payplaza
part of CM.com

App-2-app
Integration Guidelines.

For Sunmi P2 devices.

Document information

Document Title	App-2-app Integration Guidelines for Sunmi P2 devices
Classification	Public
Status	Final
Version	2.0.0

Version	Changelog
2.0.0	Initial version

Table of Contents

1. Terminology

Term	Description
App-2-app integration	Integration between two applications, namely Partner and PayPlaza Terminal application, which are on the same Sunmi device
Partner application	Android based application developed by the partner integrating with PayPlaza payment solution for payment acceptance
PayPlaza Terminal application	Android based application which calls the PayPlaza Gateway to process the payment
Sunmi	Payment terminal device manufacturer
Development Sunmi Payment device	Sunmi device that is used for development purposes and testing the application integration
PayPlaza Android POS integration Library	Android library that can be used in Partner applications to integrate with PayPlaza apps and perform payment transactions.

2. Introduction

This document explains the steps required to integrate the Partner application with the PayPlaza's App-2-app Integration solution, using the PayPlaza android POS integration library on an Android based Sunmi payment device.

In order to be able to use this library, the integrator has the choice the following flavours:

- Developing a pure native Android application that will include the library as dependency to start transactions and other operations on PayPlaza gateway.

- Developing a hybrid Android application that uses Web Views, of which the native part of the application will include the PayPlaza Android POS integration library to start transactions and other operations on PayPlaza gateway.

> ℹ️ At this point in time, PayPlaza does not support the library to be used on a browser application.

3. Download the PayPlaza apps

The following steps describe how to download the PayPlaza Terminal application for the Development Sunmi Payment device. Once you have performed the Development activities as described in chapter 5, you can verify that the payment flow performs as expected with the Partner application.

Step	Reference visual
Click 'App Store' on your Sunmi device	
Choose and install the Terminal app	
You are done!	

4. Payment solution overview

This chapter provides an overview of the required applications and how they work together as a payment solution. The transaction flow is described step by step.

Partner application, payment library and Terminal application

The App-2-app integration for the Sunmi Android platform consists of two applications and one library:

- **The Partner application**: Android based application developed by the partner integrating with PayPlaza payment solution for payment acceptance

- **The PayPlaza Terminal application**: handling card entry and the payment flow. It can perform payments and refund transactions. Also handles receipts for transactions.

- **The PayPlaza Android POS integration library**: provides functionality to handle transactions on the gateway. It requests transaction data to Partner application and forwards it to the terminal application. It also provides functionality to get information about old transactions and other info that may be useful for integrators, such as current currency code used by terminal app.

Partners need to have their Android app running on the same physical device as the PayPlaza Terminal application and include the PayPlaza Android POS integration library as a dependency on their app.

Transaction Flow

In this section we explain how the Partner application initiates a transaction and we describe both the transaction flow through the terminal application and the interactions between it and the user.
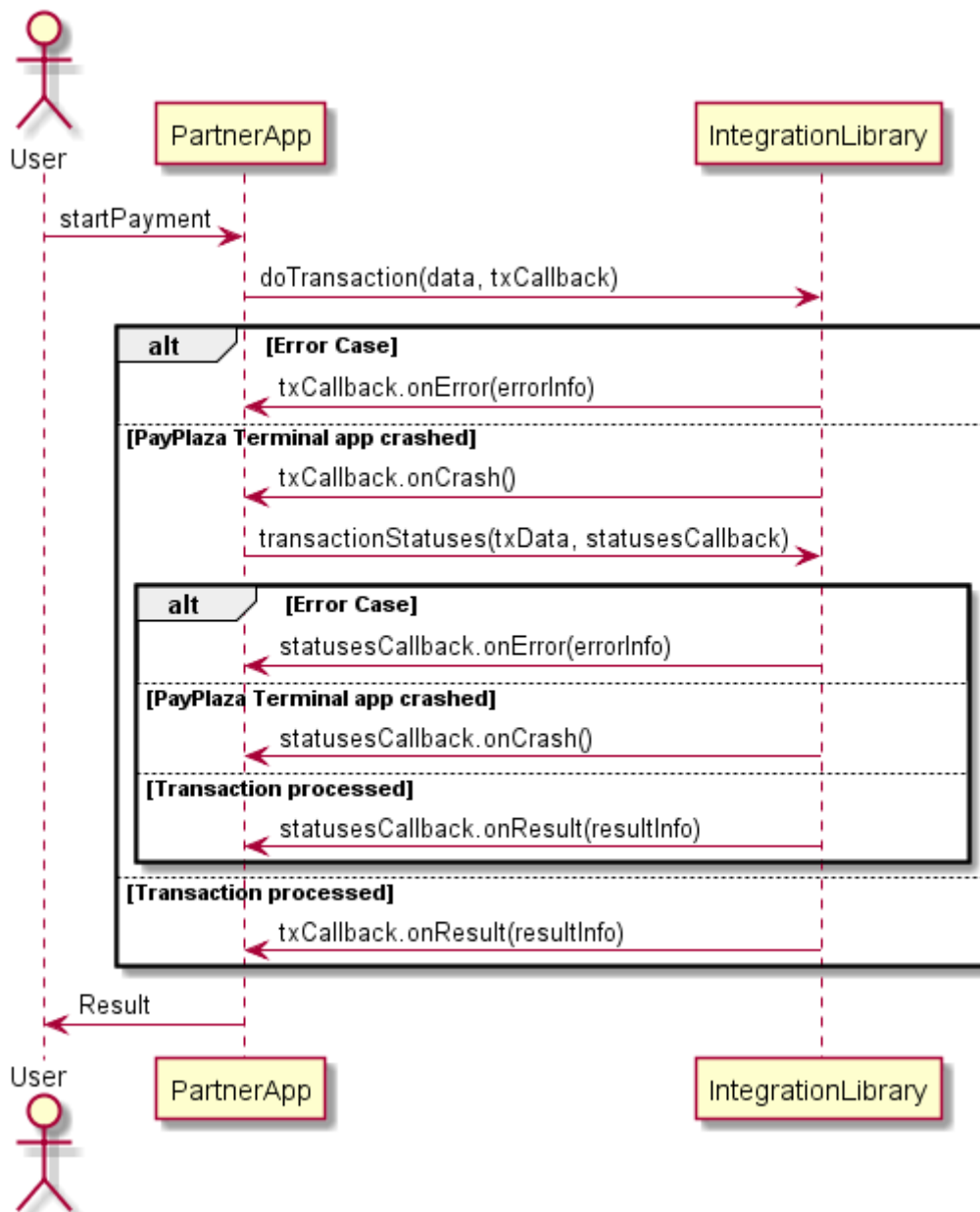
The basic functioning of the Integration library is that Partner Applications can send the transaction request (`doTransaction()` method on the Integration service class inside the Integration library) together with the call-back object. Once the result of the transaction is received, the library will trigger the appropriate call-back method.

