Verifone®

Technical Specification Electronic Cash Register Integration With CIMB X990 Android EDC Terminal

Revision History

Revision	Date	Project	Author	Description
1.0	4-Oct-2021	CIMB	Suwandhy	Initial document.
1.1	5-Oct-2021	CIMB	Yoga Suwandhy	Update & finalize the document
1.2	11-Oct-2021	CIMB	Singgih Adhimantoro	Update request message format
1.3	22-Oct-2021	CIMB	Singgih Adhimantoro	Update response message format, modify example
1.4	25-Oct-2021	CIMB	Singgih Adhimantoro	Update response message format
2.0	20-Dec-2021	CIMB	Michael Udjiawan	Add support using wifi connection
2.1	2-Feb-2022	CIMB	Michael Udjiawan	- Add Trans Type [QRIS CPM] - Change TransType [QRIS Sale] to [QRIS MPM]
2.2	21-Feb-2022	CIMB	Singgih Adhimantoro	Add support secure (SSL) wifi connection
2.3	3-Mar-2022	CIMB	Singgih Adhimantoro	Update request message format, add card number
2.4	8-Mar-2022	CIMB	Singgih Adhimantoro	Update provide windows and linux library
2.5	7-Apr-2022	CIMB	Singgih Adhimantoro	Add more supported transaction type
3.0	8-Apr-2022	CIMB	Singgih Adhimantoro	Add intent communication (intent document provided separately)
3.1	27-May-2022	CIMB	Michael Udjiawan	Update provide android aar library
4.0	7-Jul-2022	CIMB	Singgih Adhimantoro	Add support Rest API method for wifi connection only
4.1	16-Jun-2023	CIMB	Yulius Eryanto	Add issuer for filler QRIS
4.2	21-Aug-2023	CIMB	Singgih Adhimantoro	Add snipped code for android library for packRequest

Contents

1 Introduction

This Technical Specification document describes how to integrate Merchant's ECR and CIMB X990 Android EDC Terminal.

This document specifies the transaction flow and the messaging between EDC Terminal and Merchant's ECR (or POS).

2 Terms and definitions

ECR

Electronic Cash Register, also called as POS (Point of Sale) terminal. In this document this term will be used to indicate any one or more of the following:

- an ECR
- an ECR controller that is connected to a number of ECR's
- a standalone PC
- a workstation or minicomputer
- a mainframe host

EDC Terminal

A terminal or secure device to performs card payment transactions. This terminal will be integrated to the ECR to performs payment transaction. The device will also be responsible for communicating with the appropriate host system using whatever link, protocol and message formats required. In this document, Terminal will refer to Verifone X990 Android terminal with the communication dongle, Verifone X990 Commbox.

Payment Application

The name of this application is CIMB Payment application. This is the certified android payment application installed in the Terminal to perform the payment transaction securely with the Acquiring bank (CIMB) host. This Application will interact with the user/cardholder/payer upon the ECR send request message to the terminal.

Serial

It is a standard communication protocol between two smart devices, that will be used to exchange the message between the Terminal and the ECR. It is almost completely free of errors; therefore, a simple protocol can do the job well. The physical connection is using RS233 port or USB port, and a Serial/USB cable.

Local Network Connection

It is a standard communication protocol between two smart devices using socket connection or secure socket connection (using self-sign certificate), that will be used to exchange the message between the Terminal and the ECR. The physical connection is using Network Router

Terminal Mode

There are two types of Terminal Mode, i.e: Standalone & ECR Integration Mode.

In the Standalone Mode, terminal supports all features of CIMB application. In this mode Cashier will need to enter the amount to the terminal manually.

In the ECR Integration Mode, the terminal will be connected to the ECR via RS232 cable or USB cable or Local Network Router. Cashier only need to choose the transaction type and amount in the ECR. Upon receipt the request message from ECR, the terminal will run the Payment Application. User/Payer just need to choose the payment method, Card payment or QRIS. When the transaction is finish, the terminal will send back response message to ECR.

VTI

A Virtual Terminal Interface is the specification of process flow and message protocol that enable terminal to perform an integration with ECR.

