Developer Guide for setting up payments on Jio Set-Top Box

Version: **1.37**
Published Date: **29-07-2024**
Document Classification: **Confidential**

Table of Contents

Introduction

Scope

JioPay SDK is for developers who want to collect payments from users for their applications on Jio STB. Developer needs to be on-boarded and get their app fingerprint whitelisted to call JioPay SDK.

App users (customers) has the choice to make payments using any of the following:

- QR Code (using any UPI app to scan and pay).
- Credit/Debit Cards (user gets option to pay via new card or pay effortlessly using already saved cards from previous payments).
- UPI Collect (user gets option to pay via new UPI ID or pay effortlessly using already saved UPI IDs from previous payments).

Intended audience

This document is intended for developers who want to enable JioPay services and collect payment from users for their applications available on Jio STB.

Types of Supported Payments

JioPay currently supports the following types of payments on Jio STB:

- One-time Payments.
    - Consumable – can be purchased multiple times by a user.
    - Non-consumable – can be purchased only once in lifetime by a user.
- Recurring Subscription Payments.

Getting Started

App On-boarding

To get your application on-boarded with JioPay, write an email:

To	jiostore.support@ril.com
Subject	**<App Name> - JioPay integration on Jio STB | Application details for on-boarding**
Body	• App name (name of your application) • Package name (unique package name of your application) • Callback URL (on which all server-to-server callbacks would be sent) • App signature (SHA256 of your app signing certificate. Guide: https://developers.google.com/android/guides/client-auth

SKU On-boarding

To start selling products, each product(s) need to be pre-registered with the following details via email:

To	jiostore.support@ril.com
Subject	**<App Name> - JioPay integration on Jio STB | SKU details for on-boarding**

Body	• App name (matching with the app name on-boarded on previous step)
	• Package name (matching with the package name on-boarded on previous step)
	• SKU Title (title of your product, visible to user on TV while making payment)
	• SKU Description (brief description of your product, visible to users via select channels only)
	• SKU Amount (amount of your product in INR)
	• Type of purchase (any one from below list)
	  ○ Consumable
	  ○ Non-consumable
	  ○ Subscription (with length of subscription – in days)

Next Steps

On successful on-boarding of your application and SKUs in test environment, you'll receive the following details:

- **Client ID**: This is required to verify secure information sent via S2S call to your server. Client ID is confidential and should be stored securely on your servers. It is recommended to refrain from storing the Client ID at your client side code.
- **SKU ID**: This is the list of SKU IDs of the products you can sell.

Installing JioPay SDK

- Copy the below **aar** and **pom** files to a new folder (create new folder if required) located inside <Project Directory>\app\BillingService\com\jio\stbstart\in_app_payment_client_sdk\9.x.x
    - in_app_payment_client_sdk-9.x.x.aar
    - in_app_payment_client_sdk-9.x.x.pom
- Add the below code to your project's top level **build.gradle** file:

```
allprojects {
    repositories {
        google()
        mavenCentral()
        maven {
            url "BillingService"
        }
    }
}
```

- Add the below code to module level **build.gradle** file:

```
dependencies {
    implementation 'androidx.core:core-ktx:1.6.0'
    implementation 'androidx.appcompat:appcompat:1.2.0'
    implementation 'androidx.constraintlayout:constraintlayout:2.0.4'
    implementation 'org.jetbrains.kotlin:kotlin-stdlib-jdk7:1.7.0'
    implementation 'com.jio.stbstart:in_app_payment_client_sdk:9.x.x'
}
```

- Note: Please check for duplicates and avoid them for successful building of project.
- Clean project and rebuild.

Initializing Payments via JioPay SDK

To launch SDK and start payments, you need to provide following inputs in OrderInfo class using OrderInfoBuilder like below:

```
val orderInfo = OrderInfoBuilder()
    .setSkuId(skuId)
    .setOrderId(orderId)
    .setTimestamp(System.currentTimeMillis().toString())
    .build()
val intent = Intent(this@MainActivity, JioPayActivity::class.java)
intent.putExtra("ORDER_INFO", orderInfo)
startActivityForResult(intent, JiopayResponseCodes.REQUEST_PAY)
```