A transaction starts with the merchant entering an amount. The device is then handed over to the cardholder, who presents their card, authorizes the transaction, and receives the transaction outcome. The device is then handed back to the merchant and, if needed, receipts are printed.

Transition between applications is seamless and the entire flow appears as if it is handled by the ECR application alone. This is done thanks to the integration library which integrates the call to the terminal application within partner's application context.

Also, it is important to mention that transaction flow in Terminal application does not manifest itself to the user in any way. When Terminal app is opened from the launcher screen it starts some functionality checks, such as terminal configuration availability, correct keys on device, network connection and gateway availability, and after those checks are finished, a banner advising the user to switch to an ECR application (In this case, the partner application) is displayed.

It is important to point out that while the Terminal application UI is visible during a transaction, it is not possible to interact with the Partner application.

You can also see on the diagram that sdk takes into account the possibitlity that the terminal app or device can crash by any reason. In that case, Partner App should request the status of the transaction to be certain about the result of the operation. (see "Dealing with unexpected scenarios" chapter)

I18n and language negotiation

The default language in which the user interface is presented by the PayPlaza application is the language configured in the shop configuration in the Gateway. It is expected that the partner application does the same.

The PayPlaza application will switch to the cardholder preferred language if suggested by the EMV card and if the translation into this language exists. The PayPlaza application will switch language, if applicable, after the language setting on the card has been read.

It is the responsibility of the partner application developers to make their application support the languages they want to use.

If the partner wants to override the language set for the shop, they can provide a language code to the PayPlaza applications. The PayPlaza applications will use that language, provided it is supported by our applications. If a language code is provided that is not supported by PayPlaza, our applications will default to the English language.

5. Development Guidelines

This section explains how your app communicates with the PayPlaza applications.

A card payment goes through the following flow:

1. The Partner application uses the PayPlaza Android POS integration library to start a transaction with the appropriate data (transaction type, amount, currency...).

2. The PayPlaza Terminal application receives the information thanks to the integration library and calls the PayPlaza Gateway to process the payment.

3. PayPlaza Android POS integration library receives the transaction result from Terminal app and send it back to the Partner applications using a call-back. The result consists of a result data and the receipts for the transaction (if any) from the Gateway.

4. The Partner application performs the transaction result and provides printer receipts.

Integration Mechanism

In order to pass data to the PayPlaza app, the Partner app needs to use the PayPlaza Android POS integration library methods and wait for a result. To be able to use these methods, Partner application need to include the library as a dependency in the code. Integration library can be used to integrate in two different environments: development and production. You should include the correct dependency for the correct environment in which you are operating. The usage of the debug version allows registered developers to use our development environment which offers simulators for various test scenarios. A complete working example app is available at Github, to give you a head start in integrating with the SDK.

Import Integration SDK as a dependency

Code Sample. add jitpack repository to your main build or settings gradle file

```
dependencyResolutionManagement {
    repositoriesMode.set(RepositoriesMode.FAIL_ON_PROJECT_REPOS)
    repositories {
        google()
        mavenCentral()
        maven { url 'https://jitpack.io' }
    }
}
```

Code Sample. dependency for production environment

```
dependencies {
    [...]
    // CM|PayPlaza Android POS Integration library
    implementation 'com.github.cmdotcom.android-pos-integration-sdk-
kotlin:androidposintegrationsdk:<version-tag>'
}
```

Code Sample. dependency for development environment

```
dependencies {
    [...]
    // CM|PayPlaza Android POS Integration library
    implementation 'com.github.cmdotcom.android-pos-integration-sdk-
kotlin:androidposintegrationsdk-debug:<version-tag>'
}
```

> **i** <version-tag> should be change for the correct version number available in release notes

Import Integration SDK as a binary

- Download the desired .aar file from releases

- Place the downloaded file into your project. We recommend create a folder called 'lib' near 'src' folder of the app module.

