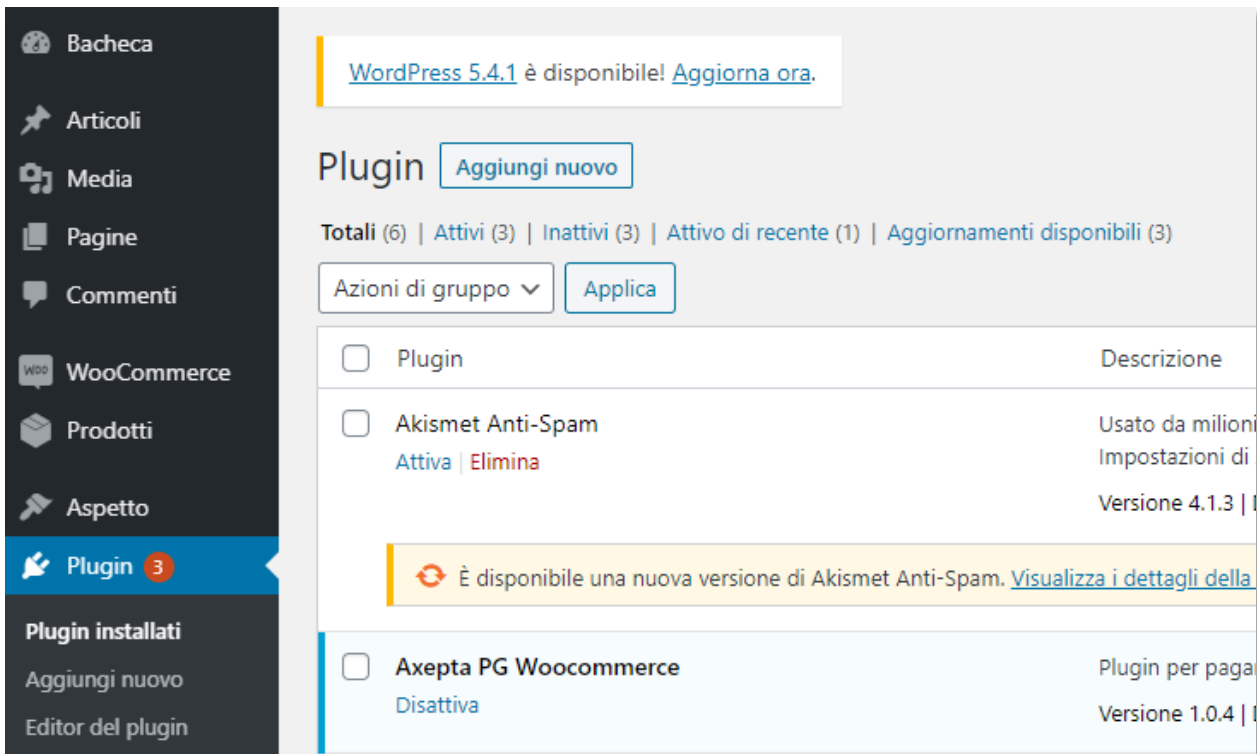
3.1 Installation

Installing the plugin requires a working instance of WordPress, with the plugin 'WooCommerce' already installed.

N.B. Access the WordPress backend via a user with full Admin privileges.

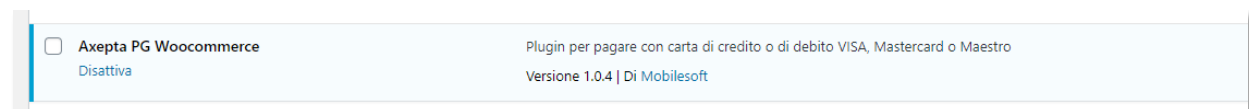
You can now upload the plugin to the platform using the native WordPress feature:

To upload the provided plugin in ZIP format, upload from the menu *Plugin > Add New*:



3.2 Activation

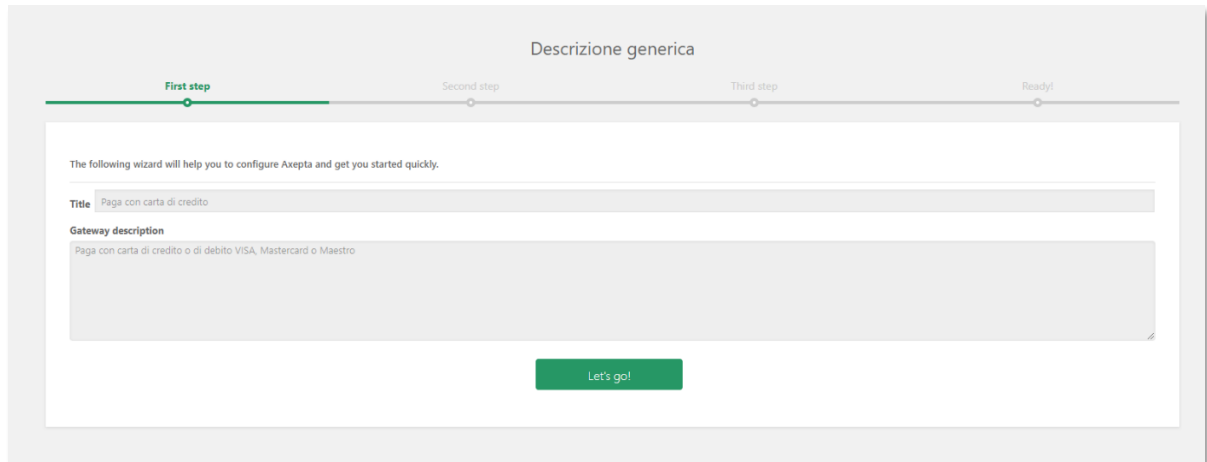
If the installation is successful, the plugin will be located in the menu *Plugin > Installed plugins*.



By clicking on the 'Activate' link, the plugin will be activated and the initial configuration wizard will open, consisting of the following steps:

Step #1: Introduction

The page offers an introductory presentation of the plugin:



Descrizione generica

First step Second step Third step Ready!

The following wizard will help you to configure Axapta and get you started quickly.

Title Pagamento con carta di credito

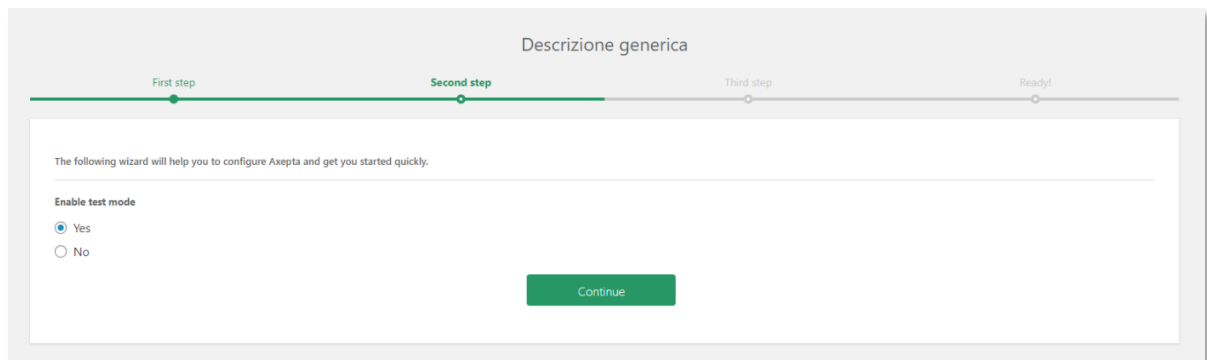
Gateway description
Pagamento con carta di credito o di debito VISA, Mastercard o Maestro

Let's go!

