भारतीय राष्ट्रीय भुगतान निगम

NATIONAL PAYMENTS CORPORATION OF INDIA

UNIFIED PAYMENTS INTERFACE

Common SDK Specification

Version 2.0

UNIFIED PAYMENTS INTERFACE

Document Version: 1.0.

Document History

Change No.	Changes done	Revision / Version Number	Date of Change	Change Description
1	Initial Document	1.0	27-10-2023	1. Initial draft including CL interfaces for Digital Rupee

Table of Contents

1. Introduction 5
2. Objectives 5
3. Components 5
3.1 Common Library files 5
3.2 Steps to integrate CL sdk file to psp/tsp (or authorized service provider) app 5
4. Specification 6
4.1 Common interface flow for Token System 6
5. Online- Common CL Specification 9
5.1. CL Initialization 9
5.2 Device Registration 10
5.2.1. Get Secret for Registration 10
5.2.2. Perform Rotation for Keys 11
5.3.1. registerApp 12
5.4 ERUPEE Onboarding 13
5.4.1. Checking TEE Supported Device 13
5.4.2. Checking Bound Between APP and CL 14
5.4.3. Create Identity – getCredential(Identity) 15
5.4.4. Register Vault Onboarding 25
5.4.5. getBalance 26
5.5. E-RUPEE KYC 27
5.5.1. Card details capturing 27
5.5.2 OTP FLOW 29
5.5.3. RegisterState 30
5.6. E-RUPEE SET/CHANGE/FORGOT TPIN 31
5.6.1. Card Capturing 31
5.6.2. OTP and NTPIN FLOW 33
5.7. Change TPIN 34
5.8. E-RUPEE UNLOAD/DEREGISTER /UPI-Lite DEREGISTER 36
5.9. E-RUPEE Online Token Pay / Lite Pay 39
6. Offline- Common CL Specification 41
6.1. Offline Transaction API 52
6.1.1. offLineTxnFlow() 52

6.1.2. Get Purse Info 53
6.1.3. Assert Identity 55
6.1.4. Verify Identity 56
6.1.5. Start Debit 57
6.1.6. Start Credit 58
6.1.7. Request Value 59
6.1.8. Transfer Value 60
6.1.9. Acknowledge Value 61
6.1.10. Restart Transfer 61
6.2. Offline Logs Syncing 62
Appendix 65
NPCI list keys xml for the Normal User 65
NPCI list keys xml for Enabling Offline for the User 67
GetLite/GetWallet for Other Services 70
GetLite/GetWallet for Offline Service 70
Purse Error Codes 71
New Common CL file integration 72

UNIFIED PAYMENTS INTERFACE

1. Introduction

NPCI UPI common library (CL) is a specification with default implementation which (authorized service provider or TSP/PSP) PSP can embed into their app to communicate with NPCI central system (PSO) to set/reset/change PIN, balance enquiry, and for debit authorization with two factor secured authentication.

Common library ensures all sensitive data encrypted with latest technology encryption standards.

These libraries will be available for two major mobile operating systems such as Android and iOS. This document explains the configuration and implementation of common library services with UPI app for Android devices and iOS devices.

2. Objectives

- Provide a common interface to interact with NPCI UPI system (PSO) to protect the user credentials.
- Protect user credentials from replay and transfer credential.
- Bind PSP/TSP app with NPCI CL and vice-versa.
- CL Captures the Credentials and protects for transport.
- NPCI UPI server validate and authenticates the user credential.
- Provide higher end security component.
- Provide extensible service offering with minimal changes in system

3. Components

3.1 Common Library files

UNIFIED PAYMENTS INTERFACE

1. Android Security Component (Securecomponent-release-signed_{version}.aar)

3.2 Steps to integrate CL sdk file to psp/tsp (or authorized service provider) app

Verifying signed module

This step is to verify the common library authenticity that it is signed by NPCI
Note: Please make sure that Java Development Kit (JDK), Version 8 is installed in the system and the Path environment variable is having <JDK Installation>/bin folder entry. Jarsigner utility comes bundled with JDK.
Command:

```
jarsigner -verify   <FileName>
```

Sample:

```
jarsigner  -verify Securecomponent-release-signed..aar
```

Expected output:

Jar file Verified.

Warnings can be ignored. After verification add the module by following steps:

1. Go to File>New>New Module
2. Select "Import .JAR/. AAR Package" and click next.
3. Enter the path to .aar/.jar file and click finish.
4. Go to File>Project Structure (Ctrl+Shift+Alt+S).
5. Under "Modules," in left menu, select "app."
6. Go to "Dependencies tab.
7. Click the green "+" in the upper right corner.
8. Select "Module Dependency"
9. Select the new added module (SecureComponent-release-signed.aar) from the list.

4. Specification

Common library provides an interface to mobile applications for both financial & non-financial transactions that can be performed between PSP/TSP/Banks. Integration methodology and flow would differ depending upon the programming platform. Interface details are elaborated below.

Common Library provides the following interfaces such as:

1 Get Secret for Registration : getChallenge(initial)
2 Rotation for Keys: getChallenge(rotate)
3 Register APP (Register Device Binding): registerAPP()
4 Check TEE Support from Device: isDeviceSupported()
5 Get Credential : getCredentail()
6 Validate CL Binding with APP: isBound()
7 Register State: registerState()
8 Get Balance: getBalance()

4.1 Common interface flow for Token System

Registration Process: It is consisting of two sub process: i) Create Identity ii) Register Wallet

A. Create Identity

1. Before registration, the APP calls the SDK's function isDeviceSupported() to check support from device in term of verify the phone supports TEE/SE/SecureEnclave or not, phone is rooted or not, phone supports token service for the user or not. If all the checks are validated successfully, then this API interface return true, otherwise false. Here the inputs of called function is service types such as UPILite, Erupee etc.
2. During registration, the SDK's function Get Secret for Registration: getChallenge() is invoked by the APP.
3. In this process, the common library will register with PSP/TSP mobile application and device.

4. The SDK generates Ko (if newly register app), device key pair, and user key pair.
5. The device key pair 1) signifies device presence, 2) Used for signing captures (pin, token, etc) of a credblock, 3) TSP is required to provide a public key certificate for User public key, 4) Used for signing payment details in offline transactions.
6. User Key Pair: 1) TSP is required to encrypt ARPC with Device public key, 2) Also used for decrypting ARPC.
7. SDK collects device details.
8. SDK forms a json with User public key, Device public key and device details.
9. SDK sends Secret as a JSON object to the APP.
10. Receiving wallet information:
    - Once registered, TSP will send back wallet details to the SDK.
    - SDK will save the information such as wallet model and wallet status.

B. Register Wallet:

This is for the registration of wallet details once the APP receives ARPC from the PSO. In this process, after receiving a certificates, wallet information from PSO, mobile APP invokes reg(Device device) API to stores the certificate, and create the wallet to complete the registration of the App.

1. After receiving the ARPC from PSO, the APP calls the SDK's function registerApp() by passing the device object to complete the registration process by to validating the response, storing the certificate, and creating the wallet.
2. If the validation is successful and then process is completed and it is return true otherwise false.

Balance details:

1. To get the latest Balance from vault for displaying to the user the APP calls the SDK's function getBalance to get the recent balance.
2. To perform the validation before transaction, the APP calls the SDK's function getBalance to get the recent balance and app can put checks based on the balance being returned.
3. While calling the interface, the app passing the input which consist of member data: wallet Address, deviceId.
4. This interface returns the Balance object which consist of array of token details object.

Tokens Loading, Online transactions, Unloading, offline transactions, Get balance, Offline transactions:

1. As per the operations (such as: Loading Tokens, Online Transaction, Unloading Token, Balance, Get Change, Set TPIN, Change TPIN, Forget TPIN, Deregistration Wallet, offline Sync, Offline Transaction) performed by the customer, the APP calls the SDK's getCredential interface to capture the credential block as per selected purpose (operation).
2. The calling of getCredential interface need to pass Credspec object which consists of data members: Controls, payInfo, etc. Here, the Controls specifies the schema for the credential(s) to be captured, which is in JSON Format.
3. In token system, the list of these credentials are to be captured by the APP from.

Controls (As per performed operation)	{ "CredAllowed": [{ "type": "PIN", "subtype": "TPIN", "dType": "NUM", "dLength": "6" }, { "type": "PIN",

	"subtype": "OTP",
	"dType": "NUM",
	"dLength": "6"
	},
	{
	"type": "PIN",
	"subtype": "Recovery codes",
	"dType": "NUM",
	"dLength": "16"
	},
	{
	“type” : “tokens”,
	“subType” : “SIGNATURE”
	“details”: [{ “den”: “10”, qty:”5”}]
	},
	{
	"type": "ARQC",
	"subType": "SIGNATURE",
	"dType": "ALPH",
	"dLength": "2048"
	},
	{
	"type": "LEDGER",
	"subType": "SIGNATURE",
	"dType": "ALPH",
	"dLength": "2048"
	},
	{
	"type":"DEVICE",
	"subtype": "LOGS",
	"dType": "ALPH",
	"dLength":"2048"
	}
	]
	Based on the number of blocks in the JSON, one or more credential input control will be rendered by the SDK.

4. The called interface of SDK returns the Credential as a JSON blocks. It is a signed and encrypted object.
5. The encrypted object which contain JSON data based on type of capture. It Could be pinData, tokenData, arqcData and Its signature.
6. The APP sends Credential data to the PSO.
7. The PSO performs signature verification and extracts the information from Credential.
8. PSO forwards the request to TSP.
9. TSP prepare the response and sends it to the PSO.
10. PSO prepare the ARPC and sends to the APP via TSP.
11. The APP calls SDK’s registerState () interface to process the response of ARPC.
12. According to requested purpose, the ARPC as a state given to SDK.
13. The CL SDK validates/processes the ARPC and stores the state information as per requested purpose.

Unregistered Service:

1. The Merchant/PSP will hit the NPCI with the pay unload CredBlock to Initiate the Un-Load/Deregister token/Deregister UPI-LITE flow, mentioning the flow type(UNLOAD or DEREGISTER).
2. In response to the Unloading/Deregister Token/Deregister UPI-Lite, NPCI will update the State of the wallet and give an updated state.

3. Same needs to be passed to the CL using registerState(..) API to
    a. ERUPEE : remove the stored tokens of the user.
    b. UPI-Lite : deregister the UPI-Lite wallet and delete all UPI-LITE related data.
4. In case of Deregister Token/UPI-Lite call, Merchant/PSP App can use isBound(..) API to check Bind status, which should be NO_BIND if successfully deregistered.
5. In case of ERUPEE Un-Load token merchant/PSP App can use getBalance(..) API to check updated tokens which should be 0.00 in this case after a successful updated.

5. Online- Common CL Specification

5.1. CL Initialization

5.1.1. Service Initialization

Initiated Class : clServices

Method Definition : init()

5.1.1.1. Request

Parameter Name	Data Type	M/O/C	Length	Description
Inputs				
context	Context			It will be the activity instance
notifier	ServiceConnectionStatusNotifier			ServiceConnectionStatusNotifier this is the class in our cl used for callback purposes.
serviceType	String			Based on what service user is going to use (ERUPEE or UPI), the cl will generate the instance of that service, and all functionality will be called on this respective service instance. There can be instances of both types active at a given time. And you can perform operations in both instances of clServices in parallel.

5.1.1.2. Response

Parameter Name	Data Type	M/O/C	Length	Description
Output				
The structure of the message is as follows We will get the instance of clServices (UPI or ERUPEE instance) based on the serviceType you passed on initService and you can use the same instance later to initiate different flows of UPI				

or ERUPEE. Make sure UPI or UPI-Lite flows are triggered over UPI instance of clService and ERUPEE flows are triggered over ERUPEE instance of clServices.

5.2 Device Registration

This steps will be processed at the time of registration which means that the NPCI CL is invoked by the PSP/TSP at the first time. Steps to be followed by PSP/TSP app to register with the common library,

Service Name	Details
Get Secret	Execute "getChallenge" service to receive a challenge from common library.
Get Token	Use generated challenge in PSP/TSP server request to receive a valid token from UPI system using "*Get Token*" service (Same as ListKeys API with specific values in creds block). Received token should be stored in PSP/TSP mobile app to ensure secure communication with common library. Once GetToken is completed, RegisterApp to be executed immediately.
Validate Device	Execute register device binding "registerDeviceBinding" service to verify and register the generated token in the common library.

PSP/TSP mobile app should execute "*Get Challenge*" service to receive a challenge from common library. PSP/TSP does a RPC to CL and asks for "Challenge" which can be decrypted only by NPCI server. The user is already registered with UPI and wants the service of UPI lite and/or Token payment. In this case, during the registration of UPI, the

5.2.1. Get Secret for Registration

Initiated Class : clServices

Method Definition : getChallenge (initial)

5.2.1.1 Request

This method is call to receive a challenge from CL that is used to get token from UPI/PSO. Once GetToken is completed, then Register of App to be executed immediately.

Parameter Name	Data Type	M/O/C	Length	Description
Inputs				
type	String			It can be "initial" or "rotate".
deviceId	String			Device ID unique and stable for

				the mobile device. Role could be fulfilled by UUID (for IOS) or android Id (for android) provided by the platform.

5.2.1.2 Response

Output of the above service will be a string containing the *"Challenge"*. TSP/PSP Mobile app should pass the generated challenge to TSP/PSP server to receive a valid token from UPI. If Common Library fails to generate a challenge, the output will return null. CL will also add version along with challenge.

Sample output: "Service-Type-Version | Challenge"

Example: 2.5.token-v2.0_v1|Challenge

Parameter Name	Data Type	M/O/C	Length	Description
Outputs				
challenge	String			Same as UPI, This challenge will be passed to NPCI.

Note: Customers coming from older versions of CL need to do device registration (getChallange + registerApp) again over UPI instance to continue using OLD UPI flows.

5.2.2. Perform Rotation for Keys

This interface is used to rotate the keys as per defined policies. For example: after receiving the response of ReqListKeys where type="GetToken", key-id="", and subType of cred block is "rotate". Then, this API updates the old token with new token value.

This interface also used for rotating other stored or generated keys as per requirements.

5.2.2.1. Request

Initiated Class : clServices

Method Definition : getChallenge (rotate)

Parameter Name	Data Type	M/O/C	Length	Description
Inputs				

type	String			It can be "initial" or "rotate".
deviceId	String			Device ID unique and stable for the mobile device. Role could be fulfilled by UUID (for IOS) or android Id (for android) provided by the platform.

5.2.2.2. Response

Sample output: "Service-Type-Version | Challenge"

Example: 2.5.token-v2.0_v1|Challenge

Parameter Name	Data Type	M/O/C	Length	Description
Outputs				
challenge	String			Same as UPI, This challenge will be passed to NPCI.

5.3.1. registerApp

UNIFIED PAYMENTS INTERFACE

After receiving a valid token from UPI, PSP/TSP mobile app should store it in the application. After which "Register App (hmac, device -id, app-id, mobile-number)" RPC service should be executed to validate the token and register the PSP/TSP app. This is for the registration of wallet details once the TSP APP receives ARPC from the PSO.