- Add the import in your build gradle file:

```
dependencies {
    [...]
    // CM|PayPlaza Android POS Integration library
    implementation files('lib/<aar_file_name>')
}
```

Once this is done, your application should initialize the application with your android context application so PayPlaza library can communicate with PayPlaza Terminal application:

Code Sample. Initialization of PayPlaza Android POS integration library.

```
override fun onCreate() {
    super.onCreate()
    AndroidPosIntegration.init(this)

}
```

Once you have done this you can access the PosIntegrationService instance in your code by using

the `getInstance` method of the `AndroidPosIntegration` object in PayPlaza Android POS Integration Library

Code Sample. Getting the PayplazaPaymentService instace.

```kotlin
val service = AndroidPosIntegration.getInstance()
```

This PosIntegrationService has a list of methods that can be used to perform operation in PayPlaza environment. The methods are the following:

```kotlin
interface PosIntegrationService {
    fun doTransaction(data: TransactionData, callback: TransactionCallback)

    fun transactionStatuses(data: RequestStatusData, callback: StatusesCallback)

    fun getLastReceipt(options: LastReceiptOptions, callback: ReceiptCallback)

    fun getTerminalDayTotals(options: DayTotalsOptions, callback: ReceiptCallback)

    fun getTerminalInfo(callback: TerminalInfoCallback)
}
```

doTransaction: Is used to perform a transaction on PayPlaza app.

transactionStatuses: It is used to request status of a previous transaction performed by PayPlaza Terminal app.

getLastReceipt: It is used to request the receipt of the last transaction performed by Terminal application.

getTerminalDayTotals: It is used to request the totals information from the gateway.

getTerminalInfo: It is used to request information about the merchant shop in which the terminal is connected.

Passing data to the library for Transaction flow

Data can be passed to the PayPlaza Android POS Integration library by using a payment object with the appropriate data.

Code Sample. Transaction Data.

```kotlin
data class TransactionData(val type: TransactionType,
                           val amount: BigDecimal,
                           val currency: Currency,
                           val orderReference: String) {

    var language: String? = null
    var refundStan: String? = null
    var refundDate: Date? = null
    var isCaptureSignature = true
    var isShowReceipt = true

}
```

Values for the payment data attributes are specified following this table;

Attribute Name	Data Type/ Class	Required	Default
type	com.payplaza.androidposintegration.enums.TransactionType Only two options.	Mandatory	
amount	BigDecimal	Mandatory	
currency	java.util.Currency	Mandatory	
orderReference	String(max 14)	Mandatory	
language	String(2)	Optional	System Language
refundStan	String(6)	Optional	
refundDate	java.util.Date	Optional	
isCaptureSignature	boolean	Optional	true
isShowReceipt	boolean	Optional	true

Attributes explanation

type is a field in which you can specify if the transaction is a payment or a refund. An Enum is defined on the integration library so partners can choose between one of them.

amount is the transaction (payment or refund) amount. It is mandatory. It is represented as a BigDecimal including the decimal separator. For example, an amount in euro's is represented as "12,34", twelve euros and thirty four cents.

currency is a java.util.Currency object and indicates the currency in which the transaction will be

performed. There is a merchant shop configuration present on the gateway. This configuration contains the store currency among other parameters. Partners can retrieve that currency using the *getInfo* operation present on the library. PayPlaza terminal app can perform a transaction using a different currency from the default one, although in some cases that is not recommended (See ref for details).

orderReference is a mandatory string of up to 14 characters. You are free to put in any text. The PayPlaza app will return your order reference in the receipt lines (see recovery procedure). Transactions are not accepted in case the order reference is not present in the intent. Order reference needs to have a unique ID per POS for each transaction and date. Order reference may contain a maximum of 14 alphanumeric characters without special characters.

laguante is specified as a two-character string that represents a valid Locale language. Should be set using ISO-639-1 Code ("EN" for English, "NL" for Dutch, …) We currently support English, French, Dutch, German and Spanish. If you need other languages, please contact us. Language is configured in the merchant shop configuration in the Gateway. Only specify a language via the Intent if you need a language different from what is configured in the Gateway for your shop.

refundStan It is a String up to 6 characters. It represents the system trace audit number. It is only used for refund transactions and it can be found on the receipt for the previous transaction that has to be refunded

transactionDate is the date and time of the transaction. It is a java.util.Date object representing the date of the previous transaction. It is only used for refund transactions.

isCaptureSignature is a Boolean, indicating whether the Terminal application should perform on-screen signature capture for signature CVM transactions. When set to true and the transaction CVM is signature, the Terminal application will ask the user to sign on screen and send the signature back on the receipt to the partner application as a bitmap (see receipt handling on library for more details). This is the default behaviour. Partners can override that behaviour by setting this attribute to false.

isShowReceipt Indicates if the receipt(s) need(s) to be shown in PayPlaza Terminal app. Default value is true, so terminal is always showing the receipts. Partners can override that behaviour by setting that attribute to false.