Step #2: environment

The page lets you select the environment to be used (Test / Production):

Note: To check that the plugin is working properly in your WordPress installation, it is recommended that you activate test mode with the first use.



Descrizione generica

First step Second step Third step Ready!

The following wizard will help you to configure Axapta and get you started quickly.

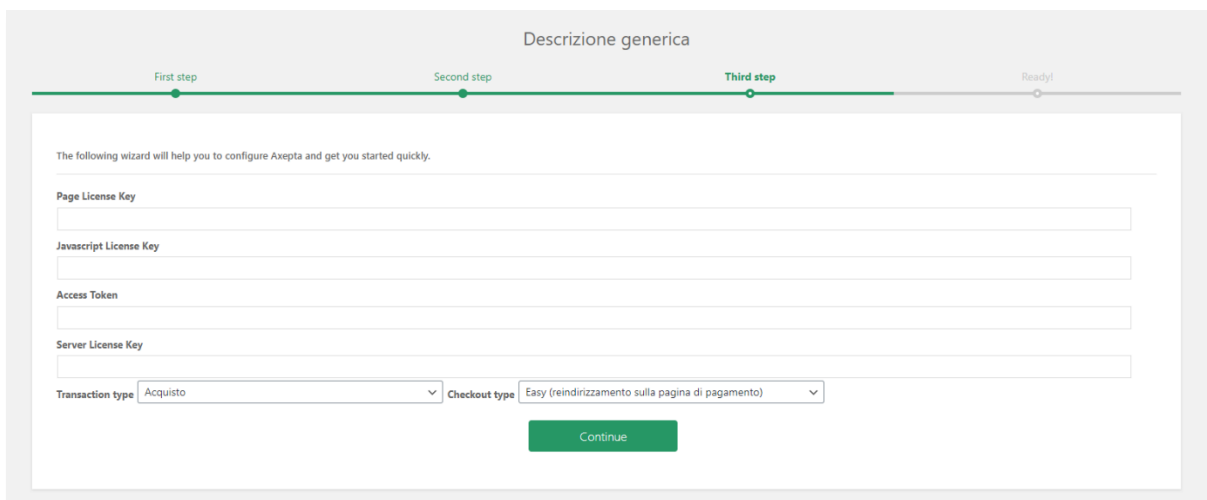
Enable test mode

☒ Yes
☐ No

Continue

Step #3: technical parameters

Contains the gateway configuration parameters (details of the fields are described in the next section):



Descrizione generica

First step Second step Third step Ready!

The following wizard will help you to configure Axapta and get you started quickly.

Page License Key

Javascript License Key

Access Token

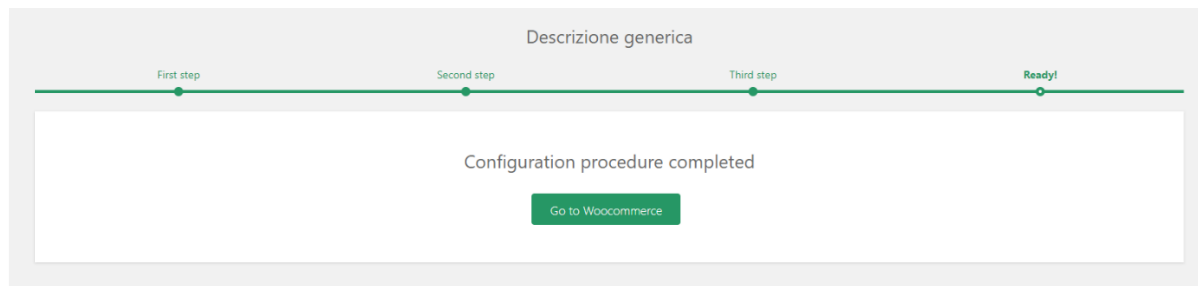
Server License Key

Transaction type Acquisto **Checkout type** Easy (reindirizzamento sulla pagina di pagamento)

Continue

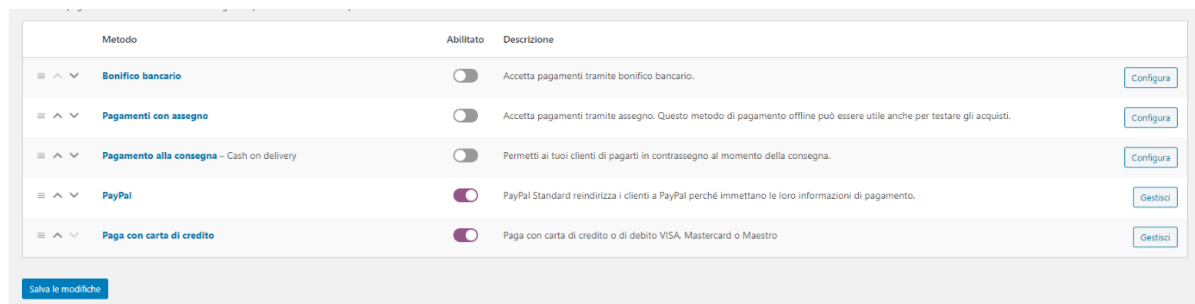
Step #4: end

Message of successful configuration following activation:



3.3 Configuration

The settings of the Axepta plugin are always available even after the first activation, from the WordPress menu *WooCommerce > Settings*, by clicking on the *Payments* tab: if the installation (described in the previous section) is successful, the payment method 'Pay by credit card' will be present:

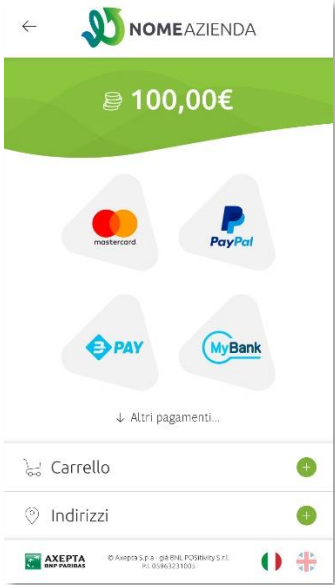
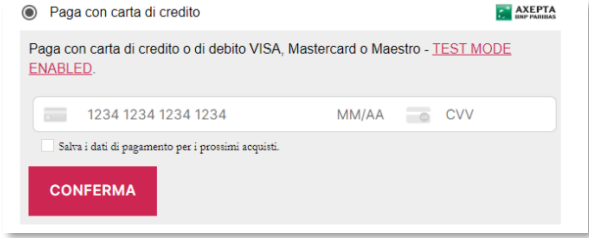


Clicking on the *Manage* button, you can access the plugin settings page, which allows you to configure the parameters:

Parameter	Value
Enable/disable	Enable or disable the payment method
Title	Name of the payment instrument
Description	Description of the payment instrument
Technical configurations	
License Key Easy	The parameter will be provided by Axepta when the service is activated.
License Key Smart	The parameter will be provided by Axepta when the service is activated.
Access Token	The parameter will be provided by Axepta when the service is activated.
Server License Key	The parameter will be provided by Axepta when the service is activated.



Test mode	If the setting is active, the selected endpoint will be the test endpoint, otherwise the production endpoint.
Transaction type	<ol style="list-style-type: none">1. Purchase At the end of the payment transaction, the user's card will be charged2. Preauthorisation At the end of the payment transaction, an authorisation will be made on the user's card: the amount will be blocked until it is moved or reversed by the merchant.3. Verification The transaction lets you verify the validity of the card, entering a transaction and reversing it immediately. Usually it is not applicable with plugin integration. For any further information, please contact Axepta support.