To register back, we can trigger registerApp(appId, mobile, deviceId, hmac, random)

Initiated Class : clServices

Method Definition : registerApp()

5.2.1.1 Request

Parameter Name	Data Type	M/O/C	Length	Description
Inputs				
appId	String			App ID of the particular App
mobile	String			Mobile Number of the user
devideId	String			Device ID unique and stable for the mobile device. Role could be

				fulfilled by UUID (for IOS) or androidId (for android) provided by the platform.
Hmac	String			This is a mandatory parameter. Steps to be followed to generate HMAC. 1. Concatenate device-id, app-id and mobile-number with the separator "|". 2. Create 16-byte random values. 3. Create a hash of the concatenated string using the SHA-256 algorithm and random string created earlier. 4. Encrypt (AES/GCM/NoPadding) the hash with the token as key using AES-256 algorithm. Use random as iv parameter. 5. Populate Hmac with the encrypted string
random	String			This is the Base64 encoded 16byte random string generated by the app. Which is used as random for creating hash and also as iv parameter while AES encrypts the hash for Hmac.

5.2.1.2 Response

Returns a Boolean Value (TRUE/FALSE) to indicate the status of App Registration

Parameter Name	Data Type	M/O/C	Length	Description
Outputs				
isRegisterApp	Boolean			A Boolean value will be generated using the above request representing success (1) or failure (0) of register device binding the device.

5.4 ERUPEE Onboarding

5.4.1. Checking TEE Supported Device

Initiated Class : clServices

Method Definition : isDeviceSupported ()

5.2.1.1 Request

Parameter Name	Data Type	M/O/C	Length	Description
Inputs				

5.2.1.2 Response

Returns a Boolean Value (TRUE/FALSE) to indicate the status of App Registration

Parameter Name	Data Type	M/O/C	Length	Description
Outputs				
isDeviceSupported	Boolean			True if the device has TEE and the device is not rooted, and false otherwise.

5.4.2. Checking Bound Between APP and CL

If the customer device is supported then this isBound() will be triggered

Initiated Class : clServices

Method Definition : isBound ()

5.2.1.1 Request

Parameter Name	Data Type	M/O/C	Length	Description
Inputs				
mobile	String			Mobile number of the user
deviceId	String			Device ID unique and stable for the mobile device. Role could be fulfilled by UUID (for IOS) or androidId (for android) provided by the platform.
referenceId	String			For UPI-Lite : referenceId ==accountRef For ERUPEE: referenceId == eRupeeWalletAddress

5.2.1.2 Response

Returns a Boolean Value (TRUE/FALSE) to indicate the status of App Registration

Parameter Name	Data Type	M/O/C	Length	Description
Outputs				

deviceStatus**	Stringified Json			{
				"status": "BOUND"|"NO_BIND",
				"syncRequired":"true"|"false",
				"kycStatus":
				"NONE"|"LIMITED"|"FULL"
				}

* syncRequired will only be there when status is BOUND
** kycStatus will only exist when serviceType = ERUPEE and status is BOUND

5.4.3. Create Identity – getCredential(Identity)

Onboarding flow of UPI-Lite and ERUPEE is similar for now. If we do onboarding on CLServices Instance of UPI, it will trigger the UPI-LITE onboarding. And if we do onboarding on CLServices Instance of ERUPEE, it will trigger the ERUPEE onboarding.

Initiated Class : clServices

Method Definition : getCredential()

5.4.2.1. Request

Parameter Name	Data Type	M/O/C	Length	Description
Inputs				
keyCode	String			NPCI This is a mandatory parameter specifying NPCI or UIDAI public key is being used. Valid values can be NPCI or UIDAI.
listKeyXmlPayLoad	String			NPCI list keys xml This is a mandatory field containing the digitally signed XML payload received from list-Keys API of UPI. Response of list-keys API not to be modified.
controls	String			{ "CredAllowed": [{ "type":"DEVICE", "subtype":"IDENTITY", "dType": "ALPH", "dLength": "2048" }] }
configuration	String			{ "enableUserAuth":"true/false", “getDeviceDetails”:"true/false", "enableOfflineTxn":"true/false"}
salt	String			{ "txnId": ["MAY96c1d03348df4a8f92ad07019cb0ac39"],

				"deviceId": "ABCDEF", "accountRef"(for UPI-Lite case only): "Ac779771d9c746afad3a7650b23132d", "eRupeeWalletAddress"(for ERUPEE case only): "A1769171r9c716afad1a7650c2398s2", "txnTimestamp": "2022-09-15T14:58:28+05:30", "appId": "com.psp1.app", "mobileNumber": "9190020507", "credType": ["binding"], "random": "<Base64 String>" }
payInfo	String			[]
trust	String			{"binding" : "trust-to-be-validated"}
languagePref	String			en_US
callback	CLRemoteResultReceiver			Instance of the CLRemoteResultReceiver to receive a response.

5.4.2.2 Reponse

Parameter Name	Data Type	M/O/C	Length	Description
Outputs				

CredBlock	String			{
				"IDENTITY": {
				"binding": {
				"type": "DEVICE",
				"subtype": "IDENTITY",
				"data": {
				"code": "NPCI/NPCI-LITE",
				"ki": "20220622",
				"encryptedBase64String": "2.5.token-v2.0_v1/2.5.lite-v2.0_v1|<encrypted base 64encoded authentication data>"
				}
				}
				},
				"det": "2.5-v2.0_v1|<encrypted base 64 encoded data>"
				}

Note: In above case if enableOfflineTxn is set to be "true", CL will add offline EC key and authentication value along with other keys

The response of the above will be a HashMap<String, String> (for Android) with the following attributes. There can be more than one entries in the HashMap, based on the number of credentials being captured by the common library.

The key of HashMap will have the credential name. This will use the name provided in the credentials. This should match with the "name" attribute in the control JSON in the input.

The value will be a JSON string containing the ki (key index) and the encrypted cred block. Example of the values in the HashMap is as per below. The keys of the Hash Map will be the subtype (e.g. MPIN)

Response for MPIN or TPIN cred block

```
{
"MPIN | TPIN ":{
"pay": {
        "type": "PIN",
        "subtype": "MPIN | TPIN",
        "data": {
                "code": "NPCI",
                "ki": "20150822",
                "encryptedBase64String": "service-version|<encrypted base 64
encoded authentication data>"
        ,


 }
},
"det": "service-version|<encrypted base 64 encoded data>"
}
```

Similarly an example of subtype: MPIN & ATMPIN cred block

```
{
"MPIN ":{
"setMpin": {
        "type": "PIN",
        "subtype": "MPIN",
        "data": {
                "code": "NPCI",
                "ki": "20150822",
```

```
                "encryptedBase64String":    ""service-version|<encrypted    base    64
encoded authentication data>"
        } }
},
"ATMPIN":{
    "setMpin":  {
        "type": "PIN",
        "subtype": "ATMPIN",
        "data": {
                "code": "NPCI",
                "ki": "20150822",
                "encryptedBase64String":    "service-version|<encrypted    base    64
encoded authentication data>"
        }}
},
"det": "service-version|<encrypted base 64 encoded data>"
}
```

Similarly an example of subtype: TPIN & RECOVERYCODES cred block

```
{
"TPIN ":{
"setTpin":  {
        "type": "PIN",
        "subtype": "TPIN",
        "data": {
                "code": "NPCI",
                "ki": "20150822",
                "encryptedBase64String":    ""service-version|<encrypted    base    64
encoded authentication data>"
        } }
},
"RECOVERYCODES":{
    " setRevoceryCodes":  {
        "type": "PIN",
        "subtype": "RECOVERYCODES",
        "data": {
                "code": "NPCI",
                "ki": "20150822",
                "encryptedBase64String":    "service-version|<encrypted    base    64
encoded authentication data>"
        }}
},
"det": "sdk-version|<encrypted base 64 encoded data>"
}
```

Similarly an example of subtype: card details cred block

Presently the last 6 digits of the Debit card number and expiry date are captured on the PSP App with base 64 encoding. In CL 1.7 version(as per the standard of UPI CL), we need to move the capturing of card details inside the NPCI Common library to ensure enhanced security .The same card detail cred block will be used in the ReqRegMob API for "FORMAT3|FORMAT2" or "FORMAT3|FORMAT1" and Aadhar consent is not provided in app.

```
{

"CARDDETAILS ":{
  "setMpin": {
        "type": "CARD",
        "subtype": "CARDDETAILS",
        "data": {
                "code": "NPCI","ki": "20150822",
                "encryptedBase64String": "service-version|<encrypted base 64
encoded authentication data>"
        }}
"det": "service-version|<encrypted base 64 encoded data>"
}
```

Response for type: Balance Enquiry

```
{
"MPIN | TPIN":{
"reqbalEnq": {
        "type": "PIN",
        "subtype": "MPIN | TPIN",
        "data": {
                "code": "NPCI","ki": "20150822",
                "encryptedBase64String": "service-version|<encrypted base 64
encoded authentication data>"
        }} ,
"det": "service-version|<encrypted base 64 encoded data>"
}
```

Response for type: Balance Check

```
{
    "reqbalChk": {
        {
        "type": "PIN",
        "subtype": "MPIN | TPIN",
        "data": {
                "code": "NPCI","ki": "20150822",
                "encryptedBase64String": "service-version|<encrypted base 64
encoded authentication data>"
        } }
},
"det": "service-version|<encrypted base 64 encoded data>"
```

```
}
```

Response for type: Pay

```
{
“MPIN | TPIN”: {
"pay": {
        "type": "PIN",
        "subtype": "MPIN | TPIN",
        "data": {
                "code": "NPCI","ki": "20150822",
                "encryptedBase64String": “service-version|<encrypted base 64
encoded authentication data>"
        }} ,
“det”: “service-version|<encrypted base 64 encoded data>"
}
```

Response for type: Collect

```
{
“MPIN | TPIN ”: {
"collect": {
        "type": "PIN",
        "subtype": "MPIN | TPIN”,
        "data": {
                "code": "NPCI",
                serviceType: “”,
                "ki": "20150822",
                "encryptedBase64String": “service-version|<encrypted base 64
encoded authentication data>"
        } },
“det”: “service-version|<encrypted base 64 encoded data>"
}
```

Response for type: mandate

```
{
“MPIN | TPIN”: {
"mandate": {
        "type": "PIN",
        "subtype": "MPIN | TPIN",
        "data": {
                "code": "NPCI",
"ki": "20150822",        "encryptedBase64String": “service-version|<encrypted
base 64 encoded authentication data>"
        } },
“det”: “service-version|<encrypted base 64 encoded data>"
```

```
}
```

Response for type: (PAY, ReqBalChk)| (COLLECT, ReqBalChk)

```
{
"MPIN | TPIN": {
   "pay":  {
         "type": "PIN",
         "subtype": "MPIN | TPIN,
         "data": {
                  "code": "NPCI",
                  "ki": "20150822",
                  "encryptedBase64String":    "service-version|<encrypted    base    64
encoded authentication data>"
         } }
}

"MPIN| TPIN": {
   "reqBalChk":  {
         "type": "PIN",
         "subtype": "MPIN | TPIN",
         "data": {
                  "code": "NPCI",
                  serviceType: "",
                  "ki": "20150822",
                  "encryptedBase64String":    "service-version|<encrypted    base    64
encoded authentication data>"
         } }
},
"det": "service-version|<encrypted base 64 encoded data>"
}
```

Response for Service Onboarding

```
{  "IDENTITY": {

   "binding": {

     "type": "DEVICE",

     "subtype": "IDENTITY",

     "data": {

       "code": "NPCI",
```

```
      "ki": "20150822",
      "encryptedBase64String": "service-version|<encrypted base 64 encoded authentication data>"
    }
  }
 },
 "det": "service-version|<encrypted base 64 encoded data>"
}
```

Response for UPI Lite Load Money

```
{
 "MPIN ": {
  "pay": {
    "type": "PIN",
    "subtype": "MPIN",
    "data": {
      "code": "NPCI",
      "ki": "20150822",
      "encryptedBase64String": "2.3.lite-v1.8|<encrypted base 64 encoded authentication data>"
    }
  },
{
// For converting into digital token
```

```
 "TPIN ": {
   "pay": {
     "type": "PIN",
     "subtype": "TPIN",
     "data": {
       "code": "NPCI",
       "ki": "20150822",
       "encryptedBase64String": "2.3.lite-v1.8|<encrypted base 64 encoded authentication data>"
     }
   }
{
"type" : "tokens",
"subType" : "SIGNATURE"
"details": [{ "den": "10", qty:"5"}]
}
 },
"SIGNATURE": {
   "pay": {
     "type": "ARQC",
     "subtype": "SIGNATURE",
     "data": {
       "code": "NPCI",
       "ki": "20150822",
       "encryptedBase64String": "2.3.lite-v1.8|<encrypted base 64 encoded authentication data>",
     }
   }
```

```
  }
  "det": “service-version|<encrypted base 64 encoded data>"
}
```

Response for Service Transaction

```
{   "SIGNATURE": {
    "pay": {
      "type": "ARQC",
      "subtype": "SIGNATURE",
      "data": {
        "code": "NPCI",
        "ki": "20150822",
        "encryptedBase64String": “2.3.lite-v1.8|<encrypted base 64 encoded authentication data>",
          "oda": “2.3.lite-v1.8|<Base 64 encoded offline data authentication>"
      }
    }
  },
  "det": “service-version|<encrypted base 64 encoded data>"
}
```

In case there is an error in common library, the returned JSON string will be as below:

```
“error”: { “errorCode” : <errorCode>, : “errorText” : <errorText> }
```

For example Digital Signature mismatch in common library, the error will be as below:

```
“error”: { “errorCode” : “L05”, : “errorText” : “Technical Issue, please contact your administrator” }
```

5.4.4. Register Vault Onboarding

In order to complete the Onboarding process, we have to update the state and device certificates to CL.

5.4.4.1. Request

For ERUPEE Onboarding, trigger the **registerVaultOnboarding** function over the ERUPEE instance of CLServices

Parameter Name	Data Type	M/O/C	Length	Description
Inputs				
mobile	String			Customer's Mobile Number
deviceId	String			Device ID unique and stable for the mobile device. Role could be fulfilled by UUID (for IOS) or androidId (for android) provided by the platform.
referenceId	String			For UPI-Lite : referenceId == accountRef For ERUPEE : referenceId == eRupeeWalletAddress
xmlPayLoad	String			For other Services: UPI Lite and Online Digital Rupee xmlPayload of onboarding will have two params, one with code "STATE" (encrypted riskParams) and other with code "DEVICE-CERT" (Device certificate) For Offline: If in last step CL sent publicKeyOffline then xmlPayload of onboarding will should three params, one with code "STATE" (encrypted riskParams) and others with code "DEVICE-CERT" (Device certificate), "DEVICE-CERT-OFFLINE" (offline purse key certificate)

5.4.4.2. Response

Parameter Name	Data Type	M/O/C	Length	Description
Outputs				
registerVautOnbaordingResult	Boolean			Register vault status True / False

Note: After successful onboarding Merchant/PSP can trigger **isBound(..) API** again to check Bind status. If it comes to be true, Merchant/PSP can also check Balance using the **getBalance(..)** API to know balance.

5.4.5. getBalance

This API will let merhant/PSP know the balance of UPI-LITE/ERUPEE For UPI-LITE Balance, trigger the getBalance function over the UPI instance of CLServices. For ERUPEE Balance, trigger the getBalance function over the ERUPEE instance of CLServices.