Classes and Methods

Class: OrderInfo

This class contains various parameters required to successfully initialize a payment:

skuId (mandatory)	**Integer**: This member variable holds the SKU ID.
orderId (mandatory)	**String**: This member variable holds the order ID of the transaction. This must be unique each time for each transaction. This should be generated by your server.
timestamp (mandatory)	**String**: This value sets the time when the order was created. This is in milliseconds converted to string.
checksum (optional)	**String**: This optional parameter can be sent to secure the payload.
obfUserId (optional)	**String**: Obfuscated User ID.
udf1 (optional)	**String**: User defined field 1.
udf2 (optional)	**String**: User defined field 2.
startDate (conditional)	**Long**: Start date (in milliseconds) for recurring payments for subscription products. (Not applicable for one-time payments)
trialPeriod (conditional)	**Integer**: Trial period (in days) for free trial subscription products (trial period amount = ₹0). (Not applicable for one-time payments)
amountForDisplay (conditional)	**String**: Amount (in ₹) for displaying during recurring payments Confirm Mandate screen. This is for display purpose only. (Not applicable for one-time payments)
subscriptionPeriodForDisplay (conditional)	**String**: Subscription period (in days) for displaying during recurring payments Confirm Mandate screen. This is for display purpose only. (Not applicable for one-time payments)

Class: OrderInfoBuilder

This is a builder class that helps to build the OrderInfo which acts as input to JioPay service.

Usage: OrderInfoBuilder.build() returns OrderInfo

Public Methods		Description
Void	setSkuId(skuId: Int)	This method sets the SKU ID for the payment. This is mandatory for all payments.
Void	setOrderId(orderId: String)	This method sets the unique order ID for the payment. This is mandatory for all payments.

Void	setTimestamp(timestamp: String)	This method sets the timestamp when the order is started. This is mandatory for all payments.
Void	setChecksum(checksum: String?)	This method sets checksum for the order. Checksum can be generated using generatePayLoadChecksum(**CLIENT_ID**, payLoad^)
Void	setObfUserId(obfUserId: String?)	This method sets the obfuscated User ID.
Void	setUdf1(udf1: String?)	This method sets the user defined field 1.
Void	setUdf2(udf2: String?)	This method sets the user defined field 2.
Void	setStartDate(startDate: Long?)	This method sets the start date during recurring payments for subscription products.
Void	setTrialPeriod(trialPeriod: Int?)	This method sets the number of days users get free trial (for subscription products). Accepted values of trialPeriod is >=3.
Void	setAmountForDisplay(afd: String?)	This method sets the amount (in ₹) for display purposes in Confirm Mandate screen during recurring payments for Subscription products (as per on boarded SKU amount).
Void	setSubscriptionPeriodForDisplay(spfd: String?)	This method sets the subscription period (in days) for display purposes in Confirm Mandate screen during recurring payments for Subscription products (as per on boarded SKU period).
OrderInfo	build()	This function initializes the OrderInfo object for payment.

^Payload pipe separated string: **timestamp|orderId|skuId|startDate|trialPeriod** (startDate and trialPeriod are conditional)

Class: JioPayActivity

This Activity is used to invoke JioPay Service for initiating a payment. It takes input of OrderInfo.

Class: JiopayResponseCodes

This class contains various constants for holding Request and Result Codes.

REQUEST_PAY	This is the request code for initiating pay request.
RESULT_SUCCESS	This is the result code of successful payment.
RESULT_FAILURE	This is the result code of failed payment.
RESULT_CANCELLED	This is the result code of cancelled payment.
RESULT_TIMEDOUT	This is the result code of timed-out payment.
TXN_STATUS	This is key to obtain transaction status from result Intent.
RESP_MESSAGE	In case of failure, this key will contain the explanation of failures (in string format).
RETRY_FLAG	This is a flag when result is not SUCCESS. Possible values are “TRUE” and “FALSE”. When user submits intent to retry a transaction, (i.e. “RETRY_FLAG” is “TRUE”), client can initiate pay request again with new orderId. Kindly refer to Sample App code for implementation technique.

Obtaining Transaction Result

There are two ways to obtain transaction results:

- **Via SDK^** (should not be considered for Order fulfilment, kindly rely only on S2S calls)
  - Transaction Success
  - Transaction Failure
  - Transaction Cancelled
  - Transaction Timed Out
  - Transaction Retry
- **Via S2S**
  - Transaction Success
  - Transaction Failure

^ In Transaction Success/Failure cases, it is advised to depend on S2S API response rather than SDK response.