3 Hardware/Software Requirements & Connection

3.1 Hardware/Software Requirement

This solution requires the following hardware:

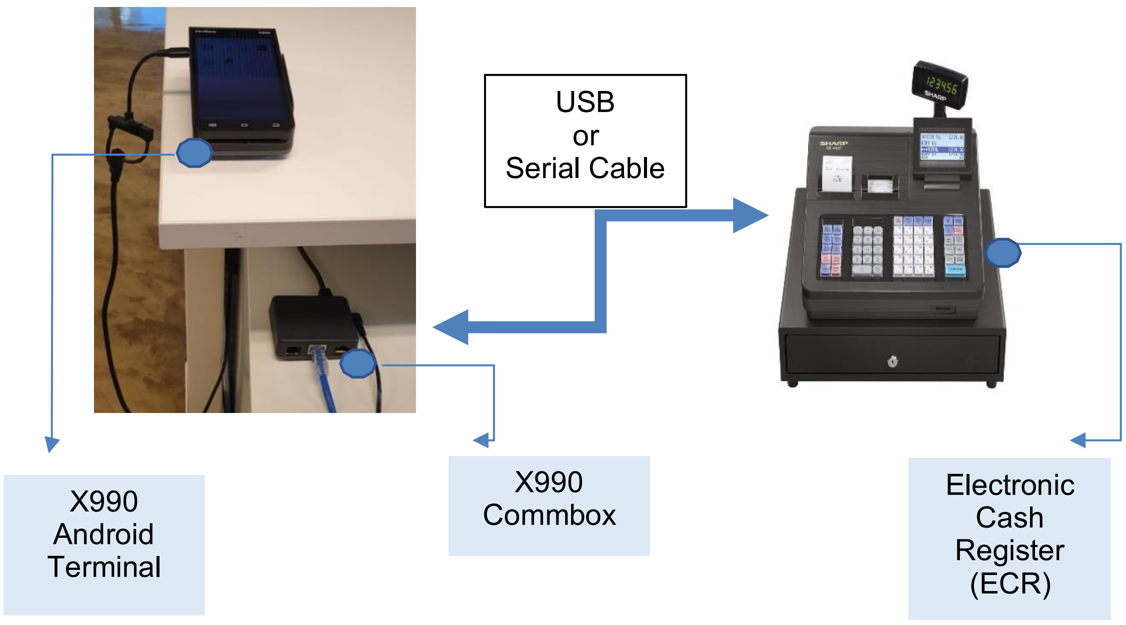
- Verifone X990 Android terminal
- Verifone X990 Commbox (to provide X990 terminal wih RS232 and USB Port)
- ECR device
- USB or RS232 cable.
- Network Router.

And the following software requirement:

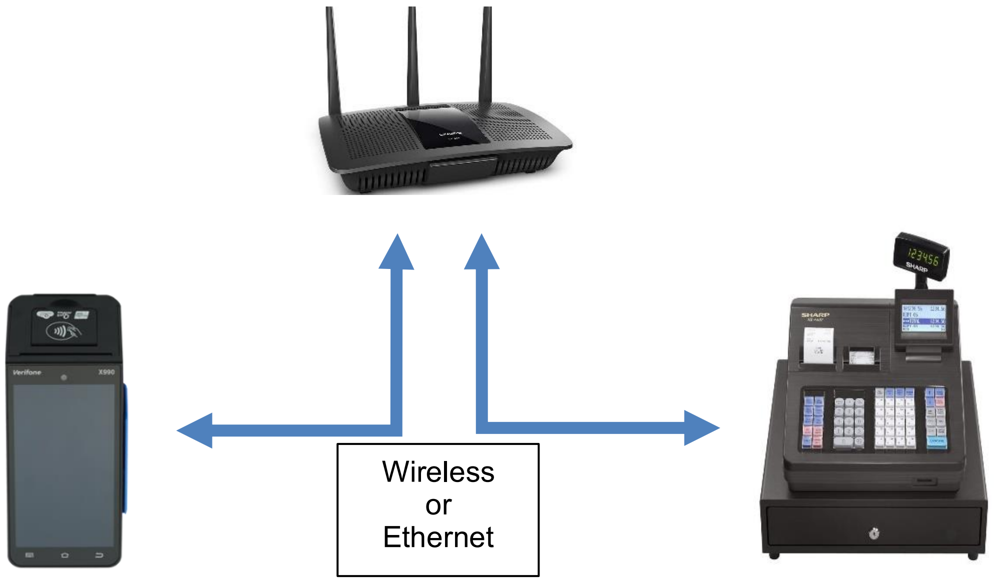
- CIMB application: version 2.1.0-16 or above
- ECR application software that is modified to follow the specification of this document