Checkout type	<p>Logical organisation of the checkout page, the following options are available:</p> <ol style="list-style-type: none"> Easy <p>On the checkout page, a button is shown that redirects to the payment page on Axepta SpA systems:</p>  Smart <p>Payment is made without leaving the operator's website:</p> 
Enable log	If this setting is enabled, the plugin will record the transactions made in log files.
Download LOG by date range	Allows to download previously saved log files.
Callback Url	URL for the offline notification of the transaction outcome (read-only).
Notify Url	URL of the page where the user will be redirected at the end of a successful transaction (read-only).
Error Url	URL of the page where the user will be redirected at the end of a failed transaction (read-only).
Additional Information Additional	



Info 1	<p>Free text field for adding information to be associated with the transaction.</p> <p>The following tags are available for adding dynamic parameters:</p> <ul style="list-style-type: none">• {site_title}• {order_number}• {order_date}• {customer_name}• {customer_email} <p>Example: An order placed by Mario Rossi on 14-05-20, the following string “<i>Order from customer {order_number} of {order_date}</i>” would be transformed into “<i>Order from customer Mario Rossi on 14-05-20</i>”</p>
Info 2	
Info 3	
Info 4	
Info 5	
Order status	
Order status - Pending payment	Label of order with ‘pending’ status
Order status - Processing	Label of order with ‘processing’ status
Order status - On hold	Label for ‘wait’ message
Order status - Completed	Label of order with ‘completed’ status
Order status Cancelled	Label or order with ‘cancelled’ status
Order status Refunded	Label of order with 'refunded' status
Order status Failed	Label of order with ‘failed’ status
Style formatting	
CSS style	CSS code inside the plugin to manage its graphic aspects.

3.4 From Test to Production

After tests are completed, you can switch to production mode. First, please write down testing parameters in some secure place, in order to prevent to lose them.

Then, you have the following two alternative ways:

1) WIZARD

- a. Run the installation WIZARD

Setup wizard

- b. enter "No" in STEP #2

- c. enter the **production parameters** (access token and license keys provided by Axepta) in STEP #3. These production parameters must be different of test parameters.

2) PLUGIN CONFIGURATION PAGE

- a. On the Plugin Configuration page (see previous paragraphs), change the *License Key Easy*, *License Key Smart*, *Access Token*, *License Key API* parameters and set the *Test mode* parameter to “No”. Note that you need only Smart or Easy license, not both of them.
- b. Save

Now, you are ready to test a real payment in production with a real card.

4 PrestaShop 1.6.x Plugin

Compatibility

PrestaShop 1.6.1 and higher
Other technology requirements are specified by the CMS itself, depending on the version.

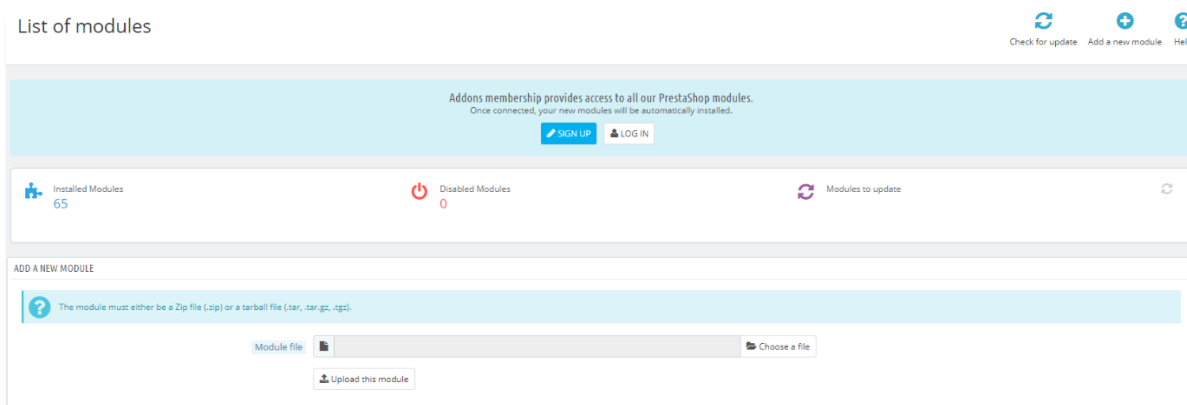
4.1 Installation

N.B. Access the PrestaShop backend via a user with full Admin privileges.

Installing the plugin requires a working instance of PrestaShop.

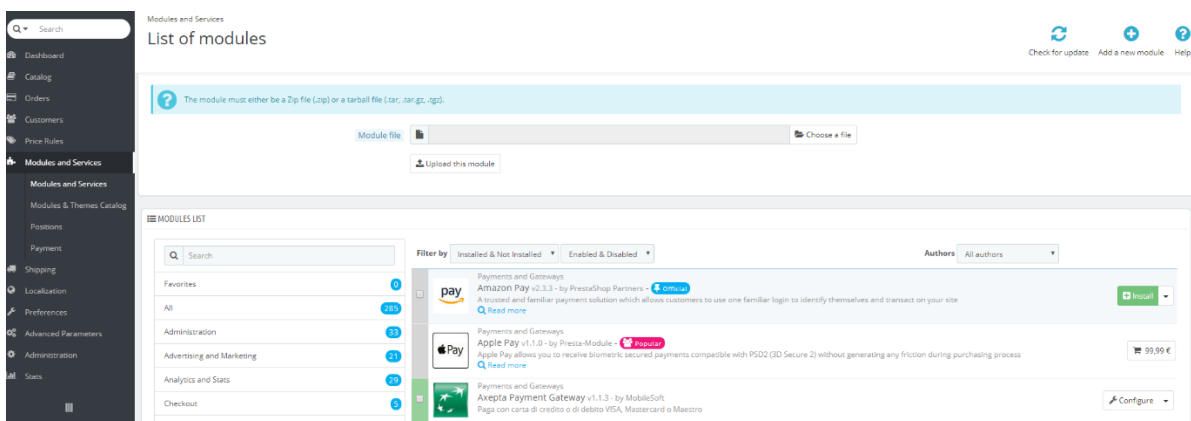
You can now upload the plugin to the platform using the native feature.

To upload the plugin provided in .zip or tarball format (.tar, .tar.gz or .tgz), upload from the *Modules and services* menu by clicking on the 'Add a new module' button:



4.2 Activation

If the installation is successful, the plugin will be located in the menu *Modules and Services* and you can enable it:



4.3 Configuration

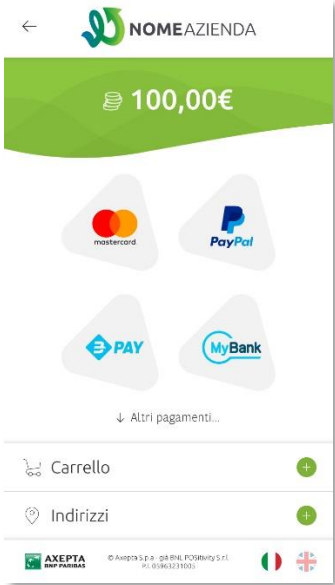
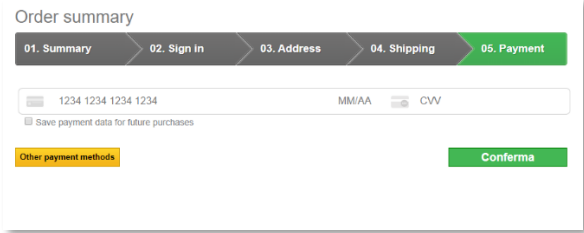
N.B. Access the PrestaShop backend via a user with full Admin privileges.