Example of Transaction PaymentData object that can be used for the transaction:

- Transaction type: purchase
- Value: 5.95€
- Currency: EUR
- Order reference: 0303-000112

Code Sample. Transaction Data object creation for Payment Transaction

```
val data = TransactionData(TransactionType.PURCHASE, BigDecimal(5.95), Currency
.getInstance("EUR"),"0303-000112")
```

Transactions of type "Refund"

There are two possibilities when a refund transaction is sent as Transaction Type on the Transaction object: Standard refund transaction and STAN/TX refund transaction.

In the first case, no other information is required and the refund transaction will be performed with the amount sent.

In the second case, STAN and Transaction date information linked to a previous transaction, need to be provided to PayPlaza application in the following form:

TRANSACTION_STAN: string (6 characters. 6 STAN digits from the previous transaction)
TRANSACTION_DATE: string (8 characters with the following format: "dd/MM/yy")

If the transaction type received by PayPlaza app is "refund" and no other related information - like the STAN and transaction date - is present on the transaction object, the refund will be processed as standard refund flow. If transaction type is refund and STAN and Transaction date are present, refund flow will be linked to a previous existing transaction. If only one of the STAN and TX Date is present, an error will be returned by PayPlaza Terminal app.

Example of Transaction Data object for a Refund transaction

- Transaction type: refund
- Value: 5.95€
- Currency: EUR
- Order reference: 0303-000113
- STAN of previous transaction: 065987
- Date of previous transaction: February 1st, 2021

Code Sample. Transaction Data object creation for Refund Transaction

```kotlin
val data = TransactionData(
            TransactionType.REFUND,
            BigDecimal(5.95),
            Currency.getInstance("EUR"),
            "0303-000113"
        )

data.refundStan = "065987" // Can be found on the receipt
val dateFormat = SimpleDateFormat("dd/MM/yy")
data.refundDate = dateFormat.parse("01/02/21") // Can be found on the receipt
```

Receiving results

In order to receive the results for a transaction with the PayPlaza android POS integration library, partners have to define a call-back object that will be called back as soon as the result has been processed by the library.

Code Sample. Definition of TransactionCallback

```
Unresolved directive in EasyPOS Integration Guidelines for Sunmi P2 Devices-2.0.0.adoc
- include::../androidposintegration/src/main/java/com/payplaza/androidposintegration
/service/callback/TransactionCallback.kt[lines=6..15]
```

Object defined by Partner application needs to implement this interface. When library has the result of the transaction, it will call method *onResult* on the call-back and Partner application can get the result using the parameter on the call-back. For that, the class *TransactionResultData* is used

Values for the Transaction Result Data attributes are specified following this table:

Atribute Name	Data Type/ Class	present
transactionResult	com.payplaza.androidposintegration.enums.TransactionResult	always
orderReference	String(max 14)	always
amount	BigDecimal	always
authResponseCode	String	If Tx processed
cardEntryMode	String	If Tx processed
ecrId	String	If present on the receipt
processorName	String	If present on the receipt
transactionDateTime	java.util.Date	If present on the receipt
transactionId	String	If Tx Processed
cardScheme	String	If present on the receipt
aid	String	if present on the receipt
cardNumber	String	If present on the receipt
cardType	Integer	If Tx Processed
stan	String	If present on the receipt
merchantReceipt	com.payplaza.androidposintegration.beans.ReceiptData	If merchant receipt received from Gateway
customerReceipt	com.payplaza.androidposintegration.beans.ReceiptData	If customer receipt received from Gateway

Attributes Explanation

transactionResult: The resultCode you receive on the call-back. It is an enum defined on the library and can be one of the following:

```
SUCCESS ("Operation was successful")
CANCELED ("Operation was canceled")
AUTHORIZATION_FAILURE ("Operation did not have authorization")
AUTHORIZATION_TIMEOUT ("Timeout in last operation authorization")
CARD_BLOCKED ("Card used is blocked")
CARD_INVALID ("Card used is invalid")
DECLINED_BY_CARD ("Operation was declined by card")
INSUFICIENT_FUNDS ("Not sufficient funds to perform last authorization")
FAILED ("Operation failed")
AMOUNT_EXCEEDED ("Amount of las operation exceeded the limit")
HOST_BLOCKED_PRINT_RECEIPT ("Operation couldn t be performed. Receipt of the last
transaction pending")
REQUEST_RECEIPT("Request Receipt of current Transaction")
```

Each value has its own description so it is easy for users to understand what happened with the transaction.

orderReference: Is the order reference used in the transaction this value was received to perform the transaction and it is sent back to the partner application

amount: Is the amount received for the transaction.

authResponseCode: Is the authorization response code received on the transaction. It is only present if the transaction has been processed.