3.2 Connection Diagram

3.2.1 RS232/USB



3.2.2 Local Network Router



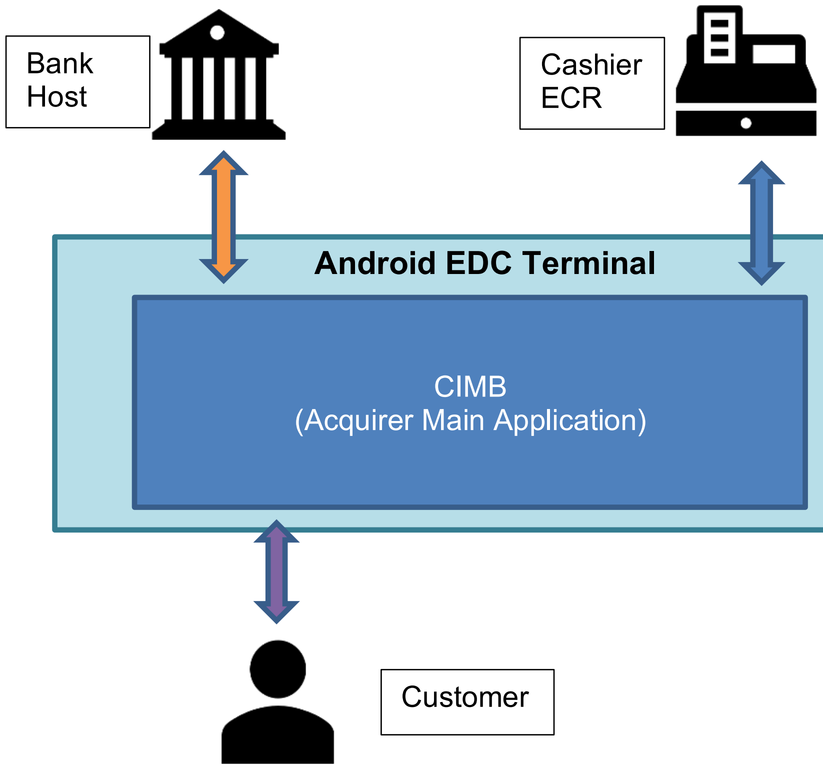
3.3 Physical Interface – Serial/USB

The Physical Interface used between the ECR and the Terminal is detailed in the following table.

Data Rate	9600 bps / 115200 bps (USB/Vx platform)
Connection	RS232C (V.24) Interface OR USB (2.0) Interface Terminal connector is a female DB25 DCE connection On a PC, this is compatible with COM1: or COM2:, and requires a straight-through cable, the same as is required for a modem.
Mode	Terminal port is full duplex
Transmission	Asynchronous, 8 data bits, no parity, 1 stop bit (N,8,1)
Characters	ASCII character set (for character fields)

4 Technical Specification

4.1 Logical Block Diagram



CIMB Application: This is the certified Payment Application from the Acquiring

Bank (CIMB). The application support standalone payment acceptance (non ECR integration) and payment acceptance without Customer entering amount (ECR Integration). Both mode have the same User Interface. Customer can pay using either Debit/Credit Cards or Contactless e-money or QRIS.

Bank Host : existing CIMB acquiring host that will process the payment transaction.

Cashier ECR: the ECR or POS application that is used by merchant/cashier that can be connected to the EDC Terminal via USB or Serial communication.

Customer: the payer/user of the terminal that can do the payment transaction in Standalone Mode or ECR Integration Mode.

When the Customer make a purchase, Cashier ECR will send a request message to the CIMB Application. The application will start and ready for transaction. Customer/Payer or Cashier does not need to enter any transaction amount. If the transaction completed, terminal will send a response to the ECR. ECR will keep the information from the bank host, such as transaction amount, approval code, card holder name and other important information.

4.2 Process Flow - ECR and Terminal

4.2.1 Native Flow

1. Normal Process

The ECR transmits a Request message. The Terminal acknowledges receipt of the message by transmitting a single ACK (06h) character.

The Terminal transmits a Response message. The ECR acknowledges receipt of the message by transmitting a single ACK (06h) character

ECR	Direction	Terminal
Request	→	
	←	ACK
	←	Response Message
ACK	→	

2. Bad LRC

If the ECR or Terminal receives a message in error (Bad Length, missing ETX, or incorrect LRC), the message should be ignored. These errors should only be caused by transmission errors, and the retransmission will correct the error. There is no automatic method for recovering from application errors that cause the message to appear corrupted

ECR	Direction	Terminal
Request	→	
	←	NAK
Request	→	
	←	ACK
	←	Response Message
NAK	→	
	←	Response Message
ACK	→	

3. Time Out

If the ECR or the Terminal sends a message, and does not receive the ACK within 2 second, the message should be transmitted again. If the second transmission does not receive an ACK within 1 second that message should be treated as undeliverable, and the application should take whatever actions are required to recover.

ECR	Direction	Terminal
Request	→	
		No ACK or No NAK within 2s
Request	→	
	←	ACK
	←	Response Message
No ACK or No NAK within 2s		
	←	Response Message
ACK	→	

4. Library

ECR process flow can be done using out provided library (.dll/.so/.aar). It supported for Windows and Linux OS Based and Android Project. Please refer to appendix for header definition to using provided library *(library file distribute separately)*.