The plugin settings are always available, from the menu *Modules and services*, by clicking the special button of the *Axepta Payment Gateway* plugin:



Clicking on the button, you can access the plugin settings page, which allows you to configure the parameters:

Parameter	Value
Description	Description of the payment instrument
Instructions	Additional information to provide on the thank you page when the order is executed.
Technical configurations	
License Key Easy	The parameter will be provided by Axepta when the service is activated.
License Key Smart	The parameter will be provided by Axepta when the service is activated.
Access Token	The parameter will be provided by Axepta when the service is activated.
Server License Key	The parameter will be provided by Axepta when the service is activated.
Test mode	If the setting is active, the selected endpoint will be the test endpoint, otherwise the production endpoint.
Transaction type	<ol style="list-style-type: none"> Purchase At the end of the payment transaction, the user's card will be charged Preauthorisation At the end of the payment transaction, an authorisation will be made on the user's card: the amount will be blocked until it is moved or reversed by the merchant. Verification The transaction lets you verify the validity of the card, entering a transaction and reversing it immediately. Usually it is not applicable with plugin integration. For any further information, please contact Axepta support.

Checkout type	<p>Logical organisation of the checkout page, the following options are available:</p> <ol style="list-style-type: none"> Easy <p>On the checkout page, a button is shown that redirects to the payment page on Axepta SpA systems:</p>  Smart <p>Payment is made without leaving the operator's website:</p> 
Enable log	If this setting is enabled, the plugin will record the transactions made in log files.
Additional Information – Additional	
Info 1	<p>Free text field for adding information to be associated with the transaction.</p> <p>The following tags are available for adding dynamic parameters:</p> <ul style="list-style-type: none"> {site_title}
Info 2	
Info 3	
Info 4	

Info 5	<ul style="list-style-type: none"> • {order_number} • {order_date} • {customer_name} • {customer_email} • Example: An order placed by Mario Rossi on 14-05-20, the following string <i>"Order from customer {order_number} of {order_date}"</i> would be transformed into <i>"Order from customer Mario Rossi on 14-05-20"</i>
Gateway information	
Callback Url	URL for the offline notification of the transaction outcome (read-only).
Notify Url	URL of the page where the user will be redirected at the end of a successful transaction (read-only).
Error Url	URL of the page where the user will be redirected at the end of a failed transaction (read-only).
Expiration Time	Duration of the payment initialisation session.

4.4 From Test to Production

After tests are completed, you can switch to production mode. First, please write down testing parameters in some secure place, in order to prevent to lose them.

Then, you have to follow these steps:

- On the Plugin Configuration page (see previous paragraphs), change the License Key Easy, License Key Smart, Access Token, License Key API parameters and set the Test mode parameter to "No". Note that you need only Smart or Easy license, not both of them.
- Save

Now, you are ready to test a real payment in production with a real card.

5 PrestaShop 1.7.x Plugin

Compatibility	PrestaShop 1.7.6 and higher Other technology requirements are specified by the CMS itself, depending on the version.
----------------------	---

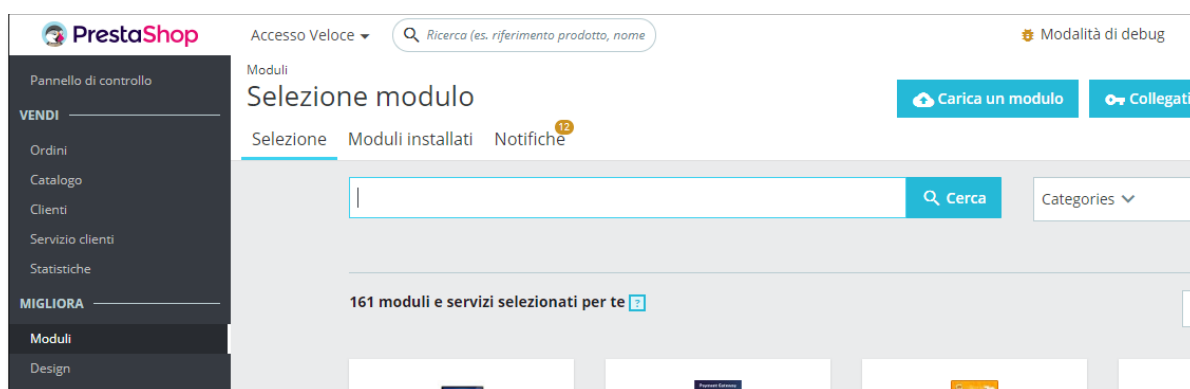
5.1 Installation

N.B. Access the PrestaShop backend via a user with full Admin privileges.

Installing the plugin requires a working instance of PrestaShop.

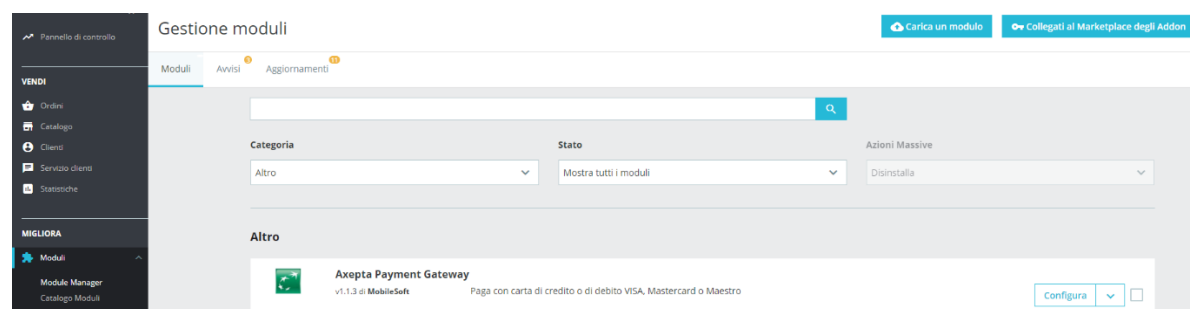
You can now upload the plugin to the platform using the native feature.

To upload the plugin provided in .zip or tarball format (.tar, .tar.gz or .tgz), upload from the *Upgrade > Modules* menu by clicking on the 'Upload a module' button:



5.2 Activation

If the installation is successful, the plugin will be located in the menu *Upgrade > Modules > Installed modules*:



By clicking on the 'Activate' link, the plugin will be activated and you will be able to access the configuration.