cardEntryMode: Is the card entry mode used in the transaction. It is present only if the transaction has been processed. It has one of the following values:

```
"CARD_ENTRY_MODE_MAG_STRIPE"
"CARD_ENTRY_MODE_MAG_STRIPE_FALL_BACK"
"CARD_ENTRY_MODE_ICC"
"CARD_ENTRY_MODE_CONTACTLESS"
"CARD_ENTRY_MODE_CONTACTLESS_MAG"
```

processorName: Is the name of the payment processor used in the transaction. It can also be found on the receipt lines.

transactionDateTime: It is the date and time of the transaction received from the gateway.

transactionId: It is the internal transaction id used in PayPlaza gateway

cardScheme: Is the Scheme of the card (Mastercard, Visa, ...) used in the transaction.

aid: Is the AID of the card used in the transaction

cardNumber: It is the masked pan of the card that has been used in the transaction.

cardType: It is an integer that indicates the type of the card used in the transaction. It can have the following values:

```
MAGNETIC CARD: 1
NFC CARD: 4
ICC (CHIP): 2
```

stan: Is the System Trace Audit Number of the transaction. It can also be found on the transaction.

merchantReceipt: It is the transaction receipt for the merchant. It is received if the merchant receipt was sent by the gateway. It is an object of the class *com.payplaza.androidposintegration.beans.ReceiptData*

customerReceipt: It is the transaction receipt for the customer. It is received if the customer receipt was sent by the gateway. Same as merchant receipt, it is an object of the class *com.payplaza.androdposintegration.beans.ReceiptData*

Receipt Data

The receipt data object received by partners has two attributes:

```
var receiptLines: Array<String>?
var signature: ByteArray?
```

receiptLines: is an array with the formatted lines of the receipt

signature: It is a Byte Array with the bitmap signature performed on Screen if the receipt needs a signature. It is only present if the signature has been done on the device. You can create a bitmap from the signature byte array present on the receipt using `BitmapFactory.decodeByteArray(⋯)`:

```
signatureInMerchantReceipt = BitmapFactory.decodeByteArray(
                        merchantReceipt.getSignature(),
                        0,
                        merchantReceipt.getSignature().length
                )
```

Simple Receipt Sample

```
Shopname

Terminal:       PPCCVBZS
Merchant:        654321
ECR:       E_SUNMI_P2lite
           _PL0919CQ00283
STAN:            130075


           CONTACTLESS
AID:      A0000000043060
                MAESTRO
Card:        ********0308
Cardnb:              0

Date: 19-07-21  09:52:52
AC:     F9A1D8E9A1014A60
Processor:          CCV
Auth. code:      FA5822
Auth. resp. code:    00


Amount:       EUR 2,49
REF:            7000135

    Payment approved

CARDHOLDER RECEIPT

      www.payplaza.com
```

Receipt with signature sample

```
Shopname

Terminal:        PPCCVBZS
Merchant:          654321
ECR:      E_SUNMI_P2lite
          _PL0919CQ00283
STAN:              130075


           CONTACTLESS
AID:      A0000000043060
                  MAESTRO
Card:        ********0308
Cardnb:                 0

Date: 19-07-21  09:52:52
AC:     F9A1D8E9A1014A60
Processor:            CCV
Auth. code:        FA5822
Auth. resp. code:      00

Amount:          EUR 2,49
REF:              7000135

    Payment approved

MERCHANT RECEIPT

CARDHOLDER SIGNATURE




.....................

      www.payplaza.com
```

⚠ It is responsibility of your application to print the signature in the appropriate position on the merchant receipt.

Error Cases

PayPlaza android POS integration library uses `onResult` method in the callback when there is no error on the transaction flow between the card, terminal and gateway. However, it could be the case when some situations may lead into errors in the flow (e.g.: connection lost). For those cases, the method `onError` is used. PayPlaza library also creates an ErrorCode Enum object with the following values:

```
NO_ERROR (0, "No error")
UNKNOWN_ERROR (-1, "Unknown error")
AMOUNT_INVALID (-2, "Amount used was invalid")
NO_INTERNET (-12, "Device is not connected to the network")
PRINTER_INIT_FAIL (-17, "Printer initialization failed")
POS_NOT_CONFIGURED (-18, "Device is not configured in gateway")
BAD_TIMEZONE (-24, "Timezone on the device is not correct")
HOST_NOT_CONNECTED (-25, "Cannot connect with the gateway")
MERCHANT_ORDER_REF_NOT_PRESENT (-29, "Order reference not present in request data")
TIMEOUT (-30, "Timeout reaching the gateway")
REPEATED_OPERATION (-31, "Transaction already in progress")
MERCHANT_ORDER_REF_TOO_LONG (-32, "Order ref exceeds allowed length")
AMOUNT_LIMIT_EXCEEDED (-33, "Transaction exceeds allowed limit")
TRANSACTION_TYPE_MISSING(-35, "Transaction Type must be sent")
CURRENCY_MISSING(-36, "Currency must be sent")
TRANSACTION_STATUS_ERROR (-50, "Status information not received")
INFO_REQUEST_FAILED (-51, "Info Request towards the gateway has failed")
```

Values also contain a description so users can know what went wrong with the previous transaction.

Dealing with unexpected scenarios

The PayPlaza app handles recovery of the payment transaction. This means, the PayPlaza app receives a payment request from partner application using the payment library and takes responsibility for properly handling that.