Transaction Result via SDK

```
override fun onActivityResult(requestCode: Int, resultCode: Int, data:
Intent?) {
    super.onActivityResult(requestCode, resultCode, data)
    val resultOfPayment =
data?.extras?.getParcelable<TransactionResponse>(JiopayResponseCodes.TXN_
RESPONSE)
    val message =
data?.extras?.getString(JiopayResponseCodes.RESP_MESSAGE)
    val status = data?.extras?.getString(JiopayResponseCodes.STATUS)
    val retryFlag =
data?.extras?.getString(JiopayResponseCodes.RETRY_FLAG)
    when (resultCode) {
        JiopayResponseCodes.RESULT_SUCCESS -> {
            Log.d(TAG, "Transaction Success Received via SDK")
        }
        JiopayResponseCodes.RESULT_FAILURE -> {
            Log.d(TAG, "Transaction Failure Received via SDK")
            if (retryFlag?.contains("TRUE", true) == true) {
                Handler(Looper.getMainLooper()).postDelayed({
                    //start Payment again
                }, 2000) //Please ensure there is a delay of at least
2000ms to avoid android.os.DeadObjectException
            }
        }
        JiopayResponseCodes.RESULT_CANCELLED -> {
            Log.d(TAG, "Transaction Cancelled Received via SDK")
            if (retryFlag?.contains("TRUE", true) == true) {
                Handler(Looper.getMainLooper()).postDelayed({
                    //start Payment again
                }, 2000)
            }
        }
        JiopayResponseCodes.RESULT_TIMEDOUT -> {
            Log.d(TAG, "Transaction Timed Out Received via SDK")
            if (retryFlag?.contains("TRUE", true) == true) {
                Handler(Looper.getMainLooper()).postDelayed({
                    //start Payment again
                }, 2000)
            }
        } else -> {
```

```
                Log.d(TAG, "Nothing Received via SDK")
            }
        }
    }
```

Class: Transaction Response

This class contains various information related to a payment. You'll receive these information via SDK only in case of successful payments. In other cases, you'll receive null.

code	Status code: 1 for success.
message	Status message.
status	Status of transaction: SUCCESS or FAILED.
orderId	Developer generated order ID with which payment was initiated.
txnId	Payment Gateway side Transaction ID.
amount	Amount of transaction.
timestamp	Timestamp (in millisecond) at which transaction happened.
startDate	Start Date of the purchase at which transaction occurred.
endDate	End Date calculated based on the product validity.
skuId	SKU ID for which payment was initiated.
currency	Currency of the transaction.
responseSignature	Signature to verify signed response.
obfUserId	Obfuscated User ID with which payment was initiated via SDK.
udf1	User Defined Field 1 with which payment was initiated via SDK.
udf2	User Defined Field 2 with which payment was initiated via SDK.

Transaction Result via S2S

There are two ways to obtain transaction status of a particular order:

- JioPay server sends Transaction Status Notification Callback to Developer server
- Developer server fetches the Transaction status via Transaction Status API.

Transaction Status Notification Callback to Developer server

This API will send the transaction notification to your server URL (which was on-boarded). This is used to confirm when:

- Any payment is completed (Success or Failure)
- When a recurring subscription is cancelled (by API or due to renewal failures)
- When user changes recurring subscription status (e.g. suspend, revoke, resume from UPI App/Bank)
- When order confirmation (of product delivery) is not SUCCESS (to inform refund status).

Request Endpoint:

The API endpoint will be your server URL which was shared by you during app on-boarding.