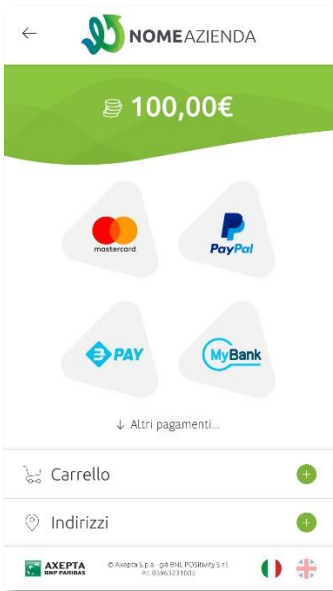
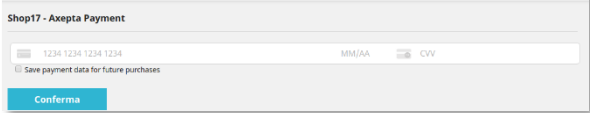
5.3 Configuration

The settings of the Axepta Payment Gateway plugin are always available, from the *Upgrade > Modules > Module Manager* menu, through the special configure button of the 'Axepta Payment Gateway' module:



Clicking on the button, you can access the plugin settings page, which allows you to configure the parameters:

Parameter	Value
Description	Description of the payment instrument
Instructions	Additional information to provide on the thank you page when the order is executed.
Technical configurations	
License Key Easy	The parameter will be provided by Axepta when the service is activated.
License Key Smart	The parameter will be provided by Axepta when the service is activated.
Access Token	The parameter will be provided by Axepta when the service is activated.
Server License Key	The parameter will be provided by Axepta when the service is activated.
Test mode	If the setting is active, the selected endpoint will be the test endpoint, otherwise the production endpoint.
Transaction type	<ol style="list-style-type: none"> 4. Purchase At the end of the payment transaction, the user's card will be charged 5. Preauthorisation At the end of the payment transaction, an authorisation will be made on the user's card: the amount will be blocked until it is moved or reversed by the merchant. 1. Verification The transaction lets you verify the validity of the card, entering a transaction and reversing it immediately. Usually it is not applicable with plugin integration. For any further information, please contact Axepta support.

Checkout type	<p>Logical organisation of the checkout page, the following options are available:</p> <ol style="list-style-type: none"> Easy <p>On the checkout page, a button is shown that redirects to the payment page on Axepta SpA systems:</p>  Smart <p>Payment is made without leaving the operator's website:</p> 
Enable log	If this setting is enabled, the plugin will record the transactions made in log files.
Additional Information - Additional	
Info 1	<p>Free text field for adding information to be associated with the transaction.</p> <p>The following tags are available for adding dynamic parameters:</p> <ul style="list-style-type: none"> {site_title}
Info 2	
Info 3	
Info 4	



Info 5	<ul style="list-style-type: none">• {order_number}• {order_date}• {customer_name}• {customer_email}• Example: An order placed by Mario Rossi on 14-05-20, the following string <i>"Order from customer {order_number} of {order_date}"</i> would be transformed into <i>"Order from customer Mario Rossi on 14-05-20"</i>
Gateway information	
Callback Url	URL for the offline notification of the transaction outcome (read-only).
Notify Url	URL of the page where the user will be redirected at the end of a successful transaction (read-only).
Error Url	URL of the page where the user will be redirected at the end of a failed transaction (read-only).
Expiration Time	Duration of the payment initialisation session.

5.4 From Test to Production

After tests are completed, you can switch to production mode. First, please write down testing parameters in some secure place, in order to prevent to lose them.

Then, you have to follow these steps:

- c. On the Plugin Configuration page (see previous paragraphs), change the License Key Easy, License Key Smart, Access Token, License Key API parameters and set the Test mode parameter to "No". Note that you need only Smart or Easy license, not both of them.
- d. Save

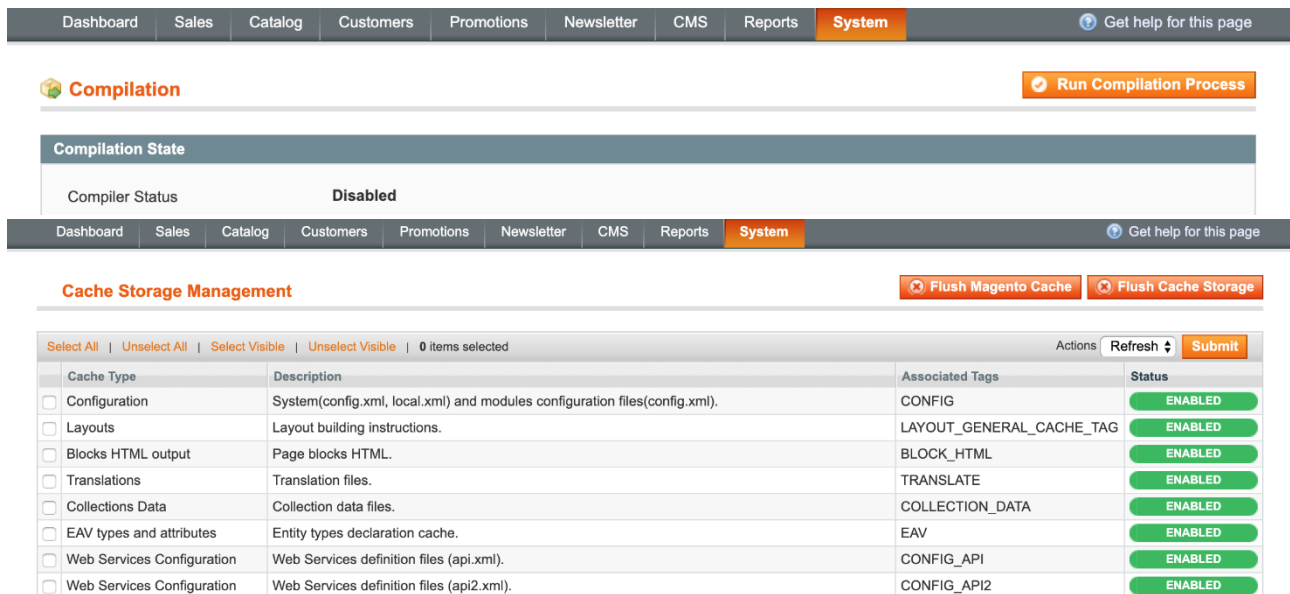
Now, you are ready to test a real payment in production with a real card.

6 Magento 1.9.x Plugin

Compatibility	Magento 1.9.3 or higher Other technology requirements are specified by the CMS itself, depending on the version.
----------------------	---

6.1 Installation

Access the Magento backend. From the top menu, select System / Cache Management and enable all cache types;



The screenshot shows the Magento 1.9.x backend interface. At the top, the 'System' menu item is selected. Below the navigation bar, the 'Compilation' section is visible, showing the 'Compiler Status' as 'Disabled'. A 'Run Compilation Process' button is present. Below this, the 'Cache Storage Management' section is shown, featuring a table of cache types and their statuses. All cache types are currently 'ENABLED'.

Cache Type	Description	Associated Tags	Status
<input type="checkbox"/> Configuration	System(config.xml, local.xml) and modules configuration files(config.xml).	CONFIG	ENABLED
<input type="checkbox"/> Layouts	Layout building instructions.	LAYOUT_GENERAL_CACHE_TAG	ENABLED
<input type="checkbox"/> Blocks HTML output	Page blocks HTML.	BLOCK_HTML	ENABLED
<input type="checkbox"/> Translations	Translation files.	TRANSLATE	ENABLED
<input type="checkbox"/> Collections Data	Collection data files.	COLLECTION_DATA	ENABLED
<input type="checkbox"/> EAV types and attributes	Entity types declaration cache.	EAV	ENABLED
<input type="checkbox"/> Web Services Configuration	Web Services definition files (api.xml).	CONFIG_API	ENABLED
<input type="checkbox"/> Web Services Configuration	Web Services definition files (api2.xml).	CONFIG_API2	ENABLED

Also in the top menu, select System / Tools / Compilation and make sure that the compilation is disabled. It should display "Compiler Status: Disabled";

In a temporary folder, extract the archive **axepta_pg_magento1-x.x.x.tgz**

Connect to your server's Magento root folder via FTP/SFTP/SSH and upload all the extracted folders to the temporary directory of step 3 (app/ js/ lib/ skin/ folders). Be sure to use merge mode. Do not replace the entire folder. This will allow the FTP/SFTP client to add only the new files. This mode is usually used by default by most FTP/SFTP software.