4.2.2 Rest API Flow

1. Normal Flow

The ECR transmits a POST Request Transaction message. The Terminal response the message by transmitting POST Response 200 OK with trxId for used to check the result of transaction on process.

The ECR periodically transmits a POST Request Result with trxId message to check result of transaction. The Terminal response the message by transmitting POST Response 503 SERVICE UNAVAILABLE or 200 OK. The last 5 transaction will be saved at Terminal.

ECR	Direction	Terminal
POST Request Trx	→	
	←	POST Response 200 OK with trxId
POST Request Result	→	
	←	POST Response (trx not done) 503 SERVICE UNAVAILABLE
POST Request Result	←	POST Response (trx done) 200 OK

2. Unauthorized

Security type that will be used is basic authentication. If the basic authentication of ECR Rest API is incorrect, the transaction won't be process. The basic authentication will be unique per Terminal.

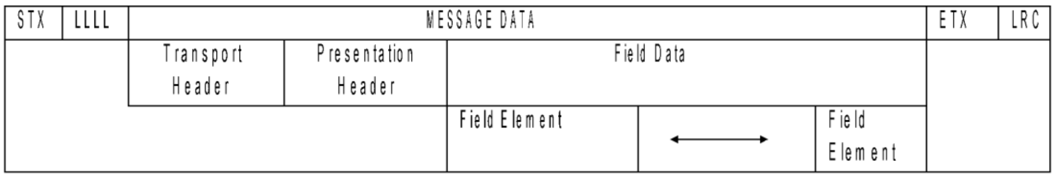
ECR	Direction	Terminal
POST Request Trx	→	
	←	POST Response 401 UNAUTHORIZED

5 Message Specification

5.1 Native

5.1.1 Message Structure

The messages that are transmitted on the link between the ECR and the Terminal will use the following structure.



Field	Bytes	Value	Comment
STX	1	02h	Start of Text This character is used to indicate the start of a frame.
LLLL	2		Length of the MESSAGE DATA to follow. This is transmitted in BCD (Binary Coded Decimal) form. The most significant byte is transmitted first, followed by the least significant byte. For example, a length of 256 bytes will be transmitted as 02h 56h. The LLLL field allows the inclusion of binary data in the message. The maximum allowable value for LLLL will depend on the implementation.
MESSAGE DATA	Variable		The message data consists of a Transport Header, a Presentation Header, and Field Data which is one or more Field Elements. These different components are more fully described in the following sections
ETX	1	03h	End of Text Logically this field is not required because of the length indicator (LLLL), but it is included as an extra check that the message was successfully received and that the receiver is in synchronization with the transmitted message.
LRC	1		Longitudinal Redundancy Character. This character is calculated by Exclusive OR-ing each character following (but not including) the STX up to (and including) the ETX.

The LRC character is the module 2 binary sum of every character in the transaction message after the STX and including the ETX.

The second byte from me.ssage (excluding STX) XOR with the third byte, then the result XOR with the fourth byte and so on until EJX.

Example:

0249 4D 47 03

49XOR4D=04

04XOR47 =43

43XOR03 =40

In addition to the above-described messages, ACK (06H) and NAK (15H) control characters are also required to ensure error free exchange of request and response messages.

An ACK indicates the successful reception of a message, a NAK indicates that the receiver requests the retransmission of the last message that was received in error.

The ACK and NAK characters are expected to be received within 2 seconds from the transmission of a message. Every message is expected to get an ACK or NAK. A message can be sent again (up to 3 times) after the 2 second ACK/NAK response time has expired. In theory, not all request messages generate a response message but they all require to be ACKed.

Filed Name	Format	Length (byte)	Value
Status	h	1	06=ACK/ 15=**NAK**

After ECR send message to EDC, EDC will reply with 06H {ACKt to the ECR ·t the message format is correct, then ECR will continue the t1ransaction process; or EDC will reply with 15H {NAK) to the ECR, if the message format is incorrect, then ECR will stop the transaction process back to idle.