Request Body:

The request will contain the following information as part of payload in JSON format:

code (mandatory)	**String**: Status code.
message (mandatory)	**String**: Status message.
orderId (mandatory)	**String**: Order ID used for the transaction (generated uniquely by you). In case of Recurring Renewal Payments, this order ID will be auto-generated by JioPay and sent in the callback details.
txnId (mandatory)	**String**: ID of the transaction.
transactionStatus (mandatory)	**String**: This is the status of the delivery of product or service used to check the status of the transaction. Possible values are: • SUCCESS • FAILURE
amount (mandatory)	**String**: Amount of the transaction.
timestamp (mandatory)	**String**: Timestamp of the API call.
startDate (mandatory)	**String**: Date when the transaction was performed.
endDate (mandatory)	**String**: End date of the subscription.
skuId (mandatory)	**String**: SKU ID used for the transaction.
checksum (mandatory)	**String**: Checksum generated exclusively for you based on payload and your Client ID. Payload is a pipe separated string of format: **orderId|txnId|transactionStatus|code|message|amount|timestamp|startDate|endDate|skuId|currency|rechargeStatus|refundStatus**
currency (mandatory)	**String**: Currency used for transaction (e.g. INR, USD, etc.)
paymentMode (mandatory)	**String**: Payment mode selected by user to do the payment. Possible values are: • UPI • CC • DC
obfUserId (optional)	**String**: Obfuscated User ID (if sent during SDK initiate payment).
udf1 (optional)	**String**: User Defined Field 1 (if sent during SDK initiate payment).
udf2 (optional)	**String**: User Defined Field 2 (if sent during SDK initiate payment).

skuType (conditional)	**String**: Possible values are: • C: in case of consumable type SKU • NC: in case of non-consumable type SKU • SUBO: in case of recurring type SKU
skuValidity (conditional)	**String**: No. of days SKU is valid for based on configuration during SKU on-boarding. This is independent of trial period days.
skuValidityUOM (conditional)	**String**: Day
parentOrderId (conditional)	**String**: 1st order ID used for making transaction for a recurring payment. For subsequent renewals, the 1st txn order ID in the subscription tree will be maintained as parentOrderId at server side to maintain the hierarchy and same thing will be sent.
txnType (conditional)	**String**: Possible values are: • PURCHASE: for one time payments • SUB: for recurring payments (both one time and renewal) • MANDATE: for recurring payments (free trial payments)
subscStatus (conditional)	**String**: Possible values are: • INITIATED: When subscription is initiated before payment • ACTIVE: When payment is successful. • CANCELLED: When user cancels subscription, when sent purchaseAck=FAILED, server cancels it. • INACTIVE: Status of the subscription when not active. Happens when renewal happened for this subscription. Old record will be inactive and new will be active if renewal done successfully. • EXPIRED: When subscription expired from PG. • SUSPENDED: When user suspends the subscription (this is temporary and becomes ACTIVE if user enables the subscription via UPI/Bank App). • FAILED: When unable to subscribe or unable to renew.
planAmount (conditional)	**String**: SKU Amount based on configuration during SKU on-boarding.
txnSubType (conditional)	**String**: Possible values are: • PURCHASE: for onetime payments only (SKU Type C / NC). • TRIALPERIOD: for free trial enabled subscription payments. • SUBSCRIPTION: for subscription without trial period or for subscription just after trial period.
refundStatus (optional)	**String**: Refund status of the particular transaction. Possible values are: • SUCCESS • FAILED • PENDING
rechargeStatus (optional)	**String**: Recharge status. Possible values are: • SUCCESS • FAILED
trialDays (conditional)	**Integer**: Number of trial days for a free subscription.
parentOrderId (optional)	**String**: The 1st order ID in the subscription tree.
errorCode (optional)	**String**: null if transaction is success, else non null.
errorReason (optional)	**String**: null if transaction is success, else non null.

Transaction Status API

Developer can also get the transaction status by using this API.

Request Endpoint:

https://<environment-url>/cr/v1/jiopay/stbservice/txn/get

Request Body:

The request body should contain the following information as part of payload in JSON format:

orderId (mandatory)	**String**: Order ID of the transaction.
appPackageName (mandatory)	**String**: App package name of the order.
checksum (mandatory)	**String**: Checksum generated using generatePayLoadChecksum(**CLIENT_ID**, payLoad^)
parentOrderId (optional)	**Boolean**: If you send the parent Order ID of the subscription tree, send true. In such cases, you will receive the transaction status of the latest child transaction.