From the Magento backend, in the top menu, select System / Cache Management and press the Flush Magento Cache button.



The payment gateway is now installed. If you need to re-enable the compilation, return via the top menu to System / Tools / Compilation and click Run Compilation Process.



6.2 Configuration

N.B. Access the Magenta backend via a user with full Admin privileges.



The plugin settings are always available from the menu *System > Configuration > Sales > Payment methods*:

Current Configuration Scope:
Default Config
Manage Stores

Configuration

GENERAL

General

Web

Design

Currency Setup

Store Email Addresses

Contacts

Reports

Content Management

CATALOG

Catalog

Configurable Swatches

Inventory

Google Sitemap

RSS Feeds

Email to a Friend

CUSTOMERS

Newsletter

Customer Configuration

Wishlist

Promotions

Persistent Shopping Cart

SALES

Sales

Sales Emails

PDF Print-outs

Tax

Checkout

Shipping Settings

Shipping Methods

Google API

Payment Methods

Payment Services

Moneybookers

SERVICES

Magento Core API

Payment Methods

Axepta

Request technical assistance

Setup wizard

Title

Paga con carta di credito

[STORE VIEW]

Gateway description

Paga con carta di credito o di debito VISA, Mastercard o Maestro

[STORE VIEW]

Select Payment Gateway

Paga con carta di credito

[STORE VIEW]

Configuration settings

Access Token

[STORE VIEW]

Server License Key

[STORE VIEW]

License Key Easy

[STORE VIEW]

License Key Smart

[STORE VIEW]

Checkout Type

Easy Checkout (reindirizzamento sulla pagina di)

[STORE VIEW]

Transaction type

Acquisto

[STORE VIEW]

Test mode

☒ Yes ☐ No

[STORE VIEW]

Enable log?

☒ Yes ☐ No

To work properly you need to enable system logs

[STORE VIEW]

Download log by date range

Download

[STORE VIEW]

Enable payment method?

☒ Yes ☐ No

[STORE VIEW]

Callback url

http://127.0.0.1/index.php/axepta_paymentservice/init/success/

[STORE VIEW]

Notify url

http://127.0.0.1/index.php/axepta_paymentservice/init/notify/

[STORE VIEW]

Error url

http://127.0.0.1/index.php/axepta_paymentservice/init/error/

[STORE VIEW]

Information settings

Add Info 1

[[site_title]] Ordine da nuovo cliente ([order_number]

[STORE VIEW]

Add Info 2

[[site_title]] Ordine da nuovo cliente ([order_number]

[STORE VIEW]

Add Info 1

[[site_title]] Ordine da nuovo cliente ([order_number]

[STORE VIEW]

Add Info 4

[[site_title]] Ordine da nuovo cliente ([order_number]

[STORE VIEW]



Clicking on the button, you can access the plugin settings page, which allows you to configure the parameters:

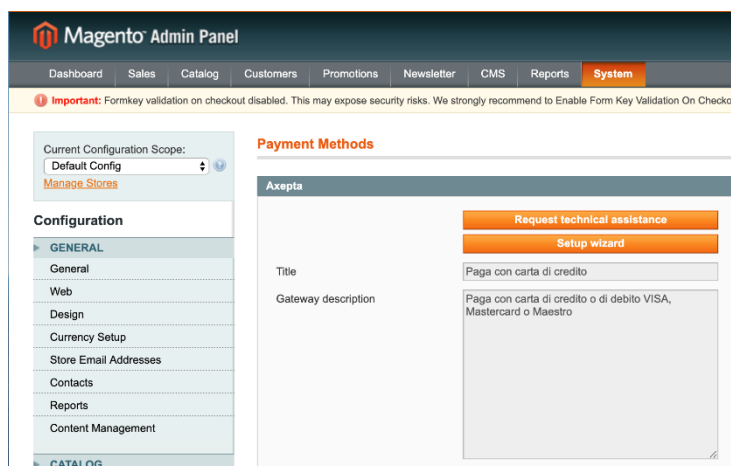
Parameter	Value
Title	Payment instrument title (read-only)
Instrument description	Description of the payment instrument (read-only)
Select payment system	Only one preset option: "Pay with credit cards"
Technical configurations	
ACCESS TOKEN	Payment instrument identification token, communicated to the operator when the service is activated.
SERVER LICENSE KEY	License for SERVER TO SERVER communications, communicated to the operator when the service is activated.
LICENSE KEY EASY	License to use EASY mode, communicated to the operator when the service is activated.
LICENSE KEY SMART	License to use SMART mode, communicated to the operator when the service is activated.
Checkout type	<p>Logical organisation of the checkout page, the following options are available:</p> <ol style="list-style-type: none">1. Easy Checkout (redirection to the payment page) On the checkout page, a button is shown that redirects to the payment page on Axepta SpA systems:2. Smart Checkout (on-site payment) On the checkout page, the credit card payment form via SDK JS is directly displayed.
Transaction type	<ol style="list-style-type: none">1. Purchase At the end of the payment transaction, the user's card will be charged2. Preauthorisation At the end of the payment transaction, an authorisation will be made on the user's card: the amount will be blocked until it is moved or reversed by the merchant.3. Verification At the end of the payment process, a verification will be carried out on the user's card: the transaction lets you verify the validity of the card, entering a transaction with a figurative amount of €1 and immediately reversing it. Usually it is not applicable



	with plugin integration. For any further information, please contact Axepta support.
Test-mode active	If the setting is active, the selected endpoint will be the test endpoint, otherwise the production endpoint. The other parameters must be aligned with the selected environment.
Enable logs	If this setting is enabled, the plugin will record the transactions made in log files.
Download the LOG for the date range	Downloads previously saved log files by selecting the date range.
Enable payment instrument	If enabled, the plugin will be active on the frontend, otherwise it will not be usable.
Callback Url	URL for the offline notification of the transaction outcome (read-only).
Notify Url	URL of the page where the user will be redirected at the end of a successful transaction (read-only).
Error Url	URL of the page where the user will be redirected at the end of a failed transaction (read-only).
Additional information	
Additional information 1	Free text field for adding information to be associated with the transaction. The following tags are available for adding dynamic parameters: <ul style="list-style-type: none">• {site_title}• {order_number}• {order_date}• {customer_name}• {customer_email}
Additional information 2	
Additional information 3	
Additional information 4	
Additional information 5	
Style formatting	
CSS style	CSS code to be applied to the payment page.

6.3 Assistance request

To send an assistance request directly from the plugin, go to the Axepta section of the menu *System > Configuration > Sales > Payment methods*; the box displayed has a Request Technical assistance button:



By clicking on it, the contact form appears:

Request technical assistance

Questo form apre un ticket di assistenza

Cc

Message *

Send Request

6.4 From Test to Production

After tests are completed, you can switch to production mode. First, please write down testing parameters in some secure place, in order to prevent to lose them.

Then, you have to follow these steps:

- e. On the Plugin Configuration page (see previous paragraphs), change the License Key Easy, License Key Smart, Access Token, License Key API parameters and set the Test mode parameter to "No". Note that you need only Smart or Easy license, not both of them.
- f. Save

Now, you are ready to test a real payment in production with a real card.

7 Magento 2.x Plugin

Compatibility	Magento 2.* Other technology requirements are specified by the CMS itself, depending on the version.
----------------------	---

7.1 Installation


N.B. Access the Magento backend via a user with full Admin privileges.

A working instance of Magento version 2.x is required to install the plugin.