5.4.5.1. Request

Initiated Class : clServices

Method Definition : getBalance()

Parameter Name	Data Type	M/O/C	Length	Description
Inputs				
mobile	String			Customer's Mobile Number
deviceId	String			Device ID unique and stable for the mobile device. Role could be fulfilled by UUID (for IOS) or androidId (for android) provided by the platform.
referenceId	String			For UPI-Lite : referenceId == accountRef For ERUPEE : referenceId == eRupeeWalletAddress

5.4.5.2. Response

Parameter Name	Data Type	M/O/C	Length	Description
Outputs				

Balance	String			For UPI-LITE it will be a Stringified JSON containing balance and nonSpendableBalance like "{\"balance\":\"50.00\",\"nonSpendable Balance\":\"0.00\",\"pendingLogsCount\ ":\"5\"} For ERUPEE it will be a Stringified JSON which will have token balance and breakup of the tokens like "{\"eRupeeWalletAddress\":\"15medLfL Vse3AsVW9dnz5rChbBVCyYTGgW\", \"retailTokens\" : [{\"serialNumber\":\"220000000048061\ ",\"denomination\": \"100.00\",\"tag\": {\"mcc\": [\"\"],\"tokenType\": \"RCBDC\"}}], \"disputedTokens\" : [{\"serialNumber\":\"220000000048062\ ",\"denomination\": \"100.00\",\"tag\": {\"mcc\": [\"mcc01\",\"mcc02\"],\"tokenType\": \"RCBDC || PCBDC\"}}], \"pbtTokens\": [{\"serialNumber\":\"220000000048063\ ", \"denomination\": \"100.00\",\"tag\": {\"mcc\": [\"mcc01\",\"mcc02\"],\"tokenType\": \"PCBDC\"}}],\"noneSpendableTokens\" : [{\"serialNumber\":\"220000000048064\ ",\"denomination\": \"100.00\",\"tag\": {\"mcc\": [\"mcc01\"],\"tokenType\": \"PCBDC\"}}],\"pendingLogsCount\":\" 5\"}" Here pendingLogsCount has a count of Transaction + Exception Logs

5.5. E-RUPEE KYC

The Merchant / PSP can check the status of a token KYC with **isBound() API** over the clServices instance of ERUPEE. If the KYC is not done, the Merchant / PSP will trigger following getCredential

5.5.1. Card details capturing

5.5.1.1. Request

Initiated Class : clServices

Method Definition : getCredential()

Parameter Name	Data Type	M/O/C	Length	Description
Inputs				
keyCode	String			NPCI This is a mandatory parameter specifying NPCI or UIDAI public key is being used. Valid values can be NPCI or UIDAI.
xmlPayload	String (Stringify JSONArray)			NPCI list keys xml This is a mandatory field containing the digitally signed XML payload received from list-Keys API of UPI. Response of list-keys API not to be modified.
controls	String (Stringify JSON Object)			{"CredAllowed" : [{"type":"CARD", "subtype: "AADHAR/DEBITCARD", "dType":"NUM","dLength":"12/6"}]}
configuration	String (Stringify JSON Object)			{"getDeviceDetails":"true/false"}
salt	String (Stringify JSON Object)			{"txnId":["txnId1"],"deviceId":"ABCD EFG","txnTimeStamp":"2022-09-15T14 :58:28+05:30","eRupeeWalletAddress": "A1769171r9c716afad1a7650c2398s2", "appId":"com.psp1.app","mobileNumbe r":"919002050725","credType":["kyc"], "random":"<Base64 String>"}
payInfo	String			[]
trust	String			{"kyc" : "trust-to-be-validated"}
languagePref	String			en_US
callback	CLRemoteResu ltReceiver			Instance of CLRemoteResultReceiver to receive a response.

5.5.1.2. Response

Parameter Name	Data Type	M/O/C	Length	Description
Outputs				
CredBlock	JSON object			{"AADHAR/DEBITCARD": {"kyc": {"type": "CARD","subtype": "AADHAR/DEBITCARD", "data": {"code": "NPCI","ki":

				"20220322", "encryptedBase64String": "2.5.token-v2.0_v1|<encrypted base 64 encoded authentication data>"}}}, "det": "2.5-v2.0_v1|<encrypted base 64 encoded data>"}

PSP will hit NPCI with this credBlock and in response NPCI will confirm and then PSP/ Merchant will trigger the OTP flow.

5.5.2 OTP FLOW

5.5.2.1. Request

Initiated Class : clServices

Method Definition : getCredential()

Parameter Name	Data Type	M/O/C	Length	Description
Inputs				
keyCode	String			NPCI This is a mandatory parameter specifying NPCI or UIDAI public key is being used. Valid values can be NPCI or UIDAI.
xmlPayload	String (Stringify JSONArray)			NPCI list keys xml This is a mandatory field containing the digitally signed XML payload received from list-Keys API of UPI. Response of list-keys API not to be modified.
controls	String (Stringify JSON Object)			{"CredAllowed" : [{"type":"CARD", "subtype: "AADHAR/DEBITCARD", "dType":"NUM","dLength":"12/6"}]}
configuration	String (Stringify JSON Object)			{"getDeviceDetails":"true/false"}
salt	String (Stringify JSON Object)			{"txnId":["txnId1"],"deviceId":"ABCDEFG", "txnTimeStamp":"2022-09-15T14 :58:28+05:30","eRupeeWalletAddress": "A1769171r9c716afad1a7650c2398s2", "appId":"com.psp1.app","mobileNumbe

				r":"919002050725","credType":["kyc"], "random":"<Base64 String>"}
payInfo	String			[]
trust	String			{"kyc" : "trust-to-be-validated"}
languagePref	String			en_US
callback	CLRemoteResultReceiver			Instance of CLRemoteResultReceiver to receive a response.

5.5.2.2. Response

Parameter Name	Data Type	M/O/C	Length	Description
Outputs				
CredBlock	JSON object			{"OTP": {"kyc": {"type":"PIN","subtype": "OTP","data": {"code": "NPCI","ki": "20220322", "encryptedBase64String": "2.5.token-v2.0_v1|<encrypted base 64 encoded authentication data>"}}}, "det": "2.5-v2.0_v1|<encrypted base 64 encoded data>"}

Merchant will again hit NPCI with this credBlock for validation purposes. In return NPCI will give an updated state **(newState)** which app will pass to CL in the following API over the ERUPEE instance.

registerState(mobile, deviceId, referenceId, newState) -> Boolean

5.5.3. RegisterState

In order to complete the Any UPI-LITE/ERUPEE flow, we have to update the state to CL. For UPI-LITE Onboarding, trigger the registerState function over the UPI instance of CLServices. For ERUPEE Onboarding, trigger the registerState function over the ERUPEE instance of CLServices.

5.5.3.1. Request

Initiated Class : clServices

Method Definition : registerState()

Parameter Name	Data Type	M/O/C	Length	Description
Inputs				
mobile	String			Customer's Mobile Number
deviceId	String			Device ID unique and stable for the mobile device. Role could be fulfilled by UUID (for IOS) or androidId (for android) provided by the platform.
referenceId	String			For UPI-Lite : referenceId == accountRef For ERUPEE : referenceId == eRupeeWalletAddress
newState	String			newState

5.5.3.2. Response

Parameter Name	Data Type	M/O/C	Length	Description
Outputs				
registerStateResult	Boolean			Register vault status True / False

5.6. E-RUPEE SET/CHANGE/FORGOT TPIN

This is a two step process where in first step Merchant/PSP app will open CL to get the Card (Aadhar/Debit) details which then pass to NPCI and in second step Merchant/PSP app will open CL to get OTP and customer's desired TPIN which again passed to NPCI for verification and setting.

5.6.1. Card Capturing

5.6.1.1. Request

Initiated Class : clServices

Method Definition : getCredential()

Parameter Name	Data Type	M/O/C	Length	Description
Inputs				
keyCode	String			NPCI This is a mandatory parameter specifying NPCI or UIDAI public key is being used. Valid values can be NPCI or UIDAI.

xmlPayload	String (Stringify JSONArray)			NPCI list keys xml This is a mandatory field containing the digitally signed XML payload received from list-Keys API of UPI. Response of list-keys API not to be modified.
controls	String (Stringify JSON Object)			{"CredAllowed" : [{"type":"CARD", "subtype: "AADHAR/DEBITCARD", "dType":"NUM","dLength":"12/6"}]}
configuration	String (Stringify JSON Object)			{"getDeviceDetails":"true/false"}
salt	String (Stringify JSON Object)			{"txnId":["txnId1"],"deviceId":"ABCD EFG","txnTimeStamp":"2022-09-15T14 :58:28+05:30","eRupeeWalletAddress": "A1769171r9c716afad1a7650c2398s2", "appId":"com.psp1.app","mobileNumbe r":"919002050725","credType":["setTpin"], "random":"<Base64 String>"}
payInfo	String			[]
trust	String			{"setTpin" : "trust-to-be-validated"}
languagePref	String			en_US
callback	CLRemoteResu ltReceiver			Instance of CLRemoteResultReceiver to receive a response.

5.6.1.2. Response

UNIFIED PAYMENTS INTERFACE

Parameter Name	Data Type	M/O/C	Length	Description
Outputs				
CredBlock	JSON object			{"AADHAR/DEBITCARD": {"setTpin": {"type": "CARD","subtype": "AADHAR/DEBITCARD", "data": {"code": "NPCI","ki": "20220322", "encryptedBase64String": "2.5.token-v2.0_v1|<encrypted base 64 encoded authentication data>"}}}, "det": "2.5-v2.0_v1|<encrypted base 64 encoded data>"}

Above CredBlock needs to be passed to NPCI, in response to which NPCI will give wallet details and a confirmation for trigger OTP and open CL for OTP and TPIN capturing. Merchant/PSP App will trigger send otp flow and then open CL for OTP and new TPIN capturing. In case customers don't have an Aadhar/Debit Card. They

can use recovery code also to set TPIN. Following is the request and response for capturing Recovery code.

5.6.2. OTP and NTPIN FLOW

5.6.2.1. Request

Initiated Class : clServices

Method Definition : getCredential()

Parameter Name	Data Type	M/O/C	Length	Description
Inputs				
keyCode	String			NPCI This is a mandatory parameter specifying NPCI or UIDAI public key is being used. Valid values can be NPCI or UIDAI.
xmlPayload	String (Stringify JSONArray)			NPCI list keys xml This is a mandatory field containing the digitally signed XML payload received from list-Keys API of UPI. Response of list-keys API not to be modified.
controls	String (Stringify JSON Object)			{"CredAllowed":[{"type":"PIN","subtype":"OTP","dType":"NUM","dLength":"6"},{"type":"PIN","subtype":"NTPIN","dType":"NUM","dLength":"6"}]}
configuration	String (Stringify JSON Object)			{"getDeviceDetails":"true/false"}
salt	String (Stringify JSON Object)			{"txnId":["txnId1"],"deviceId":"ABCDEFG","txnTimeStamp":"2022-09-15T14:58:28+05:30","eRupeeWalletAddress":"A1769171r9c716afad1a7650c2398s2","appId":"com.psp1.app","mobileNumber":"919002050725","credType":["kyc"],"random":"<Base64 String>"}
payInfo	String			[]
trust	String			{"kyc" : "trust-to-be-validated"}
languagePref	String			en_US
callback	CLRemoteResultReceiver			Instance of CLRemoteResultReceiver to receive a response.

5.6.1.2. Response

Parameter Name	Data Type	M/O/C	Length	Description
Outputs				
CredBlock	JSON object			{"OTP": {"setTpin": {"type": "PIN","subtype": "OTP", "data": {"code": "NPCI","ki": "20220322", "encryptedBase64String": "2.5.token-v2.0_v1|<encrypted base 64 encoded authentication data>"}}}, "NTPIN": {"setTpin": {"type": "PIN","subtype": "NTPIN", "data": {"code": "NPCI","ki": "20220322", "encryptedBase64String": "2.5.token-v2.0_v1|<encrypted base 64 encoded authentication data>"}}}, "det": "2.5-v2.0_v1|<encrypted base 64 encoded data>"}

Above credBlock needs to be passed to the NPCI for validation and setting TPIN, in response of which NPCI will return the status and CL-State. Which then the Merchant/PSP app has to pass to the CL using **registerState(..) API** over ERUPEE clServices instance.

5.7. Change TPIN

For the Customer who wants to change their TPIN and remember their old TPIN, the Merchant/PSP App can initiate this Change TPIN Flow. This is a one step flow where on CL the Customer has to put their old tpin and new tpin and NPCI will validate their oldTpin and update the new TPIN

5.7.1. Request

Initiated Class : clServices

Method Definition : getCredential()

Parameter Name	Data Type	M/O/C	Length	Description
Inputs				
keyCode	String			NPCI This is a mandatory parameter specifying NPCI or UIDAI public key is being used. Valid values can be NPCI or UIDAI.

xmlPayload	String (Stringify JSONArray)			NPCI list keys xml This is a mandatory field containing the digitally signed XML payload received from list-Keys API of UPI. Response of list-keys API not to be modified.
controls	String (Stringify JSON Object)			{"CredAllowed" : [{"type":"PIN", "subtype:"TPIN", "dType":"NUM", "dLength":"6"}, {"type":"PIN", "subtype:"NTPIN", "dType":"NUM", "dLength":"6"}]}
configuration	String (Stringify JSON Object)			{"getDeviceDetails":"true/false"}
salt	String (Stringify JSON Object)			{"txnId":["txnId1"],"deviceId":"ABCD EFG","txnTimeStamp":"2022-09-15T14 :58:28+05:30","eRupeeWalletAddress": "A1769171r9c716afad1a7650c2398s2", "appId":"com.psp1.app","mobileNumbe r":"919002050725","credType":["changeTpin"], "random":"<Base64 String>"}
payInfo	String			[]
trust	String			{"changeTpin" : "trust-to-be-validated"}
languagePref	String			en_US
callback	CLRemoteResu ltReceiver			Instance of CLRemoteResultReceiver to receive a response.

UNIFIED PAYMENTS INTERFACE

5.7.2. Response

Parameter Name	Data Type	M/O/C	Length	Description
Outputs				
CredBlock	JSON object			{"TPIN": {"changeTpin": {"type": "PIN","subtype": "TPIN", "data": {"code": "NPCI","ki": "20220322", "encryptedBase64String": "2.5.token-v2.0_v1|<encrypted base 64 encoded authentication data>"}}},"NTPIN": {"changeTpin": {"type": "PIN","subtype": "NTPIN", "data": {"code": "NPCI","ki": "20220322", "encryptedBase64String": "2.5.token-v2.0_v1|<encrypted base 64 encoded authentication data>"}}}, "det":

				"2.5-v2.0_v1|<encrypted base 64 encoded data>"}

Above CredBlock needs to be passed to NPCI, in response of which NPCI will give CL State, which then Merchant/PSP suppose to pass to CL using **registerState(..)** API over ERUPEE clServices instance.