^Payload pipe separated string: **orderId|appPackageName**

Response Body:

You will receive transaction details for that particular orderId in JSON format:

code (mandatory)	**String**: Status code.
message (mandatory)	**String**: Status message.
orderId (mandatory)	**String**: Order ID used for the transaction (generated uniquely by you). In case of Recurring Renewal Payments, this order ID will be auto-generated by JioPay and sent in the callback details.
txnId (mandatory)	**String**: ID of the transaction.
transactionStatus (mandatory)	**String**: This is the status of the delivery of product or service used to check the status of the transaction. Possible values are: • SUCCESS • FAILURE
amount (mandatory)	**String**: Amount of the transaction.
timestamp (mandatory)	**String**: Timestamp of the API call.
startDate (mandatory)	**String**: Date when the transaction was performed.
endDate (mandatory)	**String**: End date of the subscription.
skuId (mandatory)	**String**: SKU ID used for the transaction.
checksum (mandatory)	**String**: Checksum generated exclusively for you based on payload and your Client ID. Payload is a pipe separated string of format: **orderId|txnId|transactionStatus|code|message|amount|timestamp|startDate|endDate|skuId|currency|rechargeStatus|refundStatus**
currency (mandatory)	**String**: Currency used for transaction (e.g. INR, USD, etc.)
paymentMode (mandatory)	**String**: Payment mode selected by user to do the payment. Possible values are: • UPI • CC • DC
obfUserId (optional)	**String**: Obfuscated User ID (if sent during SDK initiate payment).
udf1 (optional)	**String**: User Defined Field 1 (if sent during SDK initiate payment).
udf2 (optional)	**String**: User Defined Field 2 (if sent during SDK initiate payment).
skuType (conditional)	**String**: Possible values are: • C: in case of consumable type SKU

	• NC: in case of non-consumable type SKU • SUBO: in case of recurring type SKU
skuValidity (conditional)	**String**: No. of days SKU is valid for based on configuration during SKU on-boarding. This is independent of trial period days.
skuValidityUOM (conditional)	**String**: Day
parentOrderId (conditional)	**String**: 1st order ID used for making transaction for a recurring payment. For subsequent renewals, the 1st txn order ID in the subscription tree will be maintained as parentOrderId at server side to maintain the hierarchy and same thing will be sent.
txnType (conditional)	**String**: Possible values are: • PURCHASE: for one time payments • SUB: for recurring payments (both one time and renewal) • MANDATE: for recurring payments (free trial payments)
subscStatus (conditional)	**String**: Possible values are: • INITIATED: When subscription is initiated before payment • ACTIVE: When payment is successful. • CANCELLED: When user cancels subscription, when sent purchaseAck=FAILED, server cancels it. • INACTIVE: Status of the subscription when not active. Happens when renewal happened for this subscription. Old record will be inactive and new will be active if renewal done successfully. • EXPIRED: When subscription expired from PG. • SUSPENDED: When user suspends the subscription (this is temporary and becomes ACTIVE if user enables the subscription via UPI/Bank App). • FAILED: When unable to subscribe or unable to renew.
planAmount (conditional)	**String**: SKU Amount based on configuration during SKU on-boarding.
txnSubType (conditional)	**String**: Possible values are: • PURCHASE: for onetime payments only (SKU Type C / NC). • TRIALPERIOD: for free trial enabled subscription payments. • SUBSCRIPTION: for subscription without trial period or for subscription just after trial period.
refundStatus (optional)	**String**: Refund status of the particular transaction. Possible values are: • SUCCESS • FAILED • PENDING
rechargeStatus (optional)	**String**: Recharge status. Possible values are: • SUCCESS • FAILED
trialDays (conditional)	**Integer**: Number of trial days for a free subscription.
parentOrderId (optional)	**String**: The 1st order ID in the subscription tree.
errorCode (optional)	**String**: null if transaction is success, else non null.
errorReason (optional)	**String**: null if transaction is success, else non null.

Order Confirmation

Once you initiate any payment request by initializing SDK, you get the result back as mentioned above. After you've received a successful transaction result, you're expected to deliver the product / service to the customer. Once the product / service is delivered, you need to send the purchase confirmation (acknowledgement) SUCCESS to JioPay server using the Purchase Ack API. If no confirmation is received from your end within 48 hours of purchase (or, if you send purchase acknowledgement FAILURE), refund process is initiated for that customer (if there are any active subscriptions, those will get cancelled).

Note: To avoid subscription cancellations / payment refunds due to system issues, kindly add RETRY logic to send purchase acknowledgement.