After processing, the app will return the result of the payment (which can have succeeded or failed) or it will return an ErrorCode result containing information on a failed operation.

But PayPlaza integration library can also call `onCrash` because of a crash on Terminal PayPlaza. If this is the case or if the Partner app crashes or is not capable of receiving the result for whatever reason, the PayPlaza integration library app allows Partner app to recover by requesting information of past transactions. This means we always need an order reference to be able to retrieve results of previous transactions. That is why order reference is a required attribute in the Transaction Data object needed by library to perform a transaction. Order reference does not need to be unique, because multiple transactions can refer to the same order reference. This happens when transactions fail, and new transactions are started with the same order reference for the same payment.

Requesting transaction information for an existing order reference

If the partner app needs to recover, or wants to retrieve old transaction results, the PayPlaza integration library allows the partner app to request the status of the previous transaction associated with an order reference. The method on the The PayPlaza app will return the results for that order reference that were sent previously.

This is flow for requesting the transaction status:
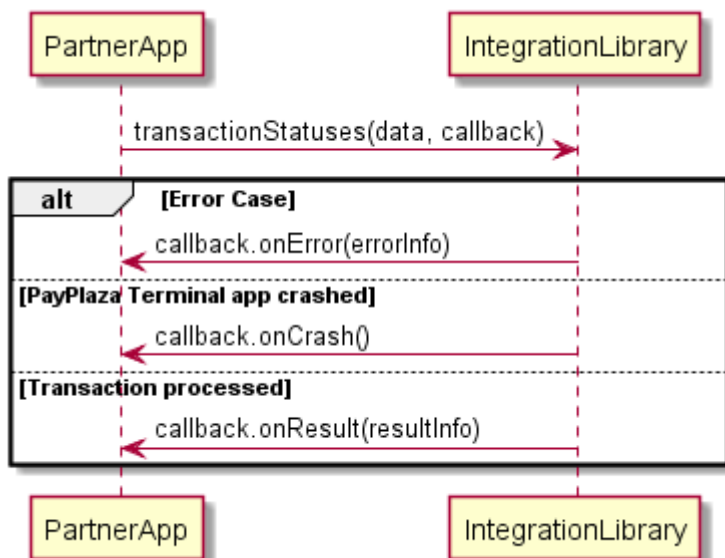
Because the order reference is not unique on a transaction level - the payment for a particular order reference can be retried a number of times, resulting in multiple transactions with the same order reference - a request for previous transaction results can result in multiple results. Of these, one may be a success result, the others are typically failure results.

The method to be used to retrieved this statuses is `transactionStatuses`. It receives an object of class

`RequestStatusData`:

> Unresolved directive **in** EasyPOS Integration Guidelines **for** Sunmi P2 Devices-2.**0.0**.adoc
> - include::../androidposintegration/src/main/java/com/payplaza/androidposintegration
> /beans/**RequestStatusData**.kt[lines=**2**..]

> ℹ️ Please note that you can create an object of class RequestStatusData with no info or an object passing the value for the merchant order reference. This is due to the fact that the transaction status can be requested with no specific order reference and you will received the status of all transactions performed by the Sunmi device that is being used in the current day.

page: It is an integer and can be used to indicate the number of the start page of the result (useful if there are too many transactions associated to one order reference)

size: It is an integer and can be used to indicate the number of statuses per reply (if 0 it is ignored).

sortFiled: It can be used to indicate in which field sorting needs to be performed.

sortValue: It can be used to indicate which type of sorting needs to be applied. Possible values are "asc" for ascending or "desc" for descending.

Example

```
val requestData = RequestStatusData("003-00012")
PayplazaPaymentService.transactionStatuses(requestData, statusesCallback)
```

StatusesCallback is used to receive the result in the same way that TransactionCallback is used to received the result of a transaction.

statusesCallback object also has a method `onResult` to receive the result of the statuses request to PayPlaza Terminal app.

> Unresolved directive **in** EasyPOS Integration Guidelines **for** Sunmi P2 Devices-2.**0.0**.adoc
> - include::../androidposintegration/src/main/java/com/payplaza/androidposintegration
> /service/callback/beans/../**StatusesCallback**.kt[lines=**6**..]

TransactionStatusesData class is defined as follows:

> Unresolved directive **in** EasyPOS Integration Guidelines **for** Sunmi P2 Devices-2.**0.0**.adoc
> - include::../androidposintegration/src/main/java/com/payplaza/androidposintegration
> /service/callback/beans/**TransactionStatusesData**.kt[lines=**8**..]

statusesInfo is a list of objects of class `TransactionStatusData`, which contains the information for one transaction:

```
Unresolved directive in EasyPOS Integration Guidelines for Sunmi P2 Devices-2.0.0.adoc
- include::../androidposintegration/src/main/java/com/payplaza/androidposintegration
/service/callback/beans/../json/TransactionStatusData.kt[lines=12..16]
```

The transaction Status Data contains the amount of the transaction, the currency used in that transaction, the result of the transaction, the type of the transaction and the receipt of the transaction. `statusesInfo` attribute contains a list of all transactions associated with the order reference used.