5.8. E-RUPEE UNLOAD/DEREGISTER /UPI-Lite DEREGISTER

- If the Customer wishes to Unload all of its tokens or deregister from ERUPEE then the Merchant/PSP App has to trigger this flow with txnAmount == Balance Amount and useTokens == Balance breakup and payeeAddr == payerAddr
- If the customer wishes to deregister from UPI-Lite then the Merchant/PSP App has to trigger this flow with txnAmount == Balance and payeeAddr == payerAddr

5.8.1. Request

Initiated Class : clServices

Method Definition : getCredential()

Parameter Name	Data Type	M/O/C	Length	Description
Inputs				
keyCode	String			NPCI This is a mandatory parameter specifying NPCI or UIDAI public key is being used. Valid values can be NPCI or UIDAI.
xmlPayload	String (Stringify JSONArray)			NPCI list keys xml This is a mandatory field containing the digitally signed XML payload received from list-Keys API of UPI. Response of list-keys API not to be modified.
controls	String (Stringify JSON Object)			For UPI-LITE : "CredAllowed": [{"type": "ARQC", "subtype": "SIGNATURE", "dType": "ALPH","dLength": "2048" }]} For ERUPEE : "CredAllowed":[{"type":"PIN", "subtype": "TPIN", "dType": "ALPH","dLength": "6"

				},{"type": "ARQC", "subtype": "SIGNATURE", "dType": "ALPH","dLength": "2048" }]}
configuration	String (Stringify JSON Object)			{"getDeviceDetails":"true/false"}
salt	String (Stringify JSON Object)			UPI-Lite : {"txnId":["txnId1"], "txnAmount":"2000.00","deviceId":"ABCDEFG","txnTimeStamp":"2022-09-15T14:58:28+05:30","payerLiteAccNumber":"A1769171r9c716afad1a7650c2398s2","appId":"com.psp1.app","mobileNumber":"919002050725","credType":["pay"], "payerAddr":"abc@psp", "payeeAddr":"abc@psp", "random":"<Base64 String>"} For E-RUPEE {"txnId":["txnId1"], "txnAmount":"2000.00","deviceId":"ABCDEFG","txnTimeStamp":"2022-09-15T14:58:28+05:30","eRupeeWalletAddress":"A1769171r9c716afad1a7650c2398s2","appId":"com.psp1.app","mobileNumber":"919002050725","credType":["pay"], "payerAddr":"abc@psp", "payeeAddr":"abc@psp", "tokenDetails" : {"loadTokens": [], "useTokens": [{"den":"200.00","qty":"10"}]}, "random":"<Base64 String>"} Note : for Unload case txnAmount should be >0.00 and <= wallet balance while for Deregister txn Amount should be equals to balance and balance can be 0.00. Also, for both the cases payerAddr should be equals to payeeAddr which is Customer's vpa. Also, for the Unload/Deregister case loadTokens array should be empty
payInfo	String			[]
trust	String			{"pay" : "trust-to-be-validated"}
languagePref	String			en_US
callback	CLRemoteResultReceiver			Instance of CLRemoteResultReceiver to receive a response.

5.8.2. Response

Parameter Name	Data Type	M/O/C	Length	Description
Outputs				
CredBlock	JSON object			For UPI-LITE : {"SIGNATURE": {"pay": {"type": "ARQC","subtype": "SIGNATURE", "data": {"code": "NPCI-LITE","ki": "20220322", "encryptedBase64String": "2.5.lite-v2.0_v1|<encrypted base 64 encoded authentication data>"}}}, "det": "2.5-v2.0_v1|<encrypted base 64 encoded data>"} For ERUPEE : {"SIGNATURE": {"pay": {"type": "ARQC","subtype": "SIGNATURE", "data": {"code": "NPCI-LITE","ki": "20220322", "encryptedBase64String": "2.5.token-v1.9_v1|<encrypted base 64 encoded authentication data>"}}}, "TPIN": {"pay": {"type": "PIN","subtype": "TPIN", "data": {"code": "NPCI","ki": "20220322", "encryptedBase64String": "2.5.token-v2.0_v1|<encrypted base 64 encoded authentication data>"}}}, "det": "2.5-v2.0_v1|<encrypted base 64 encoded data>"}

Steps:

- The Merchant/PSP will hit the NPCI with the above CredBlock to Initiate the Un-Load/Deregister token/Deregister UPI-LITE flow, mentioning the flow type(UNLOAD or DEREGISTER).
- In response to the Unloading/Deregister Token/Deregister UPI-Lite, NPCI will update the State of

the wallet and give an updated state.

- Same needs to be passed to the CL using registerState(..) API to
  - ERUPEE : remove the stored tokens of the user.
  - UPI-Lite : deregister the UPI-Lite wallet and delete all UPI-LITE related data.

- In case of Deregister Token/UPI-Lite call, Merchant/PSP App can use isBound(..) API to check Bind status, which should be NO_BIND if successfully deregistered.
- In case of ERUPEE Un-Load token merchant/PSP App can use getBalance(..) API to check updated tokens which should be 0.00 in this case after a successful update

5.9. E-RUPEE Online Token Pay / Lite Pay

Merchant/PSP App can check balance first before proceeding to txn using getBalance(..) API.
If Wallet has enough Balance then Merchant/PSP App can proceed to lite/token pay.

5.9.1. Request

Initiated Class : clServices

Method Definition : getCredential()

Parameter Name	Data Type	M/O/C	Length	Description
Inputs				
keyCode	String			NPCI This is a mandatory parameter specifying NPCI or UIDAI public key is being used. Valid values can be NPCI or UIDAI.
xmlPayload	String (Stringify JSONArray)			NPCI list keys xml This is a mandatory field containing the digitally signed XML payload received from list-Keys API of UPI. Response of list-keys API not to be modified.
controls	String (Stringify JSON Object)			"CredAllowed": [{"type": "ARQC", "subtype": "SIGNATURE", "dType": "ALPH","dLength": "2048" }]}
configuration	String (Stringify JSON Object)			{"getDeviceDetails":"true/false"}
salt	String (Stringify JSON Object)			**For UPI-Lite :** {"txnId":["txnId1"], "txnAmount":"200.00","deviceId":"ABCDEFG","txnTimeStamp":"2022-09-15T14:58:28+05:30","payerLiteAccNumber":"A1769171r9c716afad1a7650c2398s2","appId":"com.psp1.app","mobileNumber":"919002050725","credType":["pay"], "payerAddr":"abc@psp", "payeeAddr":"def@psp2", "random":"<Base64 String>"}

				For ERUPEE {“txnId”:[“txnId1”], ”txnAmount”:”220.00”,”deviceId”:”ABCDEFG”,”txnTimeStamp”:”2022-09-15T14:58:28+05:30”,“eRupeeWalletAddress”:”A1769171r9c716afad1a7650c2398s2”,”appId”:”com.psp1.app”,”mobileNumber”:”919002050725”,”credType”:[”pay”], “payerAddr”:”abc@psp”, “payeeAddr”:”def@psp2”, "tokenDetails" : { "loadTokens": [{"den":"100.00","qty":"3"}{"den":"50.00","qty":"1",], "useTokens": ["den":"200.00","qty":"1","tag":{"mcc": ["mcc01"],"tokenType":"PCBDC"}},{"den":"20.00","qty":"1","tag":{"mcc":[],"tokenType":"RCBDC"}}]}, “random”:”<Base64 String>”}
payInfo	String			[]
trust	String			{“pay” : “trust-to-be-validated”}
languagePref	String			en_US
callback	CLRemoteResultReceiver			Instance of CLRemoteResultReceiver to receive a response.

UNIFIED PAYMENTS INTERFACE

5.9.2. Response

Parameter Name	Data Type	M/O/C	Length	Description
Outputs				
CredBlock	JSON object			{"SIGNATURE": {"pay": {"type": "ARQC","subtype": "SIGNATURE", "data": {"code": “NPCI-LITE","ki": "20220322", "encryptedBase64String": "2.5.token-v2.0_v1|<encrypted base 64 encoded authentication data>"}}}, "det": "2.5-v2.0_v1|<encrypted base 64 encoded data>"}

6. Offline- Common CL Specification

After successful Onboarding in order to perform Offline txn, there is a new interface called offlineTxnFlow which is used to perform the offline transaction.

The flow of interaction between User Agents and Purses during a value transfer. In this flow the payer sends value to the payee, with the payer User Agent acting as the coordinator and the payee User Agent acting as the responder. The various interactions occur in five stages – initiation, mutual authentication, first value transfer, optional second value transfer, and finish.

A "purse" (wallet) is a logical store of value. Each purse has its own globally unique identifier. To ensure security, purse functionality and management of on-device data storage is located within the Common Library. A Common Library may hold one Digital Rupee purse (Wallet).

A "User Agent" is the logical entity that orchestrates the transfer of value from a local purse to a remote purse, which it does by communicating with the User Agent on the remote device. User Agents are peer entities, and can support value transfers in either direction, but one User Agent shall coordinate the transfer, as shown in Figure 1. The User Agent is located within a Mobile App that manages the user experience and agent-to-agent communications.

A "device" is the entity that hosts purses and User Agents, such as a mobile phone or point of sale terminal.

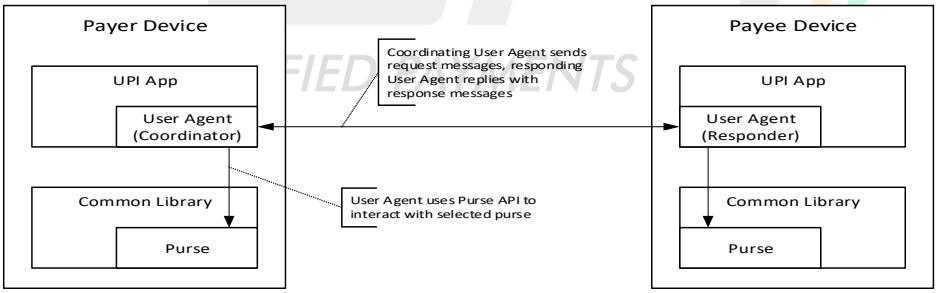
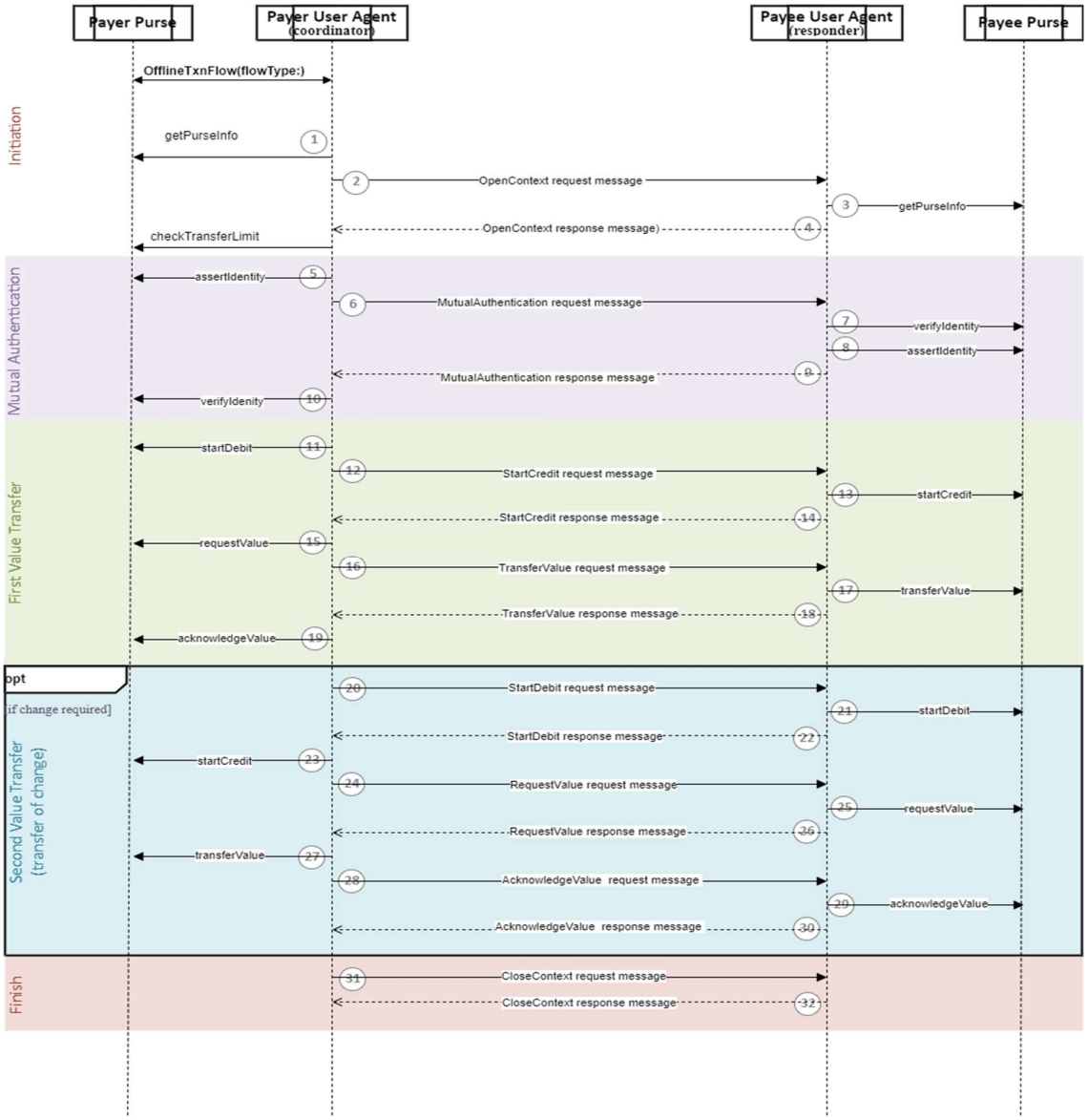


Figure 1: Logical architecture – payer and payee roles

The offline transaction as shown in below Figure.



The following table provides a step-by-step overview of the flow illustrated in Figure 2., where the payer User Agent coordinates the sending of value from the payer to the payee:

#	Description
	The payer uses the Mobile App to select the amount to send and which purse to use (the amount and means of payment could be prepopulated by, say, scanning a payment QR code provided by the payee). The Mobile App shall prompt the payer for CDCVM entry (passcode or biometric) as required.
1	The payer User Agent calls offlineTxnFlow() with flowType getPurseInfo on the purse to retrieve its purse information, including its identifier, sequence number, state, security challenge and balance (and, if Digital Rupees, details on the mix of tokens in the purse, including the mix of those tokens available for spending).
2	The payer User Agent takes the role of coordinator and sends an OpenContext request message to the payee User Agent. The OpenContext request message passes payer purse info to the payee User Agent, along with an indicator that the payer User Agent is the coordinator.

	Note, the coordinator can choose whether to reveal part or all of its available balance / token mix to the payee User Agent. Note, if the intent is to load or withdraw Digital Rupees with an online account provider, the coordinator will should the total token balance and mix of all tokens in the purse to the payee User Agent, rather than the available token balance and mix of available tokens. The OpenContext request message may also contain optional payee_info details for the payee Mobile App, such as QR code data scanned by the payer.
3	On receipt of the OpenContext request message, the payee User Agent takes the role of responder. The payee User Agent selects the relevant purse (based on the given purse product type, currency and country) and calls its offlineTxnFlow() with flowType getPursetInfo to retrieve its purse information, including its identifier, sequence number, state, security challenge and balance (and, if Digital Rupees, details on the tokens in the purse). Optional payee_info details, if present, are passed to the payee Mobile App to determine if okay to proceed.
4	If okay to proceed, the responder sends an OpenContext response message to the coordinator. The OpenContext response message conveys payee purse info to the coordinator. Note, the responder can choose whether to reveal part or all of its available balance / token mix to the coordinator. Note, if the coordinator is an online Digital Rupee account provider, the responder should send the total token balance and mix of all tokens in the purse to the coordinator, rather than the available token balance and mix of available tokens.
	On receipt of the OpenContext response message, the coordinator determines if the transfer can proceed and, if Digital Rupees, which mix of tokens to use. Note, if the chosen mix of tokens requires change, then two transfer stages will be required: first value transfer followed by second value transfer. Otherwise, only first value transfer is required.
	The Mobile App may check that it is okay for an offline value transfer to proceed: it can do this by invoking offlineTxnFlow() with flowType checkTransferLimits and comparing the given Purse Verification Results against Purse Action Codes to determine if an online transfer should be attempted instead. If it does not suggest to proceed offline payer SDK will invoke txn online if online connectivity is available by passing payee QR.
5	The coordinator calls offlineTxnFlow() with flowType checkTransferLimits and calculate Purse Verification Results at payee side. The coordinator calls offlineTxnFlow() with flowType assertIdentity, passing the challenge value received in the OpenContext response message. The purse returns its signed authentication data object, including its id and public key certificates.
6	The coordinator sends a MutualAuthentication request message to the responder. The MutualAuthentication request message contains the payer purse signed authentication data.
7	On receipt of the MutualAuthentication request message, the responder passes the received authentication data to offlineTxnFlow() with flowType verifyIdentity to confirm its validity.
8	The responder calls offlineTxnFlow() with flowType assertIdentity, passing the challenge value received in the OpenContext request message. The purse returns its signed authentication data object, including its id and public key certificates.
9	The responder sends a MutualAuthentication response message to the coordinator. The MutualAuthentication response message contains the payee purse signed authentication data.
10	The coordinator calls offlineTxnFlow() with flowType verifyIdentity, passing the authentication data received in the MutualAuthentication request message to confirm its validity. If the call is successful, then the coordinator knows that the two purses are successfully mutually authenticated.