5.1.2 Command Set

Request Message

Field	**Length**	**Type**	**Description**
Trans Type	1	h	Transaction type
Trans Amount	12	n	Transaction Amount (last 2 digits decimal)
Invoice No	6	n	Trace No / Reference Id from original transaction
Card Number	19	n	Transaction Card Number
Filler	162	an	For bank use
TOTAL	**200**		

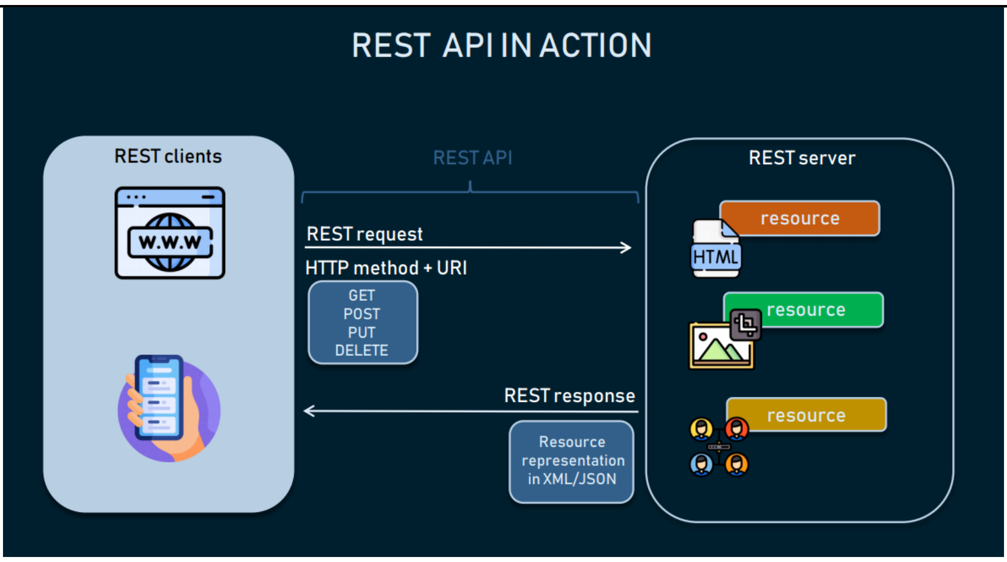
Response Message

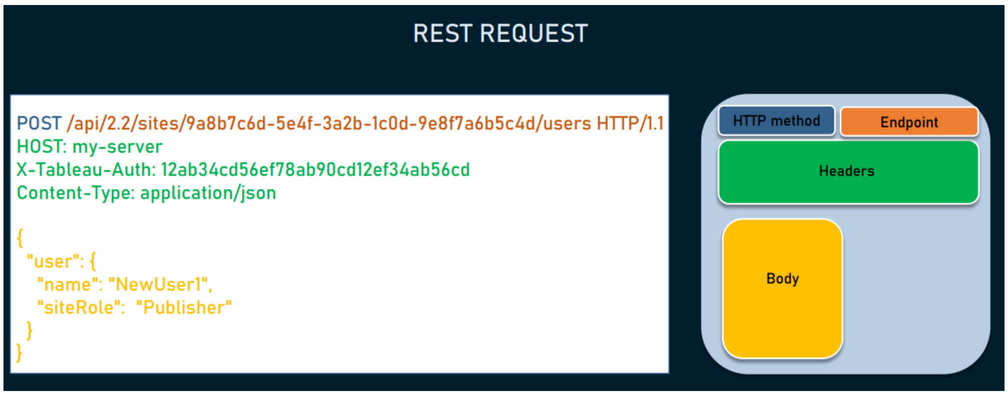
Field	**Length**	**Type**	**Description**
Trans Type	1	h	Transaction type
TID	8	an	Terminal ID
MID	15	an	Merchant ID
Trace No	6	n	
Invoice No	6	n	
Entry Mode	1	an	
Trans Amount	12	n	Last 2 digits decimal
Trans Add Amount	12	n	Last 2 digits decimal
Total Amount	12	n	Last 2 digits decimal
Card No	19	an	Will be masked
Cardholder Name	26	an	
Date	8	n	YYYYMMDD
Time	6	n	HHMMSS
Approval Code	6	an	
Response Code	2	an	
Ref Number	12	an	
Reference Id	6	an	QR Reference Id
Term	2	an	Installment Term
Monthly Amount	12	an	Installment Monthly Pay Amount (last 2 digits decimal)
Point Reward	9	an	Point Reward
Redemption Amount	11	an	Point Reward Redemption Amount
Point Balance	9	an	Point Reward Balance After Transaction
Filler	99	an	For Bank Use
TOTAL	300		

5.2 Rest API

5.2.1 Message Structure

The rest api that will be used is not like web service usually have, even the data and protocol using the same model. Method that will be allowed only POST and the security is using Basic Authentication. There will be username and password used to generate basic auth data header, and the password will be unique per Terminal.





```
POST /api/2.2/sites/9a8b7c6d-5e4f-3a2b-1c0d-9e8f7a6b5c4d/users HTTP/1.1
HOST: my-server
X-Tableau-Auth: 12ab34cd56ef78ab90cd12ef34ab56cd
Content-Type: application/json

{
 "user": {
   "name": "NewUser1",
   "siteRole":  "Publisher"
 }
}
```