Manual installation:

To install the plugin you must follow the following steps:

- Unzip the file “**axepta-pg-magento2-x.x.x.zip**” and extract the folder “**Axepta**” in the folder <rootMagento>/app/code/ (if the folder “code” is not present, you must create it and assign it read/write permissions)
- Run the commands in the following order:
 - `bin/magento setup:upgrade`
 - `bin/magento setup:di:compile`
 - `bin/magento cache:flush`
- Access the administrator panel from a browser, with the user and password of the admin user
- Access stores > configuration > sales > payment methods > other payment methods > Axepta; click on the “Axepta” tab
- Fill in the required fields with the data provided by Axepta:



DASHBOARD
SALES
CATALOG
CUSTOMERS
MARKETING
CONTENT
REPORTS
STORES
SYSTEM
FIND PARTNERS & EXTENSIONS

Configuration

Save Config

Multishipping Settings
Shipping Methods
Google API
Payment Methods
Fraud Protection
ENGAGEMENT CLOUD
SERVICES
ADVANCED

AXEPTA

Request technical assistance

Title

Paga con carte di credito

Description

Paga con carta di credito o di debito VISA, Mastercard o Maestro

Payment Gateway

Paga con carta di credito

Configurazioni tecniche

Acess Token

Server License Key

License Key Easy

License Key Smart

Checkout Type

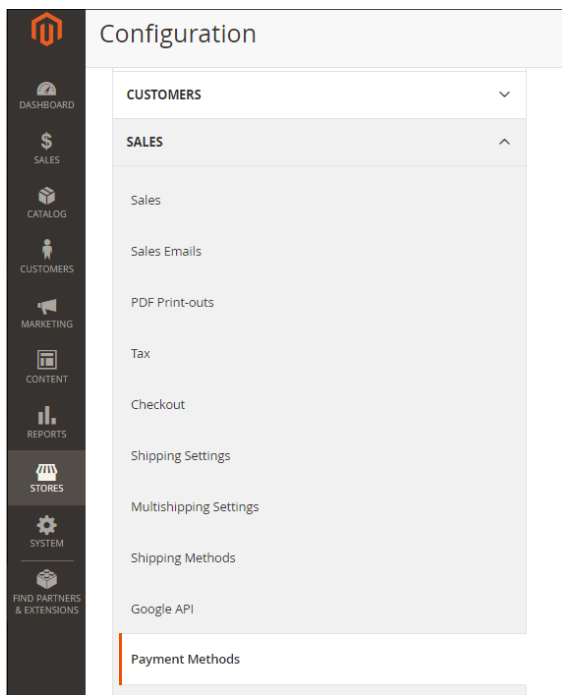
Easy Checkout (redirect on payment page)

7.2 Activation

If the installation is successful, the plugin is located in the menu *Stores > Configuration > Sales > Payment Methods*.

7.3 Configuration

The Axepta plugin settings are always available, from the menu *Stores > Configuration > Sales > Payment Methods*:



The screenshot shows the 'Configuration' page with a sidebar menu. The 'STORES' section is expanded, showing a list of options: Sales, Sales Emails, PDF Print-outs, Tax, Checkout, Shipping Settings, Multishipping Settings, Shipping Methods, Google API, and Payment Methods. The 'Payment Methods' option is highlighted with an orange bar.

Clicking on the button, you can access the plugin settings page, which allows you to configure the parameters:

27



Parameter	Value
Title	Payment instrument title (read-only)
Instrument description	Description of the payment instrument (read-only)
Select payment system	Only one preset option: "Pay with credit cards"
Technical configurations	
ACCESS TOKEN	Payment instrument identification token, communicated to the operator when the service is activated.
SERVER LICENSE KEY	License for SERVER TO SERVER communications, communicated to the operator when the service is activated.
LICENSE KEY EASY	License to use EASY mode, communicated to the operator when the service is activated.
LICENSE KEY SMART	License to use SMART mode, communicated to the operator when the service is activated.
Checkout type	<p>Logical organisation of the checkout page, the following options are available:</p> <ol style="list-style-type: none">1. Easy Checkout (redirection to the payment page) On the checkout page, a button is shown that redirects to the payment page on Axepta SpA systems:2. Smart Checkout (on-site payment) On the checkout page, the credit card payment form via SDK JS is directly displayed.
Transaction type	<ol style="list-style-type: none">1. Purchase At the end of the payment transaction, the user's card will be charged2. Preauthorisation At the end of the payment transaction, an authorisation will be made on the user's card: the amount will be blocked until it is moved or reversed by the merchant.3. Verification At the end of the payment process, a verification will be carried out on the user's card: the transaction lets you verify the validity of the card, entering a transaction with a figurative amount of €1 and immediately reversing it. Usually it is not applicable with plugin integration. For any further information, please contact Axepta support.



Test-mode active	If the setting is active, the selected endpoint will be the test endpoint, otherwise the production endpoint. The other parameters must be aligned with the selected environment.
Enable logs	If this setting is enabled, the plugin will record the transactions made in log files.
Download the LOG for the date range	Downloads previously saved log files by selecting the date range.
Enable payment instrument	If enabled, the plugin will be active on the frontend, otherwise it will not be usable.
Callback Url	URL for the offline notification of the transaction outcome (read-only).
Notify Url	URL of the page where the user will be redirected at the end of a successful transaction (read-only).
Error Url	URL of the page where the user will be redirected at the end of a failed transaction (read-only).
Additional information	
Additional information 1	Free text field for adding information to be associated with the transaction. The following tags are available for adding dynamic parameters: <ul style="list-style-type: none">• {site_title}• {order_number}• {order_date}• {customer_name}• {customer_email}
Additional information 2	
Additional information 3	
Additional information 4	
Additional information 5	
Style formatting	
CSS style	CSS code to be applied to the payment page.

7.4 Upgrading

To upgrade the plugin, follow the following steps:

Manual updating:

1. Download the .zip file from the link indicated in the email
2. Unzip the downloaded file to a temporary folder
3. Access the <root> of Magento
4. Run the commands in the following order:
 - `php bin/magento module:disable Axepta_PaymentService`

- `php bin/magento setup:di:compile`
- `php bin/magento cache:clean`

5. Go to the folder `app/code`:

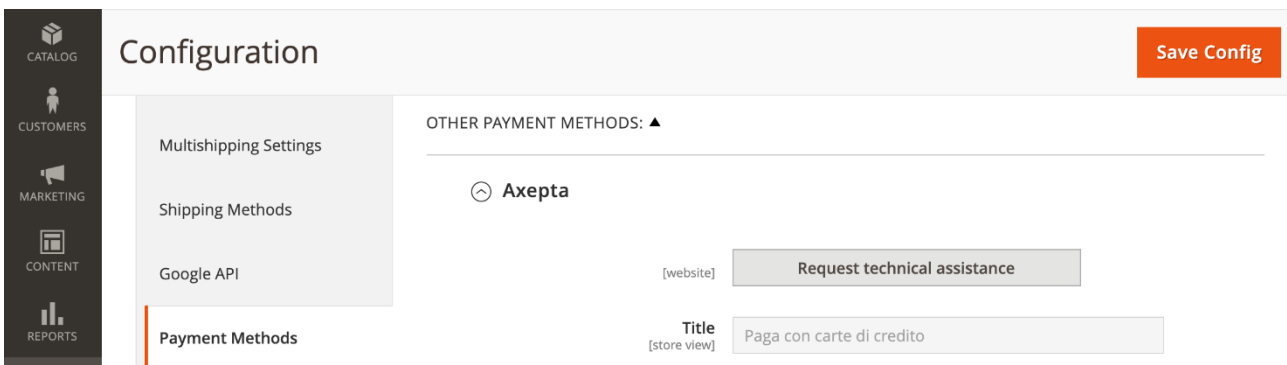
- Remove the `Axepta` folder and all its contents
- Copy `Axepta` and all its contents from the temporary folder (the one where you saved the previously unpacked files)