Request Endpoint:

https://<environment-url>/cr/v1/jiopay/stbservice/purchase/ack

Request Body:

The request body should contain the following information as part of payload in JSON format:

orderId (mandatory)	**String**: Order ID of the transaction.
purchaseAck (mandatory)	**String**: Accepted values are: • SUCCESS • FAILURE
checksum (mandatory)	**String**: Checksum generated using generatePayLoadChecksum(**CLIENT_ID**, payLoad^)

^Payload pipe separated string: **orderId|purchaseAck**

Response Body:

You will receive purchase acknowledgement status for that particular orderId in JSON format:

code (mandatory)	**Integer**: Status code, 200 for success.
status (mandatory)	**String**: Possible values are: • SUCCESS • FAILURE In case of failure, refund would be initiated.
message (optional)	**String**: If any failure occurs, it can be referred to understand the reason of failure.

Error Codes and Messages:

Scenario	HTTP Status	Code	Status	Message	Should you retry?
Successful Processing of Acknowledgement	200	200	SUCCESS	Purchase ack is processed successfully	No
Purchase Ack sending to failed transaction	400	400.115	FAILURE	Purchase ack sent for failed payment transaction	No
Duplicate Purchase Ack or System updates the Purchase Ack status after 48 hrs	400	400.14	FAILURE	Purchase ack already exist for this order	No
Invalid checksum	400	400.35	FAILURE	Invalid checksum	Yes

Order ID does not exist in JioPay system	400	400.7	FAILURE	No transaction details found	No
Server Processing Failures or Downstream request processing Failures or DB Failures	500	500 or 500.x	FAILURE	Internal Server Error	Yes

Subscription Status

You can use this API to check subscription status (only for recurring type payments) for a given customer.

Request Endpoint:

https://<environment-url>/jiopay-bill-service/subscriptions/v1/status/serv

Request Body:

The request body should contain the following information as part of payload in JSON format:

connectionId (mandatory)	**String**: Subscriber ID of the customer.
skuId (mandatory)	**String**: SKU ID for the subscription.
checksum (mandatory)	**String**: Checksum generated using generatePayLoadChecksum(**CLIENT_ID**, payLoad^)
parentOrderId (optional)	**String**: If multiple active subscriptions exist, pass this parameter to check specific subscription status (include in checksum if sent).

^Payload pipe separated string: **connectionId|skuId|parentOrderId**

Response Body:

You will receive subscription status for that particular user in JSON format:

code (mandatory)	**Integer**: Status code, 200 for success.
message (mandatory)	**String**: If any failure occurs, it can be referred to understand the reason of failure.
skuId (mandatory)	**String**: SKU ID.
skuTitle (mandatory)	**String**: SKU Title (as on-boarded).
appName (mandatory)	**String**: App Name (as on-boarded).
amount (mandatory)	**String**: SKU Amount (as on-boarded).
endDate (mandatory)	**String**: Subscription end date.
skuImage (mandatory)	**String**: Image of the SKU (as on-boarded).
subscriptionFrequency (mandatory)	**String**: Frequency of subscription (as on-boarded).
subscriptionFrequencyUnit (mandatory)	**String**: Frequency unit of subscription (days).
prenotifyStatus (mandatory)	**String**: Prenotification status of subscription. Possible values are: • SENT • NOT_SENT

Cancel Subscription

You can use this API to cancel any active subscription (only for recurring type payments) for a given customer.

Request Endpoint:

https://<environment-url>/jiopay-bill-service/subscriptions/v1/cancel/serv

Request Body:

The request body should contain the following information as part of payload in JSON format:

connectionId (mandatory)	**String**: Subscriber ID of the customer.
skuId (mandatory)	**String**: SKU ID for the subscription.
checksum (mandatory)	**String**: Checksum generated using generatePayLoadChecksum(**CLIENT_ID**, payLoad^)
parentOrderId (optional)	**String**: If multiple active subscriptions exist, pass this parameter to cancel specific subscription (include in checksum if sent).

^Payload pipe separated string: **connectionId|skuId|parentOrderId**

Response Body:

You will receive cancel subscription status for that particular user in JSON format:

code (mandatory)	**Integer**: Status code, 200 for success.
message (optional)	**String**: If any failure occurs, it can be referred to understand the reason of failure.
status (mandatory)	**String**: Indicative status.