5.2.2 Command Set

Transaction	
Method	POST
URL Format	/transaction/cimb
Request Header	**Description**
Authorization	Basic Auth

	Username: VfiF4CIMB Password: VFI + (SN Terminal)	
Content-Type	application/json	
Request Body (JSON)	**Type**	**Description**
transType	String	Transaction type
transAmount	String	Transaction Amount
invoiceNo	String	Trace No / Reference Id from original transaction
cardNumber	String	Transaction Card Number
Response Header	**Description**	
None		
Response Body (JSON)	**Type**	**Description**
trxId	String	Id of on progress transaction
Status Code	**Description**	
200	OK. The request was successfully processed	
400	BAD REQUEST. The type of content is invalid	
401	UNAUTHORIZED. The username or password are incorrect or have not been passed	
500	INTERNAL SERVER ERROR. Something wrong happens in Terminal	
503	SERVICE UNAVAILABLE. Terminal is busy	

Result

Method	POST	
URL Format	/result/cimb	
Request Header	**Description**	
Authorization	Basic Auth Username: VfiF4CIMB Password: VFI + (SN Terminal)	
Content-Type	application/json	
Request Body (JSON)	**Type**	**Description**
trxId	String	Id of on progress transaction
Response Header	**Description**	
None		
Response Body (JSON)	**Type**	**Description**
transType	String	Transaction type
tid	String	Terminal ID
mid	String	Merchant ID
traceNo	String	
invoiceNo	String	
entryMode	String	
transAmount	String	
transAddAmount	String	
totalAmount	String	
cardNo	String	Will be masked

cardholderName	String	
date	String	YYYYMMDD
time	String	HHMMSS
approvalCode	String	
responseCode	String	
refNumber	String	
referenceId	String	QR Reference Id
term	String	Installment Term
monthlyAmount	String	Installment Monthly Pay Amount
pointReward	String	Point Reward
redemptionAmount	String	Point Reward Redemption Amount
pointBalance	String	Point Reward Balance After Transaction
filler	String	For Bank Use

Status Code	**Description**
200	OK. The request was successfully processed
400	BAD REQUEST. The type of content is invalid
401	UNAUTHORIZED. The username or password are incorrect or have not been passed
404	NOT FOUND. Transaction can't be found
500	INTERNAL SERVER ERROR. Something wrong happens in Terminal
503	SERVICE UNAVAILABLE. Terminal is busy

5.3 Transaction Type

This transaction type depends on the CIMB application. Here with is the list of the Transaction Type.

Transaction Type		Description
Native	Rest API	
0x01	01	Sale
0x02	02	Installment
0x03	03	Void
0x04	04	Refund
0x05	05	QRIS MPM
0x06	06	QRIS Notification
0x07	07	QRIS Refund
0x08	08	Point Reward
0x09	09	Test Host
0x0A	0A	QRIS CPM
0x0B	0B	Settlement
0x0C	0C	Reprint
0x0D	0D	Report
0x0E	0E	Logon

5.4 Entry Mode

Entry Mode		Description
Native	Rest API	
0x44	D	D: Dip
0x53	S	S: Swipe
0x46	F	F: Fallback
0x4D	M	M: Manual
0x43	T	C: Contactless
0x60	`	QRIS MPM/CPM

5.5 Sample Command for Request

5.5.1 Native

020200013030303030303031323330303030303030303020202020202020202020202
020202020202020202020202020202020202020202020202020202020202020202020
202020202020202020202020202020202020202020202020202020202020202020202
020202020202020202020202020202020202020202020202020202020202020202020
202020202020202020202020202020202020202020202020202020202020202020202
0202020202020202020202020202020202020202020202020202020202020202020322

Value	Lable
02	STX
0200	Length
0130303030303030313233303030303030303020 2020202020202020202020202020202020202020 2020202020202020202020202020202020202020 2020202020202020202020202020202020202020 2020202020202020202020202020202020202020 2020202020202020202020202020202020202020 2020202020202020202020202020202020202020 2020202020202020202020202020202020202020 2020202020202020202020202020202020202020 2020202020202020202020202020202020202020	Message Data
03	ETX
22	LRC

5.5.2 Rest API

```
POST /transaction/cimb HTTP/1.1
Host: 192.168.0.101:9001
Authorization: Basic VmZpRjRDSU1COlZGSVYxRTAyMTI2Mzk=
Content-Type: application/json
Content-Length: 93

{
    "transType":"01",
    "transAmount":"0",
    "invoiceNo":"",
    "cardNumber":""
}
```