Receiving REQUEST_RECEIPT as transaction result

This is an special case of the recovery procedure. There are situations in which the PayPlaza Terminal has received the transaction result but it is not possible to get the receipts of the transaction (e.g.: because of a connection lost event). In this case, we cannot be certain of the transaction result until the receipt is received for the transaction.

If PayPlaza Integration library responds with that result, that means that Partner application should use the method `getLastReceipt` in order to complete the current transaction (see chapter 6 for more information about this operation).

6. Other Operations in SDK

Last Receipt functionality

It is also possible to request the receipt for the last transaction performed by PayPlaza terminal application. This can be used to retrieve a copy of the last receipt received on Partner application, but also to retrieve the receipt of the last transaction even if the result did not reach your application.

This is the flow for requesting the receipt of the previous transaction:
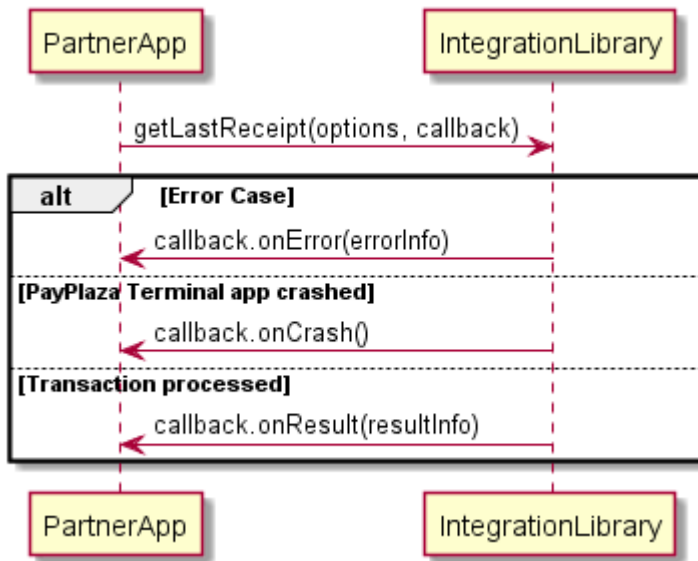
Request for the last receipt

```kotlin
val lastReceiptOptions = LastReceiptOptions(true)
paymentService.getLastReceipt(options, receiptCallback)
```

lastReceiptOptions: is an object containing the options for the operation:

```kotlin
val isShowReceipt: Boolean
```

You can select whether to show the receipt in PayPlaza Terminal application or not by setting that attribute to true (receipt shown in terminal app) or false (receipt not shown in terminal app).

receiptCallback is a ReceiptCallback object that will be used by the PayPlaza Android POS integration library to call Partner application back when the result of the operation and the receipt (if any) are received from the gateway.

```
fun onResult(data : LastReceiptResultData)

fun onError (error : ErrorCode)

fun onCrash ()
```

receiptCallback object must implement ReceiptCallback interface which means that it needs to implement the onResult, onError and onCrash methods (same as other call-back objects) **data** is an object of class LastReceiptResultData which has the following attribute

```
val receiptData : ReceiptData
```

This **receiptData** is the same as the receipt received on the transaction operations and contains the receipt received from the gateway.

If anything goes wrong in the call to the gateway, method onError will be triggered with the appropriate information, same as with the other functionality on the PayPlaza Library.

Day totals functionality

PayPlaza also provides integrators with the possibility to request the total amount money handled by the device in one day. We call that the Day totals of the terminal.

This is the flow for the get Terminal day totals functionality.
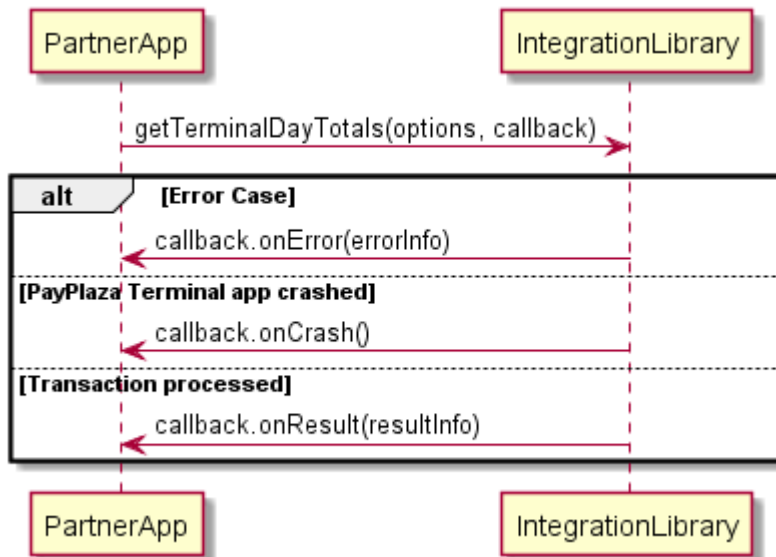
Partners can make use of that functionality by calling the day totals operation on the library service:

```
val dayTotalsOptions = DayTotalsOptions(true)
dayTotalsOptions.setFrom("10:00")
paymentService.getTerminalDayTotals(dayTotalsOptions, receiptCallback)
```

dayTotalsOptions is an object containing the options for the operation:

```
val isShowReceipt: Boolean
var from : String? = null
```

Same as with the last receipt functionality, you can select whether to show the receipt in terminal application or not. You can also select the hour of the present day from which the day totals will be calculated. If Partner application does not set that parameter, terminal will take `00:00` as the default hour and the day totals will be calculated from that time to the current time of the day in which the request is submitted.

receiptCallback is the same call-back that must be implemented for the last receipt functionality (see Last Receipt functionality chapter)

Request Info functionality

PayPlaza Terminal also provides this functionality to retrieve some useful information for partners. Terminal is configured on a merchant store on the gateway with some information (like the store name, address, language of the store, currency to use, ...). This information can be requested by partners by using this functionality present on the PayPlaza integration library.