Error Codes and Messages:

Code	Message
400.35	Invalid checksum
400.4	No pg mapping is present for the corresponding pg name
400.46	Invalid skuid
400.5	No record found
400.6	No app record found for the given package name
400.62	No ACTIVE Subscription Exists
400.63	Cancel subscription failure
500.2	Something went wrong. Please try after sometime.
500.5	Unknown error occured

Checksum Generation Guidelines

The algorithm used to generate checksum is **HMACSHA256**. Below is the pseudo code snippet used to generate checksum in Java^:

```
String generatePayLoadChecksum (param1, param2) {
    String key = param1;
    String message = param2;
    Mac hasher = Mac.getInstance("HmacSHA256");
    hasher.init(new SecretKeySpec(key.getBytes(), "HmacSHA256"));
    byte[] hash = hasher.doFinal(message.getBytes()); // to lowercase
hexits
    DatatypeConverter.printHexBinary(hash);
}
```

^For other platforms, refer https://gist.github.com/jasny/2200f68f8109b22e61863466374a5c1d

To generate checksum for SDK input/S2S APIs, developer can use below function:

```
generatePayLoadChecksum(CLIENT_ID, payLoad)
```

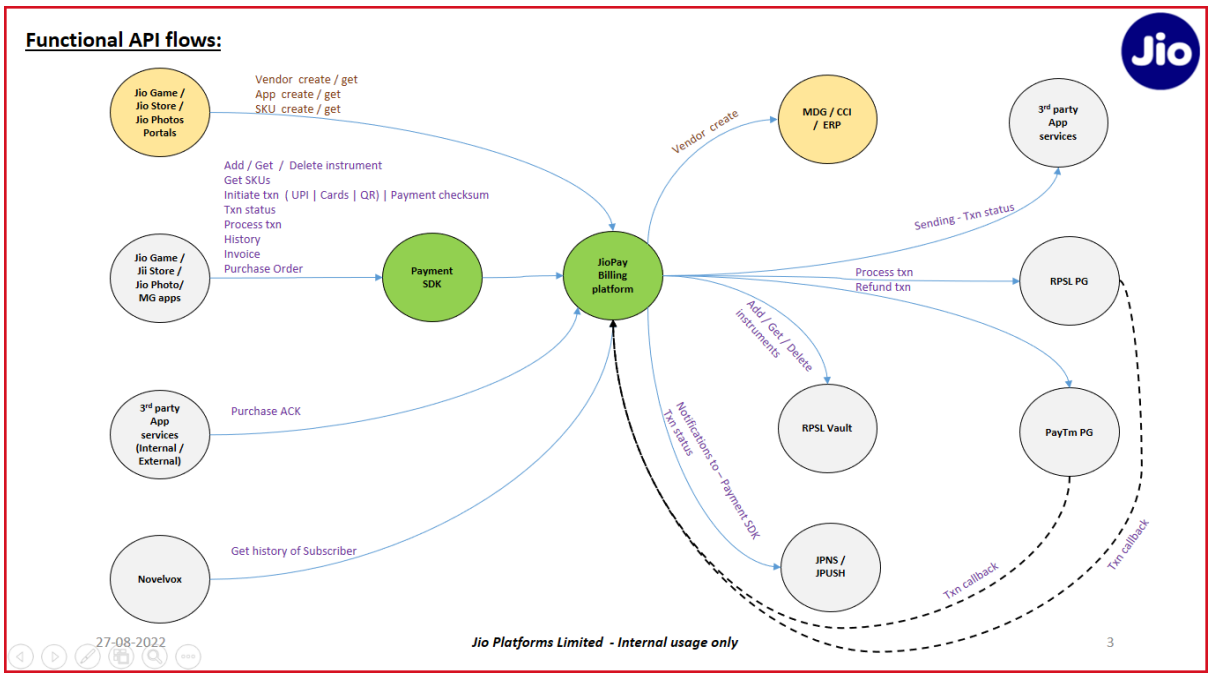
payLoad format is pipe separated string as described in individual API sections.