	Note, mutual authentication guards against man-in-the-middle, replay, spoofing and impersonation attacks.
11	The coordinator calls offlineTxnFlow() with flowType startDebit to begin first value transfer, passing a startTransferData object that defines the payer and payee purse ids and sequence numbers, the current date/time, the amount and, if Digital Rupee, the mix of tokens to transfer. The payer purse verifies that it can pay the requested amount and sets its state to AWAITING_REQ in readiness for a later a offlineTxnFlow() with flowType requestValue call. Note, value is not debited at this step.
12	The coordinator sends a StartCredit request message to the responder. The StartCredit request message contains a startTransferData object that has the exact same values as passed to the payer purse (in step 11 above).
13	On receipt of the StartCredit request message, the responder calls offlineTxnFlow() with flowType startCredit, passing the given startTransferData object. The payee purse verifies that it can accept the amount and, if yes: • Sets its state to AWAITING_VAL in readiness for a later transferValue call, • Computes a request signature over the startTransferData. • Returns a requestValueData object containing the request signature.
14	The responder sends a StartCredit response message to the coordinator. The StartCredit response message contains the payee signature in the requestValueData.

15	The coordinator calls offlineTxnFlow() with flowType requestValue, passing the requestValueData received in the StartCredit response message.
	The payer purse checks the request signature in the received requestValueData is valid then:
	• Updates its state to AWAITING_ACK,
	• Creates a transferValueData object,
	• For Digital Rupees:
	  ○ Removes tokens from its Purse Token Store in the same number and denominations as the request token mix,
	  ○ Adds the tokens to the transferValueData,
	• Computes a transfer signature over the startTransferData, request signature and tokens (if present),
	• Adds the transfer signature to the transferValueData,
	• Returns the transferValueData.
	Note, value is debited from the payer purse at this step.

16	The coordinator sends a TransferValue request message to the responder. The TransferValue request message contains the transferValueData.
17	On receipt of the TransferValue request message, the responder calls offlineTxnFlow() with flowType transferValue, passing the given transferValueData (which may contain Digital Rupee token(s)). The payee purse checks the transfer signature and tokens, if present, in the received transferValueData are valid then: • For Digital Rupees: ○ Adds the tokens to its Purse Token Store. • Computes an acknowledgement signature over the startTransferData, request signature, tokens (if present) and transfer signature, • Records the transfer as complete (including the startTransferData, request signature, tokens (if present), transfer signature and acknowledgement signature) in its Transaction Log, • Updates its state to IDLE, • Returns an acknowledgeValueData object containing the acknowledgement signature.

18	The responder sends a TransferValue response message to the coordinator. The TransferValue response message contains the acknowledgeValueData.
19	The coordinator calls offlineTxnFlow() with flowType acknowledgeValue, passing the acknowledgeValueData received in the TransferValue response message. The payer purse checks the acknowledgement signature in the received acknowledgeValueData is valid then: • Records the transfer as complete (including the startTransferData, request signature, tokens (if present), transfer signature and acknowledgement signature) in its Transaction Log, • Updates its state to IDLE.
	If the coordinator determined after step 4 that a transfer of change is not required, then it can proceed to step 31 to finish the transfer. Otherwise, if the transfer of change is required, it can continue to step 20 to begin the second value transfer. Note, the payer User Agent will remain coordinator during second value transfer even though value will be transferred from the payee purse to the payer purse.
20	The coordinator prepares a new startTransferData for the transfer of change, setting the payerId and payerSeq to the payee purse id and sequence number plus one, and setting the payeeId and payeeSeq to the payer purse id and sequence number plus one. The coordinator sends a StartDebit request message to the responder. The StartDebit request message contains the startTransferData object that defines the change amount and mix of tokens to transfer.
	Note, on receipt of a StartDebit request message, if the payee is not currently authenticated on their device, the responder User Agent may optionally request input of CDCVM from the payee before continuing.
21	The responder calls startDebit() to begin second value transfer, passing the given startTransferData, which specifies the amount and the mix of tokens to transfer. The purse verifies that it can pay the requested amount and sets its state to AWAITING_REQ in readiness for a later a offlineTxnFlow() with flowType requestValuecall.
22	The responder sends a StartDebit response message to the coordinator.
23	The coordinator calls offlineTxnFlow() with flowType startCredit, passing startTransferData, which specifies the amount and the mix of tokens to transfer. The purse verifies that it can accept the amount and, if yes: • Sets its state to AWAITING_VAL in readiness for a later transferValue() call, • Computes a request signature over the over startTransferData, • Returns a requestValueData object containing the request signature.
24	The coordinator sends a RequestValue request message to the responder. The RequestValue request message contains the requestValueData.
25	On receipt of the RequestValue request message, the responder calls offlineTxnFlow() with flowType requestValue, passing the requestValueData. The purse checks the request signature in the received requestValueData is valid then: • Updates its state to AWAITING_ACK, • Creates a transferValueData object, • Removes tokens from its Purse Token Store in the same number and denominations as the requested token mix, • Adds the tokens to the transferValueData, • Computes a transfer signature over the startTransferData, request signature and tokens, • Adds the transfer signature to the transferValueData, • Returns the transferValueData. Note, value is debited from the payee purse at this step.

26	The responder sends a RequestValue response message to the coordinator. The RequestValue response message contains the transferValueData.
27	The coordinator calls offlineTxnFlow() with flowType transferValue, passing the given transferValueData (which contains the token(s)). The purse checks the transfer signature and tokens in the received transferValueData are valid then: • Adds the tokens to its Purse Token Store, • Computes an acknowledgement signature over the startTransferData, request signature, tokens and transfer signature, • Records the transfer as complete (including the startTransferData, request signature, tokens, transfer signature and acknowledgement signature) in its Transaction Log, • Updates its state to IDLE, • Returns an acknowledgeValueData object containing the acknowledgement signature.
28	The coordinator sends an AcknowledgeValue request message to the responder. The AcknowledgeValue request message contains the acknowledgeValueData.
29	The responder calls offlineTxnFlow() with flowType acknowledgeValue, passing the acknowledgeValueData received in the AcknowledgeValue request message. The payee purse checks the acknowledgment signature in the received acknowledgeValueData is valid then: • Records the transfer as complete (including the startTransferData, request signature, tokens, transfer signature and acknowledgement signature) in its Transaction Log, • Updates its state to IDLE.
30	The responder sends an AcknowledgeValue response message to the coordinator.
31	The coordinator sends a CloseContext request message to the responder.
32	The responder sends a CloseContext response message to the coordinator. The coordinator finishes the transfer by closing the communications channel with the responder.

UNIFIED PAYMENTS INTERFACE

CL provides OfflineTxnFlow interfaces with various flow types to support the offline transaction. To perform offline transaction, CL supports OfflineTxnFlow interfaces with the various flow types from checking the pre-condition of transaction, mutual authentication, transfer, re-transfer. The OfflineTxnFlow interface provides various flow to complete the offline trasnaction in device-to-device such as getPurseInfo, assertIdentity, verifyIdentity, startDebit, startCredit, requestValue, transferValue, acknowledgeValue, closeContext, restartTransfer.

#	Interface Definition	Description
1	OfflineTxnFlow() **Request Sample:** { "mobile": "930303033", // mobile number of user "deviceId": "", // UUID (for IOS) or androidId (for android) "referenceId": "", // accountRef(for upi lite) or eRupeeWalletAddress "flowType": "getPurseInfo | assertIdentity | | verifyIdentity | startDebit | startCredit| requestValue | transferValue | acknowledgeValue| closeContext| restartTransfer"	This method is used by APP to perform the offline transaction. This method provide flow types to call the interface to perform the offline transaction

	"requestData": " StringifiedJson", // Value of requestData in case flowType
	}
	
	Response Sample:
	{
	"status":"Status_Codes",
	"flowType": "getPurseInfo | assertIdentity |
	| verifyIdentity | startDebit | startCredit|
	requestValue | transferValue |
	acknowledgeValue| closeContext|
	restartTransfer",
	"responseData": "StringifiedJson"
	}	

Flow Types		

2	getPurseInfo
	
	Request Parameter
	
	{
	"mobile": "930303033",
	"deviceId": "",
	"referenceId": "",
	"flowType": "getPurseInfo"
	"requestData":{}
	}
	}
	
	Response Parameters
	
	{
	"status":"Status_Codes: Ref Section 2.12",
	"flowType": "getPurseInfo",
	"responseData":{
	"id": "1234567890",
	"seq": 123,
	"state": "IDLE",
	"product": "Digital Rupee",
	"currency": "INR",
	"country": "0356",
	"type": "consumer",
	"mcc": null,
	"balance": "85.00", -- For UPI-LITE
	"tokenMix": [-- For ERUPEE
	{"qty":"3","den": "20.00"},
	{"qty": "5","den": "5.00"}],
	"availableBalance": "7000.00", -- For ERUPEE
	"availableTokenMix": [
	{"qty": "3","den": "20.00"},
	{"qty": "2","den": "5.00"}],
	"challenge": "1357924680"
	}
	}
		Flow type: getPurseInfo is invoked to retrieve purse details during transfer initiation. The information provided is included in OpenContext request/response messages by the User Agent.

3	assertIdentity Request Parameter { "mobile": "930303033",	Flow type: assertIdentity(challenge) is invoked to generate signed

	"deviceId": "",
	"referenceId": "",
	"flowType": " assertIdentity",
	"requestData":{
	"assertIdentity":{"challenge":"xzy"}
	}}
	
	Response Parameter
	
	{
	"status":"Status_Codes",
	"flowType":"assertIdenity",
	"responseData": {
	"id": "1234567890",
	"seq": "123",
	"product": "Digital Rupee/Upi-Lite",
	"currency": "INR",
	"country": "0356",
	"type": "consumer",
	"mcc": null,
	"purseCertificate":
	"MIIBJDCBywIBBDAKBggqhkjOPQQDA...",
	"signature":
	"MEYCIQDBKaRP/4WZOWYWl4+zE5Wu+..."
	
	}
	}
		proof of the purse's identity and other attributes.
		
		challenge: a string type. An unpredictable nonce value provided by the remote User Agent in its OpenContext request/response message

4	verifyIdentity
	
	Request Parameter
	{
	"mobile": "930303033",
	"deviceId": "",
	"referenceId": "",
	"flowType": "verifyIdentity",
	"requestData": {
	"verifyIdentity":{
	"id": "1234567890",
	"seq": "123",
	"product": "Digital Rupee",
	"currency": "INR",
	"country": "0356",
	"type": "consumer",
	"mcc": null,
	"listKeyXmlPayload": "xmlPayload",
	"purseCertificate":
	"MIIBJDCBywIBBDAKBggqhkjOPQDVQ...",
	"signature":
	"MEYCIQDBKaRP/4WZOWY9n5ggCbSt..."
	}}
	}
	
	Response Parameter
	
	{
	"status":"Status_Codes:Ref Section 2.12",
	"flowType": "verifyIdentity",
	"responseData": {
	"isVerified ": "true / false"
	}
	}
		Flow types:
		verifyIdentity(requestData) is invoked to verify the given purse identity is valid.
		If the identity is verified, then the purse id (wallet id) and certificates are recorded for verification of subsequent signed value transfer signatures.

5	startDebit	Flow types: startDebit is invoked by:

	Request Parameter
	{
	"mobile": "930303033",
	"deviceId": "",
	"referenceId": "",
	"flowType": "startDebit",
	"requestData": {
	"payerId": "1122334455",
	"payerSeq": "457",
	"payerVpa": "abc@handle",--
	(optional) only needed for UPI-LITE
	"payeeId": "1234567890",
	"payeeSeq": "124",
	"payeeVpa": "xyz@handle2",--
	(optional) only needed for UPI-LITE
	"timestamp": "2023-03-16T10:45:26Z",
	"currency": "INR",
	"amount": "45.00",
	"tokenMix": [
	{"qty": "2","den": "20.00"},
	{"qty": "1","den": "5.00"}
	]
	}
	}
	
	Response Parameters:
	
	{
	"status":"Status_Codes:Ref Section 2.12",
	"flowType": "startDebit",
	"responseData": {
	"txnId":
	"f3e8907253e02926ee5d2c63e83efad71aa"}
	}
	}
		• The coordinating User Agent at start of first value transfer, or
		• The responding User Agent at start of second value transfer.

6	startCredit
	
	Request Parameter
	{
	"mobile": "930303033",
	"deviceId": "",
	"referenceId": "",
	"flowType": "startCredit",
	"requestData": {
	"payerId": "1234567890",
	"payerSeq": "123",
	"payerVpa": "xyz@handle2",--
	(optional) only needed for UPI-LITE
	"payeeId": "1122334455",
	"payeeSeq": "456",
	"payeeVpa": "abc@handle",--
	(optional) only needed for UPI-LITE
	"timestamp": "2023-03-16T10:45:26Z",
	"currency": "INR",
	"amount": "45.00",
	"tokenMix": [
	{"qty":"2","den": "20.00"},
	{"qty": "1","den": "5.00"}
	]
		Flow type: startCredit is invoked by:
		• The responding User Agent during first value transfer,
		• The coordinating User Agent during second value transfer.

	}
	}
	
	Response Parameters:
	
	{
	"status":"Status_Codes:Ref Section 2.12",
	"flowType": "startCredit",
	"responseData": {
	txnId":
	"f3e8907253e02926ee5d2c63e83efad71aa"
	"signature":
	"MEUCIQCGSkA6EzgyB1ZZRUo7dIPVI3O..."
	}
	}	

7	requestValue
	
	Request Parameter
	{
	"mobile": "930303033",
	"deviceId": "",
	"referenceId": "",
	"flowType": "requestValue",
	"requestData": {
	"signature":
	"MEUCIQCGSkA6EzgyB1ZZRUo7dIPVI3OucZl6..."
	}
	}
	
	Response Parameters:
	
	{
	"status":"Status_Codes:Ref purse error codes",
	"flowType": "requestValue",
	"responseData": {
	"Tokens": [
	"4oK5MjAgdG9rZW4sIGlkPTEyMy",
	"4oK5MjAgdG9rZW4sIGlkPTQ1",
	"4oK5NSB0b2tlbiwgaWQ9Nzg5"
	],
	"signature":
	"MEYCIQCeKeNTmTZl9LqVJ7ctjNJcPMmp+J..."
	}
	
	}
		Flow type: requestValue is invoked by:
		• The coordinating User Agent during first value transfer,
		• The responding User Agent during second value transfer.
		Note, if successful, requestValue() will debit value from the purse.