5.6 Sample Command for Response

5.6.1 Native

0203000F3130303036323031303030303031303033333430303030303030303031564
9534100000000000000000000003030303337333030303538334430303030303030
3033303030303030303030303030353030303030303030303030383030343833353737342
A2A2A2A2A2A3534373700000053494E4747494841444849 4D414E544F524F2F00
00000000000032303231313232333233323233333202032313335353230303830323
7353738363130330000000000000000000000000000000000000000000000000000
0000000417070726F766564000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000351

Value	Lable
02	STX
0300	Length
0141524953303030363030303030383737303030 3030303130303034303230303030303344303030 3030303031323330303030303030303030303030 303030303030303031323330303335363533362A 2A2A2A2A2A30303934000000544D442041524946 204A434C3036000000000000000000000000003230 3231313232383130333332383133303936363030 3430363531313535313132300000000000000000 0000000000000000000000000000000000000000 0000000000000000000000000000000000000000 0054524154E53414354494F4E2053554343455353 0000000000000000000000000000000000000000 0000000000000000000000000000000000000000 0000000000000000000000000000000000000000 0000000000000000000000000000000000000000	Message Data
03	ETX
51	LRC

5.6.2 Rest API

```
HTTP/1.1 200 OK
Server: ECR/4.0.0
Date: Fri, 08 Jul 2022 08:42:06 GMT+00:00
Cache-Control: no-store
Content-Type: application/json
Pragma: no-cache
Connection: close
```

```
Content-Length: 625

{
    "transType": "01",
    "tid": "10006200",
    "mid": "000001003340000",
    "traceNo": "000031",
    "invoiceNo": "000061",
    "entryMode": "D",
    "transAmount": "125",
    "transAddAmount": "0",
    "totalAmount": "125",
    "cardNo": "518856******3707",
    "cardholderName": "TCMASTER07",
    "date": "20220708",
    "time": "164144",
    "approvalCode": "508236",
    "responseCode": "00",
    "refNumber": "931908552922",
    "referenceId": "",
    "term": "",
    "monthlyAmount": "",
    "pointReward": "",
    "redemptionAmount": "",
    "pointBalance": "",
    "filler": "TRANSACTION SUCCESS |Bank CIMB Niaga"
}
```