6. Run the commands in the following order:

- `composer remove Axepta/PaymentService --ignore-platform-reqs`
- `php bin/magento module:status`
- `php bin/magento module:enable Axepta_PaymentService`
- `php bin/magento setup:upgrade`
- `php bin/magento setup:di:compile`
- `php bin/magento cache:clean`

7.5 Assistance request

To send a support request directly from the plugin, go to the Axepta section of the menu *Stores > Configuration > Sales > Payment Methods*; the box displayed has a Request technical assistance button:



By clicking on it, the contact form appears:

Request technical assistance

Questo form apre un ticket di assistenza

Cc

Messaggio *

Invia

7.6 From Test to Production

After tests are completed, you can switch to production mode. First, please write down testing parameters in some secure place, in order to prevent to lose them.

Then, you have to follow these steps:

- g. On the Plugin Configuration page (see previous paragraphs), change the License Key Easy, License Key Smart, Access Token, License Key API parameters and set the Test mode parameter to “No”. Note that you need only Smart or Easy license, not both of them.
- h. Save

Now, you are ready to test a real payment in production with a real card.

8 Credit cards for demonstration use

Below are the cards to be used for tests (test mode enabled). N.B. Where specified, the correct expiry date and/or CVV code must be indicated otherwise the transaction will fail.

PAN	Scheme	Expiry	CVV	Enrstatus	Authstatus	Authorisation outcome
455777333333335	Visa	-	-	Y	Y	OK
455777222222229	Visa	-	-	Y	A	OK
411111111111111	Visa	10/2023	-	N	-	OK
455500000000001	Visa	-	-	N	-	OK
411111112225555	Visa	-	-	N	-	OK
401151444444116	VisaElectron	-	-	N	-	OK
401151999222222	VisaElectron	-	-	N	-	OK
400500000000007	VisaDebit	10/2023	-	N	-	OK
400500445555556	VisaDebit	-	-	N	-	OK
543013222222226	Mastercard	-	-	N	-	OK
589353554444429	Mastercard	-	-	N	-	OK
5790640100000005	Mastercard	-	-	N	-	OK
5430131234567891	Mastercard	-	-	Y	N	KO
5548535889622125	Mastercard	-	-	N	-	OK
540117222222227	Mastercard	-	-	Y	Y	OK
543013222222226	Mastercard	-	-	N	-	OK
5548536000000126	Mastercard	-	-	Y	Y	OK
526492111111115	MastercardDebit	-	555	N	-	OK
5545910000000019	MastercardDebit	-	-	Y	N	KO
5020639451965933	Maestro	-	-	Y	Y	OK
5893535596092423	Maestro	-	-	Y	A	KO

Where:

- *enrStatus* represents the card enrollment status in the 3D Secure service
 - Y – Authentication available;
 - N – Cardholder not participating in the service;



- U – Authentication not possible;
- E – Error.
- *AuthStatus* represents the outcome of card authentication in the 3D Secure service
 - Y – Authenticated;
 - A - Attempted authentication;
 - N – Cardholder not authenticated;
 - U – Authentication not possible.

The testing environment includes some test cases through which the behaviour of the solution can be simulated when some conditions that would invalidate the transaction occur. They respond to the following Pan / Amount combinations:

PAN	Amount	Expiry	Outcome	Description
455777333333335 540117222222227 455777222222229 5548536000000126 5020639451965933	101,00	any	PG_01045	Authorization denied
	102,00	any	PG_01058	Incorrect merchant code
	103,00	any	PG_01057	Invalid card
	104,00	any	PG_01086	Holder not enabled for this operation
	105,00	any	PG_01089	Frequency limit exceeded
	106,00	any	PG_01180	Stolen card
	107,00	any	PG_01038	Format error
	108,00	any	PG_01080	Contact issuer
	109,00	any	PG_01078	Suspected fraud
543013222222226 411111111111111 401151999222222 400500000000007 526492111111115	103,00	any	PG_01060	Insufficient funds
	104,00	any	PG_01086	Holder not enabled for this operation
	105,00	any	PG_01089	Frequency limit exceeded
540117222222227 526492111111115	> 200	any	PG_01060	Insufficient funds
411111111111111 400500000000007	Any different from above	Any different from 10/23	PG_01018	Card expired



8.1 Testing 3DS 2.x

If merchant is enabled to 3DS 2.x, then it needed to make some specific tests. In particular you can follow below table in order to test some 3DS 2.x scenarios.

PAN	Scheme	Expiry	cvv	transStatus	Authorization outcome
4012001036275556	VISA	10/23	any	U	UNABLE
5255999999999992	MC	10/23	any	U	UNABLE
4012001038443335	VISA	10/23	any	Y	RISK BASED AUTHENTICATION (FRICTIONLESS)
5453010000073866	MC	10/23	any	Y	RISK BASED AUTHENTICATION (FRICTIONLESS)
5256103270096532	MC	10/23	any	Y	RISK BASED AUTHENTICATION (FRICTIONLESS)
4824983270096509	VISA	10/23	any	Y	RISK BASED AUTHENTICATION (FRICTIONLESS)
4012001036853337	VISA	10/23	any	N	REFUSED BY THE PAYMENT GATEWAY
5453010000073684	MC	10/23	any	N	REFUSED BY THE PAYMENT GATEWAY
4012001037141112	VISA	10/23	any	C	FULL 3DSecure (CHALLENGE)
5453010000073205	MC	10/23	any	C	FULL 3DSecure (CHALLENGE)

9 Credit, Void and Confirm

Using the Axepta portal called **Byond**, merchant can make some operations on approved transaction, in particular:

- For a successful **PURCHASE** transaction, you can return an amount in order to return part or the entire amount paid into your wallet. This is called **Credit**.
- For a successful **AUTH** transaction, you can return an amount in order to return part or the entire amount paid into your wallet. This is called **Void**.
- For an **AUTH** transaction, you can confirm a specific amount in order to make the payment for a part or the entire amount. This is called **Confirm**.

This is very easy to do, by the means of this user friendly portal.

Alternatively, merchant can do these operations using Axepta's REST API. In this case, merchant has more flexibility but it needs to develop code. In order to do this, please contact Axepta support and ask for documentation.



10 Merchant side PCI data security information

The PCI DSS standard defines the compliance requirements that merchants must meet. The table below lists these requirements (Self Assessment Questionnaire SAQ and Report on Compliance RoC) for each type of technical integration solution that the merchant decides to implement.

Product	Technical solution	Description	Compliance (≤6,000,000 transactions/year)	Compliance (>6,000,000 transactions/year)
Easy Checkout	SDK Redirect	The user is redirected to the AXEPTA payment page and the card data can be entered on the same page	SAQ A	RoC ^A
Smart Checkout	SDK JS	The card data entry form is presented by the merchant and the data is transmitted directly to the payment gateway without interaction with the merchant server	SAQ A-EP	RoC ^{A-EP}

Where in principle the criteria are as follows:

- SAQ A / RoC^A = the entire payment page is managed by Axepta
- SAQ A-EP / RoC^{A-EP} = the merchant site does not store, process or transmit card data, but controls how the data is collected