8	transferValue
	
	Request Parameter
	{
	"mobile": "930303033",
	"deviceId": "",
	"referenceId": "",
	"flowType": "transferValue",
	"requestData":{
	"tokens": [
	"4oK5MjAgdG9rZW4sIGlkPTEyMy",
	"4oK5MjAgdG9rZW4sIGlkPTQ1",
	"4oK5NSB0b2tlbiwgaWQ9Nzg5"
	],
		Flow type: transferValue is invoked by:
		• The responding User Agent during first value transfer,
		• The coordinating User Agent during second value transfer.
		Note, if successful, transferValue() will credit the transfer to the purse according to its product type:
		• For Digital Rupee, the transferred token(s) will be

	"signature": "MEYCIQCeKeNTmTZl9LqVJ7ctjNJcPMmp+J...", "listKeyXmlPayload": "xmlPayload" -- only needed for ERUPEE
	}
	}
	
	Response Parameters:
	
	{
	"status":"Status_Codes:Ref Section 2.12",
	"flowType": "transferValue",
	"responseData ": {
	"signature": "MEYCIQDzj0csnjycj3vOqcbVNbV6qjeSKs..."
	}
	}
		added to the Purse Token Store (as either 'spendable' or 'non-spendable' tokens) and the Purse Balance will be increased by the sum value of the tokens received,

9	acknowledgeValue
	
	Request Parameter
	{
	"mobile": "930303033",
	"deviceId": "",
	"referenceId": "",
	"flowType": "*acknowledgeValue*",
	"requestData":
	{
	"signature": "MEYCIQC8DdLtQr+V5+pZmPZkcpqV4SmglXW3v+"
	}
	}
	
	Response Parameters:
	
	{
	"status":"Status_Codes: Ref Section 2.12",
	"flowType": "acknowledgeValue",
	"responseData": {
	"status": "true / false"
	}
	}
		Flow type: acknowledgeValue is invoked by:
		• The coordinating User Agent during first value transfer,
		• The responding User Agent during second value transfer.
		Note, if Flow type: acknowledgeValue is not successfully invoked to end an active value transfer, then the purse will record the transfer in its Exception Log when starting another transfer

10	restartTransfer
	
	Request Parameter
	{
	"mobile": "930303033",
	"deviceId": "",
	"referenceId": "",
	"flowType": " restartTransfer",
	"requestData":
	{}
	}
	
	Response Parameters:
	
	{
	"status":"Status_Codes: Ref Section 2.12",	

UNIFIED PAYMENTS INTERFACE

	"flowType": " restartTransfer”,
	"responseData": {
	"tokens": [
	"4oK5MSB0b2tlbiwgaWQ9MTExLCBmb3JtYXQg..",
	"4oK5MSB0b2tlbiwgaWQ9MjIyLCBmb3JtYXQg..",
	"4oK5MSB0b2tlbiwgaWQ9MzMzLCBmb3JtY.."],
	-- only in case of ERUPEE
	"signature":
	"MEQCIGnycbA4lfrZvmEERJxTkP8ORGb2B3jDPy5p
	17 kwAdZPAiB4THkAw/3u/JlM+RSHNV1l5PnBBg
	f3eSraW6aF/y1ZVw==" }
	
	}	

6.1. Offline Transaction API

6.1.1. offLineTxnFlow()

Initiated Class : clServices

Method Definition : offLineTxnFlow()

Method Name	Inputs	Data Type	Outputs	Data Type
offLineTxnFlow	request	StringifiedJson	response	StringifiedJson

UNIFIED PAYMENTS INTERFACE

6.1.1.1 Request

This method is call to receive a challenge from CL that is used to get token from UPI/PSO. Once GetToken is completed, then Register of App to be executed immediately.

Method Name	Inputs	Data Type	Outputs	Data Type
offLineTxnFlow	request	StringifiedJson	response	StringifiedJson

Parameter Name	Data Type	Description
Inputs		
request		
mobile	String	Customer’s Mobile Number
deviceId	String	Device ID unique and stable for the mobile device. Role could be fulfilled by UUID (for IOS) or androidId (for android) provided by the platform.
referenceId	String	For UPI-Lite : referenceId == accountRef

		For ERUPEE : referenceId == eRupeeWalletAddress
flowType	String	Possible Values : getPurseInfo, assertIdentity, verifyIdentity, startDebit, startCredit, requestValue, transferValue, acknowledgeValue, closeContext, restartTransfer
requestData	StringifiedJson	As per the flowType: Refer getPurseInfo, assertIdentity, verifyIdentity, startDebit, startCredit, requestValue, transferValue, acknowledgeValue, closeContext, restartTransfer

6.1.1.2. Response

Parameter Name	Data Type	Description
Inputs		
response		
flowType	String	Flow types Values : getPurseInfo, assertIdentity, verifyIdentity, startDebit, startCredit, requestValue, transferValue, acknowledgeValue, closeContext, restartTransfer
responseData	StringifiedJson	Value of responseData as per flowType: Refer: getPurseInfo, assertIdentity, verifyIdentity, startDebit, startCredit, requestValue, transferValue, acknowledgeValue, closeContext, restartTransfer

6.1.2. Get Purse Info

Initiated Class : clServices

Method Definition : offLineTxnFlow.flowType=getPurseInfo

Method Name. Element	Inputs	Data Type	Outputs	Data Type
offLineTxnFlow.flowType =getPurseInfo	requestData		responseData	StringifiedJson

6.1.2.1. Request

Parameter Name	Cond	Data Type	Description
requestData			

6.1.2.2. Response

Parameter Name	Cond	Data Type	Description
responseData			
Status	M	string	Indicates the success or failure of the request
id	M	string	Purse ID (Wallet id)
seq	M	string	Purse Sequence Number
state	M	string	Purse State
product	M	string	Purse Product Type "Digital Rupee" and "UPI Lite" are the only options defined in this specification.
currency	M	string	Purse Currency "INR" is the only option defined in this specification.
country	M	string	Purse Country "0356" is the only option defined in this specification.
type	M	string	Purse Type "consumer", "merchant" and "issuer" are the only options defined in this specification. (other purse types can also be defined and added here in the list as per future requirements)
mcc	M	string	Purse Merchant Category Code A field for indicating merchant category. Set to null or an empty string ("") if not relevant for the purse type.
balance	M	string	Purse Balance (in minor currency units, e.g., 7000 = ₹70.00).
tokenMix	C	array	Present if Purse Product Type equals "Digital Rupee". JSON formatted data comprising the number and denomination of all tokens in the Purse Token Store.
availableBalance	C	string	Present if Purse Product Type equals "Digital Rupee".

UNIFIED PAYMENTS INTERFACE

			Purse Available Balance – the sum value of available tokens (excluding any 'non-spendable' or expired tokens).
availableTokenMix	C	array	Present if Purse Product Type equals "Digital Rupee". Array of the number and denomination of available tokens (excluding any non-spendable or expired tokens).
challenge	M	string	A nonce Purse Challenge value for the counterparty purse to use during mutual authentication.

6.1.3. Assert Identity

Initiated Class : clServices

Method Definition : offLineTxnFlow.flowType=assertIdentity

Method Name.Element	Inputs	Data Type	Outputs	Data Type
offLineTxnFlow. flowType= assertIdentity	requestData	StringifiedJson	responseData	StringifiedJson

UNIFIED PAYMENTS INTERFACE

6.1.3.1. Request

Parameter Name	Cond	Data Type	Description
requestData			
challenge	M	string	An unpredictable nonce value provided by the remote User Agent in its OpenContext request/response message

6.1.3.2. Response

Parameter Name	Cond	Data Type	Description
responseData			
id	M	string	Purse ID
seq	M	string	Sequence Number
product	M	string	Purse Product Type

currency	M	string	Purse Currency
country	M	string	Purse Country
type	M	string	Purse Type
mcc	M	string	Purse Merchant Category Code
purseCertificate	M	string	Base64 encoded Purse Public Key Certificate
signature	M	string	Base64 encoded purse signature computed over the id, product, currency, country, type, mcc, purseCertificate and a given challenge using the Purse Private Key

6.1.4. Verify Identity

Initiated Class : clServices

Method Definition : offLineTxnFlow.flowType=verifyIdentity

Method Name.Element	Inputs	Data Type	Outputs	Data Type
offLineTxnFlow. flowType= verifyIdentity	requestData	StringifiedJson	responseData	StringifiedJson

6.1.4.1. Request

Parameter Name	Cond	Data Type	Description
requestData			
id	M	string	Purse ID
seq	M	string	Sequence Number
product	M	string	Purse Product Type
currency	M	string	Purse Currency
country	M	string	Purse Country
type	M	string	Purse Type
mcc	M	string	Purse Merchant Category Code
listKeyXmlPayload	M	string	RespListKey(type:keyList) received from NPCI and given as a input while calling the
purseCertificate	M	string	Base64 encoded Purse Public Key Certificate
signature	M	string	Base64 encoded purse signature computed over the id, product, currency, country, type, mcc, purseCertificate and a

			given challenge using the Purse Private Key

6.1.4.2. Response

Parameter Name	Cond	Data Type	Description
responseData			
status	M	string	Indicates the success or failure of the request

6.1.5. Start Debit

Initiated Class : clServices

Method Definition : offLineTxnFlow.flowType= startDebit

Method Name.Element	Inputs	Data Type	Outputs	Data Type
offLineTxnFlow. flowType= startDebit	requestData	StringifiedJson	responseData	StringifiedJson

6.1.5.1. Request

Parameter Name	Cond	Data Type	Description
requestData			
payerId	M	string	The payer Purse ID
payerSeq	M	string	The payer Purse Sequence Number
payeeId	M	string	The payee Purse ID
payeeSeq	M	string	The payee Purse Sequence Number
timestamp	M	string	The date/time of the transaction (UTC Format)
currency	M	string	The transaction currency ("INR" is the only option defined in this specification)
amount	M	string	The amount of value to be transferred
tokenMix	C	string	Required if Purse Product Type equals "Digital Rupee" JSON formatted data comprising the number and denomination of tokens to transfer. The sum value of tokenMix must equal the amount.

6.1.5.2. Response

Parameter Name	Cond	Data Type	Description
responseData			
txnId	M	string	Transaction id First MSB 35 character txnId = sha 1(payerID.payerSeq.payerChallenge.payeeID.payeeSeq. payeeChallenge)

6.1.6. Start Credit

Initiated Class : clServices

Method Definition : offLineTxnFlow.flowType= startCredit

Method Name.Element	Inputs	Data Type	Outputs	Data Type
offLineTxnFlow. flowType= startCredit	requestData	StringifiedJson	responseData	StringifiedJson

6.1.6.1 Request

Parameter Name	Cond	Data Type	Description
requestData			
Status	M	string	Indicates the success or failure of the request
payerId	M	string	The payer Purse ID (Wallet id)
payerSeq	M	string	The payer Purse Sequence Number
payeeId	M	string	The payee Purse ID (Wallet id)
payeeSeq	M	string	The payee Purse Sequence Number
timestamp	M	string	The date/time of the transaction (UTC Format)
currency	M	string	The transaction currency ("INR" is the only option defined in this specification)
amount	M	string	The amount of value to be transferred

tokenMix	C	string	Required if Purse Product Type equals "Digital Rupee" JSON formatted data comprising the number and denomination of tokens to transfer. The sum value of tokenMix must equal the amount.

6.1.6.2 Response

Parameter Name	Cond	Data Type	Description
responseData			
txnId	M	string	Transaction id
signature	M	string	Base64 encoded request signature computed over the startTransferData using the responder purse private key

6.1.7. Request Value

Initiated Class : clServices

Method Definition : offLineTxnFlow.flowType= requestValue

Method Name.Element	Inputs	Data Type	Outputs	Data Type
offLineTxnFlow. flowType= requestValue	requestData	StringifiedJson	responseData	StringifiedJson

6.1.7.1 Request

Parameter Name	Cond	Data Type	Description
requestData			
signature	M	string	base64 encoded purse signature computed over *Purse Transaction Data Object* data by the payee purse

6.1.7.2 Response

Parameter Name	Cond	Data Type	Description
responseData			
tokens	M	jSON array	A json array of encoded base-64 tokens information selected for offline transaction
signature	M	string	a base64 encoded purse signature computed over *Purse Transaction Data Object* data by the payer purse
status	M	string	Indicates the success or failure of the request (see section **Error! Reference source not found.**)

6.1.8. Transfer Value

Initiated Class : clServices

Method Definition : offLineTxnFlow.flowType= transferValue

Method Name.Element	Inputs	Data Type	Outputs	Data Type
offLineTxnFlow. flowType= transferValue	requestData	StringifiedJson	responseData	StringifiedJson

6.1.8.1 Request

Parameter Name	Cond	Data Type	Description
requestData			
Tokens	M	Array	A list of tokens selected for offline transaction
Signature	M	string	base64 encoded purse signature computed over *Purse Transaction Data Object* data by the payer purse.

6.1.8.2 Response

Parameter Name	Cond	Data Type	Description

responseData			
Signature	C	string	base64 encoded purse signature computed over Purse Transaction Data Object data by the payee purse.
status	M	String	Indicates the success or failure of the request (see section **Error! Reference source not found.**)

6.1.9. Acknowledge Value

Initiated Class : clServices

Method Definition : offLineTxnFlow.flowType= acknowledgeValue

Method Name.Element	Inputs	Data Type	Outputs	Data Type
offLineTxnFlow. flowType= acknowledgeValue	requestData	StringifiedJson	responseData	StringifiedJson

6.1.9.1 Request

Parameter Name	Cond	Data Type	Description
requestData			
signature	M	string	Base64 encoded acknowledge signature computed using the coordinator purse private key

6.1.9.2 Response

Parameter Name	Cond	Data Type	Description
responseData			
status	M	string	Indicates the success or failure of the request (see section **Error! Reference source not found.**)

6.1.10. Restart Transfer

Initiated Class : clServices

Method Definition : offLineTxnFlow.flowType= restartTransfer

Method Name.Element	Inputs	Data Type	Outputs	Data Type
offLineTxnFlow. flowType= restartTransfer	requestData	StringifiedJson	responseData	StringifiedJson

6.1.10.1 Request

Parameter Name	Cond	Data Type	Description
requestData			

6.1.10.2 Response

Parameter Name	Cond	Data Type	Description
responseData			
tokens	M	Array	A list of token selected for re-transfer
signature	M	string	base64 encoded purse signature – the last signature produced by the restarting purse.
status	M	integer	Indicates the success or failure of the request (see section 2.12)

6.2. Offline Logs Syncing

Once Customer comes Online, Customer is supposed to Sync Offline Logs to realize his updated balance. For that Customer has to trigger getCredential(LOGS).