Frequently Asked Questions

Q. What is the sample flow for integrating payments in STB?

A. The sample flow for successfully integrating payments is:

- Developer sends email for App On-boarding and receives a secure unique client ID.
- Developer sends product details for SKU On-boarding and receives list of on-boarded SKUs.
- Developer hosts their SKUs at their server, fetches and lists them on their app's screen.
- Developer initiates payment via SDK on device via **startActivityForResult** with a unique order ID.
- JioPay handles the complete payment journey of user.
- JioPay SDK returns transaction status in **onActivityResult**.
- For transaction success/failure cases, developer verifies actual payment status using S2S API calls.
- Developer unlocks the premium benefits to the user and sends a purchase acknowledgement using S2S API call. If purchase acknowledgement is not sent within 48 hours, money will be refunded back to customer.



Q. What is the **minSdkVersion** supported by JioPay SDK?

A. The **minSdkVersion** supported by JioPay SDK is **24**.

Q. How to test the payments feature integration in test environment?

A. To test payments feature integration in test environment, you'll receive a test APK from integration support. You need to follow the below steps to uninstall the production version of JioPay and install this build on your STB:

- Ensure that your STB has ADB enabled
- Connect your STB to machine using **adb connect <IP address>**
- Once connected successfully, run the commands sequentially:
    - **adb uninstall com.jio.jiopay**

- **adb install <File path of developer build>** e.g. adb install C:\Users\abc\Desktop\bs-stb-sit-debug-v9.x.x.apk

You're ready to launch payments from your APK in test environment.

Q. I'm able to start payment in test environment, but getting error screen.

A. This can happen due to multiple reasons:

- Your application package has changed: Ensure that your app package name remains the same which was on-boarded.
- SKU does not belong to your package: Ensure that you use the SKU IDs assigned to you.
- JioPay test app is not properly installed: Ensure that you have received success confirmation after performing **adb install**.
- Signature mismatch: Ensure that you always sign your APK using the same keystore file. To get your signature updated in JioPay database, connect to your integration support.
- Duplicate Order ID: Ensure that you send a unique order ID every time you initiate a payment through the SDK.
- Payment of Consumable SKUs: Consumable SKUs are designed for non-recurring payments. After each consumable payment, you need to send Order Confirmation via S2S before making a purchase of the same SKU again.
- Payment of Non-consumable SKUs: Non-consumable SKUs are one-time payments. These are meant for only one payment in life-time.

Q. I'm able to start payment in production environment, but getting error screen.

A. This can happen due to multiple reasons in addition to the reasons stated in test environment:

- Production payments are not supported in ADB enabled and rooted devices: To get the debug JioPay APK of production, connect to your integration support.

Q. Can I get the description of errors in case of Failure, Cancellation, Time-out, etc.?

A. **JiopayResponseCodes.RESP_MESSAGE** holds the descriptive message of errors (in string format). Same can be referred for this case. However, the below list is not exhaustive and hence you are advised not to show these messages to customers on TV screen.

Result Codes	Response Messages	Scenario	New Response
TRANSACTION_SUCCESSFUL	Success Messages	User does a successful transaction.	Result Of Payment = Transaction Response object Result Code = 280 Status Of Payment = SUCCESS Message Of Payment = null
TRANSACTION_FAILED	Can vary depending on actual scenario.	Failure in downstream systems.	Result Of Payment = null Result Code = 281 Status Of Payment = FAILED Message Of Payment = Can vary depending on actual scenario.
TRANSACTION_FAILED	SSO_FAILURE	User Authentication Failure.	Result Of Payment = null Result Code = 281 Status Of Payment = FAILED

			Message Of Payment = SSO_FAILURE
TRANSACTION_CANCELLED	NOT_INITIATED	User exits by back button without initiating transactions.	Result Of Payment = null Result Code = 286 Status Of Payment = CANCELLED Message Of Payment = NOT_INITIATED
TRANSACTION_CANCELLED	INTIATED_TXN_CANCELLED	User exits by back button after initiating transactions.	Result Of Payment = null Result Code = 286 Status Of Payment = CANCELLED Message Of Payment = INITIATED_TXN_CANCELLED
TRANSACTION_TIMEOUT	INTIATED_TXN_TIMEDOUT	User initiates the transaction on SDK and does not do any action.	Result Of Payment = null Result Code = 288 Status of Payment = TIMEDOUT Message Of Payment = INITIATED_TXN_TIMEDOUT

For any other queries, write to jiostore.support@ril.com.