Request info data sample

```
paymentService.getTerminalInfo(infoCallback)
```

In this case, there is no need to send extra information on the request, therefore, the only parameters present on the call.

infoCallback is an object of class TerminalInfoCallback. Partner object must implement the methods present on the interface same as with other functions present on the library. It contains the following functions

```
fun onResult(data : TerminalInfoData)

fun onError (error : ErrorCode)

fun onCrash ()
```

onResult method is called when PayPlaza Library has received the information requested by Partner application. Parameter *data* contains all the data that is sent by Terminal application:

```
var storeName: String? = null
var storeAddress: String? = null
var storeCity: String? = null
var storeZipCode: String? = null
var storeLanguage: String? = null
var storeCountry: String? = null
var storeCurrency: Currency? = null
var deviceSerialNumber: String? = null
var versionNumber: String? = null
```

storeName is the name of the store in which the terminal is configured on the gateway.

storeAddress is the address of the store in which the terminal is configured on the gateway.

storeCity is the city of the store in which the terminal is configured on the gateway.

storeZipCode is the zip code of the store in which the terminal is configured on the gateway.

storeLanguage is the default language of the store in which the terminal is configured on the gateway

storeCurrency is the default currency of the store in which the terminal is configured on the gateway

deviceSerialNumber is the serial number of the sunmi device in which the terminal application is running.

versionNumber is the version number of the terminal application.

> ℹ️ onResult method in TerminalInfoCallback interface is the same as previous callbacks on the other operations

7. Technical Support

For technical questions, please contact PayPlaza via ecr.developer@payplaza.com.

Appendix A: Simulator Test scenarios

In order to do a full system test there are several test scenario's that will result in a response message which is not successful. These messages are triggered by the usage of the following amounts and the corresponding results are being presented by the simulators PayPlaza created on its development platform. To make the PayPlaza platform as accurate as possible, PayPlaza has several simulators imitating the behavior of important processors.

> **ℹ** The only response code for a successful transaction is SUCCESS. With all other response codes the transaction has failed. Other (fail) messages may occur in production environment. To use contactless flow and error codes are required. The same codes are used but with a different amount

case	Value		Description	Result Code
	All	**Under NFC limit**		
1	€200,01	€20,01	The cardholder has reached the total amount limit of his/her card	AMOUNT_EXCEEDED
2	€200,02	€20,02	The cardholder does not have enough funds to buy the goods	INSUFFICIENT_FUNDS
3	€200,03	€20,03	The issuer was not able to respond in time	AUTHORIZATION_TIMEOUT
4	€200,04	€20,04	The issuer denies the transaction due to special conditions, i.e., fraud suspect	AUTHORIZATION_FAILURE
5	€200,05	€20,05	The card used is valid but has been blocked, i.e., exceeded the number of PIN tries	CARD_BLOCKED
6	€200,06	€20,06	The card used is not valid anymore, i.e., the card is expired	CARD_INVALID
7	€200,07	€20,07	The cardholder has entered a wrong PIN and the processor gives another opportunity to enter it correctly	PIN_INCORRECT_RETRY
8	€200,08	€20,08	This case represents the generic error case.	FAILED

9		€20.90	The issuer asks for further identification, in this case the PIN is needed	SCA_ADDITION_REQUIRED
10	SCA	€20.91	The issuer asks to change the interface used to perform a contact chip transaction	CONTACT_CHIP_REQUIRED

cases 7, 9 and 10 can be performed by sending the corresponding amount from partner application, but PayPlaza terminal application is going to handle the Pin Retry and SCA process by itself. This way, if the process is successful, partner Application will receive SUCCESS as a result code or the corresponding error code otherwise.

As mentioned before, the PayPlaza platform has several different Processor simulators, which try to mimic the real world behavior of actual processors in production. As a result it should be noted that not all cases are supported by every processor, which will be reflected in the behavior of the processor simulators as stated below.

Case	OmniPay Simulator	CCV Simulator	Equens Simulator	SIX Simulator	Amex Simulator
1	YES	YES	YES	YES	Not Supported
2	YES	YES	YES	YES	Not Supported
3	YES	YES	YES	Not Supported	Not Supported
4	YES	YES	YES	YES	YES
5	YES	YES	YES	YES	YES
6	YES	YES	YES	YES	Not Supported
7	YES	Not Supported	Not Supported	YES	YES
8	YES	YES	YES	YES	YES
9	YES	YES	YES	YES	Not Supported
10	YES	YES	YES	YES	Not Supported