Onboarding flow of UPI-Lite and ERUPEE offline is similar for now. If we do onboarding on CLServices Instance of UPI, it will trigger the UPI-LITE onboarding. And if we do onboarding on CLServices Instance of ERUPEE, it will trigger the ERUPEE onboarding. And based on the new flag "enableOfflineTxn" in configuration, merchants can tell CL to generate keyPair for offline as well

6.2.1. Request

Initiated Class : clServices

Method Definition : getCredential(LOGS)

Parameter Name	Data Type	M/O/C	Length	Description
Inputs				
keyCode	String			NPCI
listKeyPayLoad	String			NPCI list keys xml
controls	String			{ "CredAllowed": [{ "type": "DEVICE", "subtype": "LOGS", "dType": "ALPH", "dLength": "2048" }] }
configuration	String			{ “getDeviceDetails”:"true/false"}
salt	String			{"txnId":["MAY96c1d03348df4a8f92ad07019cb0ac39"],“deviceId”: ”ABCDEF”,”accountRef”(for UPI-Lite case only) :”Ac779771d9c746afad3a7650b23132d”,“eRupeeWalletAddress”(for ERUPEE case only) :”A1769171r9c716afad1a7650c2398s2”,”txnTimestamp”:”2022-09-15T14:58:28+05:30”,“appId”:”com.psp1.app”,“mobileNumber”:”919002050725”,”credType”:[”clsync/walletsync”], “random”: “<Base64 String>” } Note: clsync used for log sync in UPI Lite walletsync used for log sync in ERUPEE
payInfo	String			[]
trust	String			{“logs” : “trust-to-be-validated”}
languagePref	String			en_US
callback	CLRemoteResultReceiver			Instance of the CLRemoteResultReceiver to receive a response.

6.2.2 Reponse

Parameter Name	Data Type	M/O/C	Length	Description

Outputs				
CredBlock	String			{"LOGS": {"logs": {"type": "DEVICE","subtype": "LOGS", "data": {"code": "NPCI/NPCI-LITE","ki": "20220622", "encryptedBase64String": "2.5.token-v2.0_v1/2.5.lite-v2.0_v1|<encrypted base 64 encoded authentication data>"}}}, "det": "2.5-v2.0_v1|<encrypted base 64 encoded data>"}

After Pushing logs, NPCI will give an ARPC which needs to be passed to CL in registerState(..) API.

Appendix

NPCI list keys xml for the Normal User


```
"<?xml version= \"1.0\" encoding= \"UTF-8\"?>
<ns2:RespListKeys xmlns:ns2= \"http://npci.org/upi/schema/\">
    <Head msgId= \"1vRRpw1b9vnsr5cfpu2V\" orgId= \"NPCI\" ts= \"2022-05-
01T23:59:44+05:30\" ver= \"2.0\" />
    <Resp reqMsgId= \"RNBd4f8394c82bc48279b991bcf17dddf80\" result=
\"SUCCESS\" />
    <Txn id= \"RNBe05e00ce1ad340d6a69ed2a570c4e895\" note= \"get token\"
refId= \"123456789\" refUrl= \"http://www.IDBIbank.com\" ts= \"2022-05-
02T00:00:00+05:30\" type= \"ListKeys\" />
    <keyList>
        <key code= \"NPCI\" ki= \"20150822\" owner= \"NPCI\" type= \"PKI\">
            <keyValue xmlns:xs= \"http://www.w3.org/2001/XMLSchema\"
xmlns:xsi= \"http://www.w3.org/2001/XMLSchema-instance\" xsi:type=
\"xs:string\">MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAuMKxWfy0WcPp98muB
Wa6yhpmb6ZGZGSKHRIOv05UlIN5TbUPl6yEerh7Wj0+JyKfsOntRdAVhkLJGRoHwH6gEEeFNHge7kP
ea/B33
cQAbqa39mnP5F1aaZT3tjJnKrfI1Wum0crdb7dAMzft4JILOEa+s3Uh7OdYEl/Xp7EisdSoJ345Cj0
LTfLZE
QzRdVGovXZrfLByJysH11V9tDrIVv75C/3UndwjHt3NrqzNBoUMh5VZRFkcwuebUAkhIed5gvoysJw
d0yYGr
AUXNrXJJDTAj5diCuasWyfWZR9lsX5l14hdxF+lqadR/pgII53DW5oEy2LMXgvt2u/qmSml8wIDAQA
B
        </key>
        <key code= \"NPCI-LITE\" ki= \"202112281\" owner= \"NPCI\" type=
\"PKI\">
            <keyValue xmlns:xs= \"http://www.w3.org/2001/XMLSchema\"
xmlns:xsi= \"http://www.w3.org/2001/XMLSchema-instance\" xsi:type=
\"xs:string\">MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAhOHi38AxMorovPqqI
ncRDNxIhJG1XB1DwdICJcZPfBpYN5V1o5rYU4HMIChQ0GhSnbbQ6qRhY3SetBFCaVAALLAZd6KdtmG
pIYonf
LiS5pXNmohiDfL1i1we76TLr5KqGaHDkuUtDoScPC+wol04fByscHU3Y9ahNtxnWGyRiKng38qv0ZZ
Eg0KIP
WhPYybUmKNClGyk6x6NpOGRYdbt02PSfpvD3Y6+4KXNkxEqxFTAvUhgw7YGIsrl/3+7K3X56g8vSYW
wJoF1J
CxoxkCxXE+9xUgihaZT5Vus4AQ8MqXdaGZ+TA27S1zOytEdMZj/o2KsW/bXhsRUko141M9U+QIDAQA
B
        </key>
        <key code= \"NPCI-LITE\" ki= \"202112282\" owner= \"NPCI\" type=
\"PKI\">
            <keyValue xmlns:xs= \"http://www.w3.org/2001/XMLSchema\"
xmlns:xsi= \"http://www.w3.org/2001/XMLSchema-instance\" xsi:type=
\"xs:string\">MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAk9KOI4qjnA9qBPPYo
```

```
+EGGCBkGNpn4J5UTU6uGE2vkHtwq1hS8/Pv52x7FHvqoWP4/Kc71Af5Rnd5ZGlBR4mcgth4qPwz9QZ
3kbnQC
LWgvACGEKXWp+6Sg7f83lyUngSuPOGyirOWk5hrY0nAtNtNoq5Wiz4t0z6UemTDF4FYOm3hqfVbhet
+fCemZ
6bOwIEKlmdUseH6ePzJeskRISwohRaRePCPRgJrOmGLYI+dro6FHUBVHGHMh4EAOrIJs89d7MslALr
3NBwSP
8oOSmTEKqrxWiJqJoDshnqTYKfs4PpFePv9ZsSuFrAJkvDplw06s2hWK6alREgBU1wYPncZ9QIDAQA
B
        </key>
    </keyList>
    <keyListSignature code= \"PRIMARY\" owner= \"NPCI\" type= \"SIGNATURE\">
        <signatureValue xmlns:xs= \"http://www.w3.org/2001/XMLSchema\"
xmlns:xsi= \"http://www.w3.org/2001/XMLSchema-instance\" xsi:type=
\"xs:string\">kt5OTSeL5e6dpEcYSrYIoInDLKwMW1UPtTTFCGIZf6yLMU55jn7AOSRFyyeEi
MwxPXqkdfLz2XeOM45MYeS99xU+9y1jF0QxR9gshF+Wtj7kgpSVQN6BBZa5lKPX5XWUMQR8zPr4nQ9
ndKX+r
LPgGzeZiJpdxIPNK5aXAcBSuUIm4GDetDeujVKcgfjoMGICTowXa433Qk0HljW5h3Jh+3j1MYUFhFw
yqxIUw
G/9J4/IBNnXXjaNTmZDS5UHMUozwU7oxS91BUh8iTZUEz3KFPf0goZGoXwFxmkDfvVc3ydcvi7zKIp
Yrtxzx
JMkqzfh4xiRm/1MpiUsEVli5UAenQ==</signatureValue>
    </keyListSignature>
    <Signature xmlns= \"http://www.w3.org/2000/09/xmldsig#\">
        <SignedInfo>
            <CanonicalizationMethod Algorithm=
\"http://www.w3.org/TR/2001/REC-xml-c14n-20010315\" />
            <SignatureMethod Algorithm= \"http://www.w3.org/2001/04/xmldsig-
more#rsa-sha256\" />
            <Reference URI= \"\">
                <Transforms>
                    <Transform Algorithm=
\"http://www.w3.org/2000/09/xmldsig#enveloped-signature\" />
                </Transforms>
                <D igestMethod Algorithm=
\"http://www.w3.org/2001/04/xmlenc#sha256\" />
                <DigestValue>rFHLm0ONv+hqBCYC
a4sP+BvUvzKuNpfmOG8RWWDhgCI=</DigestValue>
            </Reference>
        </SignedInfo>
        <SignatureValue>u
LDPMtR+wRc19T2dU4rYgrtF8DPIkSkwo/xZ5sUwAykEH1mj+Ge9SOAHpdrYw1p9aFRY9dg2p1i1DpK
9S1kDG
X3MEgk24keSw4QR1imly9br5oM7v4MJ5Ge1lPEGb1tJygKwTs3gRv924PiGFWhmCLHWgWH6banEMaB
5loNGN
7r8IInovYgiJeZm0H/Av/wu0C9t5T1N9AeWpXB03e5MooyaNWg7pxnaju1+H3km59/Pqaj2od+/Gcn
fCIgiW
```

```
zSZmEX6eSJHgjvnHSkhv6YO5dWuedpiDsZqdbty/FNPPK63x/cVtdmWP8X9fHfNehbXl9TJurZyIQC
MMfJE3
vNEAA==</SignatureValue>
    </Signature>
</ns2:RespListKeys>"
```


NPCI list keys xml for Enabling Offline for the User


```
<?xml version= \"1.0\" encoding= \"UTF-8\"?>
<ns2:RespListKeys xmlns:ns2= \"http://npci.org/upi/schema/\">
    <Head msgId= \"1vRRpw1b9vnsr5cfpu2V\" orgId= \"NPCI\" ts= \"2022-05-
01T23:59:44+05:30\" ver= \"2.0\" />
    <Resp reqMsgId= \"RNBd4f8394c82bc48279b991bcf17dddf80\" result=
\"SUCCESS\" />
    <Txn id= \"RNBe05e00ce1ad340d6a69ed2a570c4e895\" note= \"get token\"
refId= \"123456789\" refUrl= \"http://www.IDBIbank.com\" ts= \"2022-05-
02T00:00:00+05:30\" type= \"ListKeys\" />
    <keyList>
        <key code= \"NPCI\" ki= \"20150822\" owner= \"NPCI\" type= \"PKI\">
            <keyValue xmlns:xs= \"http://www.w3.org/2001/XMLSchema\"
xmlns:xsi= \"http://www.w3.org/2001/XMLSchema-instance\" xsi:type=
\"xs:string\">MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAuMKxWfy0WcP
p98muBWa6yhpmb6ZGZGSKHRIOv05UlIN5TbUPl6yEerh7Wj0+JyKfsOntRdAVhkLJGRoHwH6gEEe
FNHge7kPea/B33cQAbqa39mnP5F1aaZT3tjJnKrfI1Wum0crdb7dAMzft4JILOEa+s3Uh7OdYEl/Xp
7Eis
dSoJ345Cj0LTfLZEQzRdVGovXZrfLByJysH11V9tDrIVv75C/3UndwjHt3NrqzNBoUMh5VZRFkcwue
bUAkhIed5gvoysJwd0yYGrAUXNrXJJDTAj5diCuasWyfWZR9lsX5l14hdxF+lqadR/pgII53DW5oEy
2L
MXgvt2u/qmSml8wIDAQAB</keyValue>
        </key>
        <key code= \"NPCI-LITE\" ki= \"202112281\" owner= \"NPCI\" type=
\"PKI\">
            <keyValue xmlns:xs= \"http://www.w3.org/2001/XMLSchema\"
xmlns:xsi= \"http://www.w3.org/2001/XMLSchema-instance\" xsi:type=
\"xs:string\">MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAhOHi38AxMoro
vPqqIncRDNxIhJG1XB1DwdICJcZPfBpYN5V1o5rYU4HMIChQ0GhSnbbQ6qRhY3SetBFCaVAALLA
Zd6KdtmGpIYonfLiS5pXNmohiDfL1i1we76TLr5KqGaHDkuUtDoScPC+wol04fByscHU3Y9ahNtxnW
GyRiKng38qv0ZZEg0KIPWhPYybUmKNClGyk6x6NpOGRYdbt02PSfpvD3Y6+4KXNkxEqxFTAvUh
gw7YGIsrl/3+7K3X56g8vSYWwJoF1JCxoxkCxXE+9xUgihaZT5Vus4AQ8MqXdaGZ+TA27S1zOytEd
MZj/o2KsW/bXhsRUko141M9U+QIDAQAB</keyValue>
        </key>
        <key code= \"NPCI-LITE\" ki= \"202112282\" owner= \"NPCI\" type=
\"PKI\">
            <keyValue xmlns:xs= \"http://www.w3.org/2001/XMLSchema\"
xmlns:xsi= \"http://www.w3.org/2001/XMLSchema-instance\" xsi:type=
\"xs:string\">MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAk9KOI4qjnA9qB
PPYo+EGGCBkGNpn4J5UTU6uGE2vkHtwq1hS8/Pv52x7FHvqoWP4/Kc71Af5Rnd5ZGlBR4mcgth4qP
```

```
wz9QZ3kbnQCLWgvACGEKXWp+6Sg7f83lyUngSuPOGyirOWk5hrY0nAtNtNoq5Wiz4t0z6UemTDF
4FYOm3hqfVbhet+fCemZ6bOwIEKlmdUseH6ePzJeskRISwohRaRePCPRgJrOmGLYI+dro6FHUBVH
GHMh4EAOrIJs89d7MslALr3NBwSP8oOSmTEKqrxWiJqJoDshnqTYKfs4PpFePv9ZsSuFrAJkvDplw0
6s2hWK6alREgBU1wYPncZ9QIDAQAB</keyValue>
        </key>
        <key code= \"PRIMARY-OFFLINE\" ki= \"202112281\" owner= \"NPCI\" type=
\"PKI\">
            <keyValue xmlns:xs= \"http://www.w3.org/2001/XMLSchema\"
xmlns:xsi= \"http://www.w3.org/2001/XMLSchema-instance\" xsi:type=
\"xs:string\">LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSUNTekNDQWZLZ0F3SU
JBZ0lVYkxJdG9XTE9SRkNDalVDWDRIUThPODg4cVNVd0NnWUlLb1pJemowRUF3SXcKUXpFZ
E1Cc0dBMVVFQ2d3VVJYaGhiWEJzWlNCT1VFTkpJRkp2YjNRZ1EwRXhDekFKQmdOVkJBWVR
Ba2xPTVJVdwpFd1lEVlFRRERBeHVjR05wWDNKdmIzUmZZMkV3SGhjTk1qTXdOakU0TURnME1
ETXdXaGNOTkRNd05qRXpNRGcwCk1ETXdXakJETVIwd0d3WURWUVFLREJSRmVHRnRjR3hsS
UU1UVEwa2dVbTl2ZENCRFFURUxNQWtHQTFVRUJoTUMKU1U0eEZUQVRCZ05WQkFNTURH
NXdZMmxmY205dmRGOWpZVEJaTUJNR0J5cUdTTTQ5QWdFR0NDcUdTTTQ5QXdFSApBMElB
QkVCUElMV1NtM2JSL2JsK2F3V3pkejdmSllUdER6Q05pYlRJdXJkRGl5ejhTVWsyWUFybUNFYzZ
4MHZtCmhsYmxaenNNRWxwa0JvcHZqWjhNRkhOWkN5R2pnY013Z2NBd0R3WURWUjBUQVFIL
0JBVXdBd0VCL3pBZEJnTlYKSFE0RUZnUVUzL21OMEY2d29sWG9NR3RiUjY1ZlBjVEVsRUl3Z
mdZRFZSMGpCSGN3ZFlBVTMvbU4wRjZ3b2xYbwpNR3RiUjY1ZlBjVEVsRUtoUjZSRk1FTXhIVE
FiQmdOVkJBb01GRVY0WVcxd2JHVWdUbEJEU1NCU2IyOTBJRU5CCk1Rc3dDUVlEVlFRR0V3Sk
pUakVWTUJNR0ExVUVBd3dNYm5CamFWOXliMjkwWDJOaGdoUnNzaTJoWXM1RVVJS04KUUp
mZ2REdzd6enlwSlRBT0JnTlZIUThCQWY4RUJBTUNBb1F3Q2dZSUtvWkl6ajBFQXdJRFJ3QXdSQ
UlnVVlXMQphdXlRaTNWaTd4VlRVcFR1bGsrY2Q4a0trRTBoZWQ0Znk3NnBJczhDSURFdGxpVytlb
nhrNExMSDhVOUhTcDB5ClVyZWVsMWU5T3hxU1dlM29HbUxXCi0tLS0tRU5EIENFUlRJRklDQV
RFLS0tLS0K</keyValue>
        </key>
        <key code= \"SECONDARY-OFFLINE\" ki= \"202112282\" owner= \"NPCI\"
type= \"PKI\">
            <keyValue xmlns:xs= \"http://www.w3.org/2001/XMLSchema\"
xmlns:xsi= \"http://www.w3.org/2001/XMLSchema-instance\" xsi:type=
\"xs:string\">LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSUNTekNDQWZLZ0F3SU
JBZ0lVYkxJdG9XTE9SRkNDalVDWDRIUThPODg4cVNVd0NnWUlLb1pJemowRUF3SXcKUXpFZ
E1Cc0dBMVVFQ2d3VVJYaGhiWEJzWlNCT1VFTkpJRkp2YjNRZ1EwRXhDekFKQmdOVkJBWVR
Ba2xPTVJVdwpFd1lEVlFRRERBeHVjR05wWDNKdmIzUmZZMkV3SGhjTk1qTXdOakU0TURnME1
ETXdXaGNOTkRNd05qRXpNRGcwCk1ETXdXakJETVIwd0d3WURWUVFLREJSRmVHRnRjR3hsS
UU1UVEwa2dVbTl2ZENCRFFURUxNQWtHQTFVRUJoTUMKU1U0eEZUQVRCZ05WQkFNTURH
NXdZMmxmY205dmRGOWpZVEJaTUJNR0J5cUdTTTQ5QWdFR0NDcUdTTTQ5QXdFSApBMElB
QkVCUElMV1NtM2JSL2JsK2F3V3pkejdmSllUdER6Q05pYlRJdXJkRGl5ejhTVWsyWUFybUNFYzZ
4MHZtCmhsYmxaenNNRWxwa0JvcHZqWjhNRkhOWkN5R2pnY013Z2NBd0R3WURWUjBUQVFIL
0JBVXdBd0VCL3pBZEJnTlYKSFE0RUZnUVUzL21OMEY2d29sWG9NR3RiUjY1ZlBjVEVsRUl3Z
mdZRFZSMGpCSGN3ZFlBVTMvbU4wRjZ3b2xYbwpNR3RiUjY1ZlBjVEVsRUtoUjZSRk1FTXhIVE
FiQmdOVkJBb01GRVY0WVcxd2JHVWdUbEJEU1NCU2IyOTBJRU5CCk1Rc3dDUVlEVlFRR0V3Sk
pUakVWTUJNR0ExVUVBd3dNYm5CamFWOXliMjkwWDJOaGdoUnNzaTJoWXM1RVVJS04KUUp
mZ2REdzd6enlwSlRBT0JnTlZIUThCQWY4RUJBTUNBb1F3Q2dZSUtvWkl6ajBFQXdJRFJ3QXdSQ
UlnVVlXMQphdXlRaTNWaTd4VlRVcFR1bGsrY2Q4a0trRTBoZWQ0Znk3NnBJczhDSURFdGxpVytlb
nhrNExMSDhVOUhTcDB5ClVyZWVsMWU5T3hxU1dlM29HbUxXCi0tLS0tRU5EIENFUlRJRklDQV
```

```
RFLS0tLS0K</keyValue>
        </key>
        <key code="ISSUER-ADDRESS" ki="202112284" owner="ISSUER" type="PKI">
            <keyValue xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:type="xs:string">MHgzMzMyYzVmN2YzNmMxODkzNDc3OTllMTdlMmYzZTgyNDZhMjBhMzc
z</keyValue>
        </key>
    </keyList>
    <keyListSignature code= \"PRIMARY\" owner= \"NPCI\" type= \"SIGNATURE\">
        <signatureValue xmlns:xs= \"http://www.w3.org/2001/XMLSchema\"
xmlns:xsi= \"http://www.w3.org/2001/XMLSchema-instance\" xsi:type=
\"xs:string\">nwPVh787UdsTVnp+06FQrY8wmYdSma44IUP75o+yz5xT2nHXAV/3ekUlb/3z1
OTLWSWZ4p88C6GdxfaLLLMAORCLE/8V76BLRkSVYM4t5K+jnMjArwmspX8RJrOcR/1lTIPUMfn
gHPBT8LXyDLJNlcPZWkFFWTtwTrznVdRiQNPjhKwxLQCOtMQa8F5PKHKEjqbtkFRB4Ch+2zkyy
oAadX8/yGkVFWAzu5c0HHBknBwPOLzpcfAvJfuWFkSx8h0jHUErW5xUQVEHxLIdDKeA+un7kb/J
RzFEUcsVD0wzu2yYb+qr0FqW+O732Ws4LMYcHuOU1lF6mF/cP3f2nwTUvA==</signatureValue>
        </
    </keyListSignature>
    <Signature xmlns= \"http://www.w3.org/2000/09/xmldsig#\">
        <SignedInfo>
            <CanonicalizationMethod Algorithm=
\"http://www.w3.org/TR/2001/REC-xml-c14n-20010315\" />
            <SignatureMethod Algorithm= \"http://www.w3.org/2001/04/xmldsig-
more#rsa-sha256\" />
            <Reference URI= \"\">
                <Transforms>
                    <Transform Algorithm=
\"http://www.w3.org/2000/09/xmldsig#enveloped-signature\" />
                </Transforms>
                <DigestMethod Algorithm=
\"http://www.w3.org/2001/04/xmlenc#sha256\" />
                <DigestValue>rFHLm0ONv+hqBCYCa4sP
+BvUvzKuNpfmOG8RWWDhgCI=</DigestValue>
            </Reference>
        </SignedInfo>
        <SignatureValue>uLDP
MtR+wRc19T2dU4rYgrtF8DPIkSkwo/xZ5sUwAykEH1mj+Ge9SOAHpdrYw1p9aFRY9dg2p1i1DpK9S
1kDGX3MEgk24keSw4QR1imly9br5oM7v4MJ5Ge1lPEGb1tJygKwTs3gRv924PiGFWhmCLHWgWH
6banEMaB5loNGN7r8IInovYgiJeZm0H/Av/wu0C9t5T1N9AeWpXB03e5MooyaNWg7pxnaju1+H3km5
9/Pqaj2od+/GcnfCIgiWzSZmEX6eSJHgjvnHSkhv6YO5dWuedpiDsZqdbty/FNPPK63x/cVtdmWP8X
9f
HfNehbXl9TJurZyIQCMMfJE3vNEAA==</SignatureValue>
    </Signature>
</ns2:RespListKeys>
```

GetLite/GetWallet for Other Services

```
<?xml version= \"1.0\" encoding= \"UTF-8\"?>
<ns2:RespListKeys xmlns:ns2= \"http://npci.org/upi/schema/\">
    <Head msgId= \"1GRDpegAQsERNengqkHT\" orgId= \"NPCI\" ts= \"2016-12-
20T03:16:14+05:30\" ver= \"1.0\" />
    <Resp reqMsgId= \"AXI4006a149b21f47de8d57c8165d844068 \" result= \"SUCCESS
\" />
    <Txn id= \"AXIb50ca48e9c854339b5d7cbbf5da81c9f \" note= \"List Keys \"
refId= \"1 \" refUrl= \"http://olivecrypto.com/abc \" />
    <paramsList>
        <param code= \"DEVICE-CERT\" owner= \"NPCI \">
            <paramValue xmlns:ns3= \"http://npci.org/cm/schema/\" xmlns:xs=
\"http://www.w3.org/2001/XMLSchema \" xmlns:xsi=
\"http://www.w3.org/2001/XMLSchema-instance \" xsi:type=
\"xs:string\">"+DEVICE_CERTIFICATE_VALUE+"</paramValue>
        </param>
        <param code= \"STATE\" owner= \"NPCI \">
            <paramValue xmlns:ns3= \"http://npci.org/cm/schema/\" xmlns:xs=
\"http://www.w3.org/2001/XMLSchema \" xmlns:xsi=
\"http://www.w3.org/2001/XMLSchema-instance \" xsi:type=
\"xs:string\">"+STATE_VALUE+"</paramValue>
        </param>
    </paramsList>
    <paramListSignature code= \"PRIMARY\" owner= \"NPCI\" type= \"SIGNATURE\">
        <signatureValue xmlns:xs= \"http://www.w3.org/2001/XMLSchema\"
xmlns:xsi= \"http://www.w3.org/2001/XMLSchema-instance\" xsi:type=
\"xs:string\">"+PARAMLIST_SIGNATURE_VALUE+"</signatureValue>
    </paramListSignature>
</ns2:RespListKeys>
```

GetLite/GetWallet for Offline Service

```
<?xml version= \"1.0\" encoding= \"UTF-8\"?>
<ns2:RespListKeys xmlns:ns2= \"http://npci.org/upi/schema/\">
    <Head msgId= \"1GRDpegAQsERNengqkHT\" orgId= \"NPCI\" ts= \"2016-12-
20T03:16:14+05:30\" ver= \"1.0\" />
    <Resp reqMsgId= \"AXI4006a149b21f47de8d57c8165d844068 \" result= \"SUCCESS
\" />
    <Txn id= \"AXIb50ca48e9c854339b5d7cbbf5da81c9f \" note= \"List Keys \"
refId= \"1 \" refUrl= \"http://olivecrypto.com/abc \" />
    <paramList>
        <param code= \"DEVICE-CERT\" owner= \"NPCI\">
            <paramValue xmlns:ns3= \"http://npci.org/cm/schema/\" xmlns:xs=
\"http://www.w3.org/2001/XMLSchema \" xmlns:xsi=
```

```
\"http://www.w3.org/2001/XMLSchema-instance \" xsi:type=
\"xs:string\">MIIBIDANBgkqhkiG9w0BAQEFAAOCAQ....</paramValue>
        </param>
        <param code= \"DEVICE-CERT-OFFLINE\" owner= \"NPCI \">
            <paramValue xmlns:ns3= \"http://npci.org/cm/schema/\" xmlns:xs=
\"http://www.w3.org/2001/XMLSchema \" xmlns:xsi=
\"http://www.w3.org/2001/XMLSchema-instance \" xsi:type=
\"xs:string\">MIIBIDANBgkqhkiG9w0BAQEFAAOCAQ....</paramValue>
        </param>
        <param code= \"STATE\" owner= \"NPCI \">
            <paramValue xmlns:ns3= \"http://npci.org/cm/schema/\" xmlns:xs=
\"http://www.w3.org/2001/XMLSchema \" xmlns:xsi=
\"http://www.w3.org/2001/XMLSchema-instance \" xsi:type=
\"xs:string\">MIIBIDANBgkqhkiG9w0BAQEFAAOCAQ....</paramValue>
        </param>
    </paramList>
    <paramListSignature code= \"PRIMARY\" owner= \"NPCI\" type= \"SIGNATURE\">
        <signatureValue xmlns:xs= \"http://www.w3.org/2001/XMLSchema\"
xmlns:xsi= \"http://www.w3.org/2001/XMLSchema-instance\" xsi:type=
\"xs:string\">MIIBIDANBgkqhkiG9w0BAQEFAAOCAQ...</signatureValue>
    </paramListSignature>
</ns2:RespListKeys>
```

Purse Error Codes

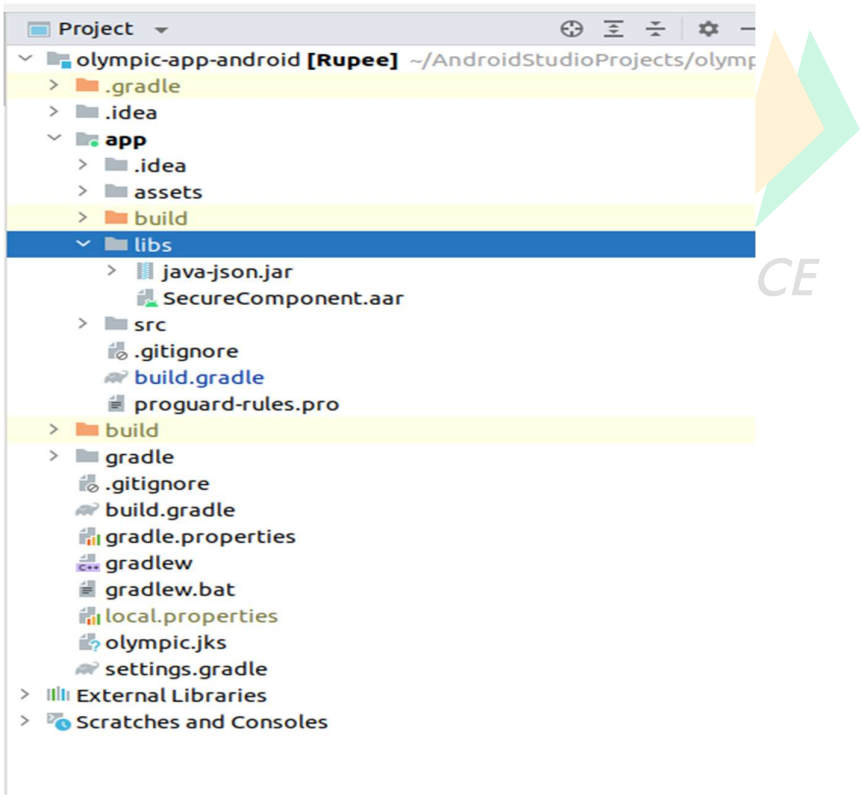
Status Code	Description
0	Success
100	Error – undefined cause
101	Conditions of use not satisfied
110	Insufficient funds
111	Tokens not available
112	Invalid token mix
113	Declined – purse action code
114	Invalid token
115	Duplicate token
120	Invalid issuer certificate
121	Invalid purse certificate
122	Incompatible purse product type
123	Incompatible purse currency
124	Incompatible purse country
130	Invalid authentication signature
131	Invalid request value signature

132	Invalid transfer value signature
133	Invalid acknowledge value signature
134	Invalid issuer signature
140	Purse on deny list

New Common CL file integration

1. Get the Common CL aar file.
    a. Extract the .zip/.tar file shared by NPCI.
    b. Please make sure while extracting the file use the correct password provided by NPCI.

2. Adding the .aar file in project.
    a. Open the Project in Android Studio.
    b. Go to project>app>libs folder.
    c. Remove the old SecureComponent.aar file from the libs folder.
    d. Copy Paste the new .aar file extracted in the `libs` folder.



3. Sync the Project
    a. Add below line in build.gradle of project.

implementation fileTree(dir: 'libs', include: ['*.jar', '*.aar'])

b. Go to project>app.
c. In Android Studio there will be yellow coloured build folder delete the folder.
d. Sync the Project.

Note: Please note while syncing the project the system is connected to open internet.

UNIFIED PAYMENTS INTERFACE