Appendix A Windows Library Header

```
#pragma once

extern "C" {
          /**
          * @brief Get information of dll library version
          * @param[out] szVersion : version data (lenght 6-13 characters)
          * @return NULL
          */
          __declspec(dllexport) void ecrGetVersion(char* szVersion);

          /**
          * @brief Open socket communication
          * @param[in] szIp : Destionation ip address
          * @param[in] inPort : Destination port
          * @param[in] isSsl : Secure connection flag
          *         NON SSL             0 (default)
          *         SSL                           1
          * @return
          *   0 : Success
          *  -1 : Internal error
          *  -2 : Failed to initialize
          *  -3 : Failed to connect
          *  -4 : Failed to set communication option
          */
          __declspec(dllexport) int ecrOpenSocket(char* szIp, int inPort, int isSsl);

          /**
          * @brief Send data to socket communication
          * @param[in] szData : Buffer data to be send
          * @param[in] inLen : Total data to be send
          * @return
          *   0 : Success
          *  -1 : Internal error
          *  -2 : Serial port not opened
          *  -3 : Data sent not complete
          */
          __declspec(dllexport) int ecrSendSocket(unsigned char* szData, unsigned int inLen);

          /**
          * @brief Receive data from socket communication
          * @param[out] szData : Buffer data to receive
          * @param[in] inSize : Size of buffer data to receive
          * @return
          * var : Data received
          *  -1 : Internal error
          *  -2 : Serial port not opened
          *  -3 : Invalid length
          *  -4 : Invalid lrc
          */
          __declspec(dllexport) int ecrRecvSocket(unsigned char* szData, unsigned int inSize);

          /**
          * @brief Close socket communication
          */
          __declspec(dllexport) void ecrCloseSocket(void);

          /**
          * @brief Serial communication data
          * @param[in] chBaudRate :
          *         1200                0
          *         2400                1
          *         4800                2
          *         9600                3 (default)
          *         14400               4
          *         19200               5
          *         38400               6
          *         57600               7
          *         115200              8
          *         128000              9
          *         256000              10
          * @param[in] chStopBit :
          *         ONESTOPBIT          0 (default)
          *         ONE5STOPBITS        1
          *         TWOSTOPBITS         2
          * @param[in] chParity :
          *         NOPARITY            0 (default)
          *         ODDPARITY           1
          *         EVENPARITY          2
          *         MARKPARITY          3
          *         SPACEPARITY         4
          */
          struct SerialData {
                    char szComm[10];                          /*Serial communication port number*/
                    unsigned char chBaudRate;      /*Serial communication baudrate*/
                    unsigned char chDataBit;       /*Serial communication data length*/
                    unsigned char chStopBit;       /*Serial communication stop bit*/
                    unsigned char chParity;                   /*Serial communication parity bit*/
          };

          /**
          * @brief Open serial communication port
          * @return
          *    0 : Success
          *   -1 : Internal error
          *   -2 : Failed to flush data
          *   -3 : Failed to set communication timeout
          *   -4 : Failed to set communication option
          */
          __declspec(dllexport) int ecrOpenSerialPort(struct SerialData* srSerialData);

          /**
          * @brief Send data to serial communication
          * @param[in] szData : Buffer data to be send
          * @param[in] inLen : Total data to be send
          * @return
          *    0 : Success
          *   -1 : Internal error
          *   -2 : Serial port not opened
          *   -3 : Data sent not complete
          */
          __declspec(dllexport) int ecrSendSerialPort(unsigned char* szData, unsigned int inLen);

          /**
          * @brief Receive data from serial communication
          * @param[out] szData : Buffer data to receive
          * @param[in] inSize : Size of buffer data to receive
          * @return
          * var : Data received
          *   -1 : Internal error
          *   -2 : Serial port not opened
          *   -3 : Invalid length
          *   -4 : Invalid lrc
          */
          __declspec(dllexport) int ecrRecvSerialPort(unsigned char* szData, unsigned int inSize);

          /**
          * @brief Close serial communication port
          */
          __declspec(dllexport) void ecrCloseSerialPort(void);

          /**
          * @brief Data structure for request transaction
          * @note Refer to documentation
          */
          struct ReqData {
                    unsigned char chTransType;
                    char szAmount[12];
                    char szAddAmount[12];
                    char szInvNo[12];
                    char szCardNo[19];
          };

          /**
          * @brief Pack message to be send for request transaction
          * @param[out] szReqMsg : Buffer raw request message
          * @return
          * var : Raw request message length
          *   -1 : Invalid parameters
          */
          __declspec(dllexport) int ecrPackRequest(unsigned char* szReqMsg, struct ReqData* srReqData);

          /**
          * @brief Data structure for response transaction
          * @note Refer to documentation
          */
          struct RspData {
                    unsigned char chTransType;
                    char szTID[8];
                    char szMID[15];
                    char szBatchNumber[6];
                    char szIssuerName[25];
                    char szTraceNo[6];
```

```
                    char szInvoiceNo[6];
                    unsigned char chEntryMode;
                    char szTransAmount[12];
                    char szTransAddAmount[12];
                    char szTotalAmount[12];
                    char szCardNo[19];
                    char szCardholderName[26];
                    char szDate[8];
                    char szTime[6];
                    char szApprovalCode[8];
                    char szResponseCode[2];
                    char szRefNumber[12];
                    char szBalancePrepaid[12];
                    char szTopupCardNo[19];
                    char szFiller[84];
          };

          /**
          * @brief Parse message from receive for response transaction
          * @param[in] szReqMsg : Buffer raw response message
          * @return
          *    0 : Success
          *   -1 : Invalid length
          *   -2 : Invalid lrc
          */
          __declspec(dllexport) int ecrParseResponse(unsigned char* szRspMsg, struct RspData* srRspData);
}
```

Appendix B Linux Library Header

```
#ifndef __EcrLibrary_H__
#define __EcrLibrary_H__

/**
* @brief Get information of dll library version
* @param[out] szVersion : version data (lenght 6-13 characters)
* @return NULL
*/
extern void ecrGetVersion(char* szVersion);

/**
* @brief Open socket communication
* @param[in] szIp : Destionation ip address
* @param[in] inPort : Destination port
* @param[in] isSsl : Secure connection flag
*         NON SSL             0 (default)
*         SSL                           1
* @return
*    0 : Success
*   -1 : Internal error
*   -2 : Failed to initialize
*   -3 : Failed to connect
*   -4 : Failed to set communication option
*/
extern int ecrOpenSocket(char* szIp, int inPort, int isSsl);

/**
* @brief Send data to socket communication
* @param[in] szData : Buffer data to be send
* @param[in] inLen : Total data to be send
* @return
*    0 : Success
*   -1 : Internal error
*   -2 : Serial port not opened
*   -3 : Data sent not complete
*/
extern int ecrSendSocket(unsigned char* szData, unsigned int inLen);

/**
* @brief Receive data from socket communication
* @param[out] szData : Buffer data to receive
* @param[in] inSize : Size of buffer data to receive
* @return
* var : Data received
*   -1 : Internal error
*   -2 : Serial port not opened
*   -3 : Invalid length
*   -4 : Invalid lrc
*/
extern int ecrRecvSocket(unsigned char* szData, unsigned int inSize);

/**
* @brief Close socket communication
*/
extern void ecrCloseSocket(void);

/**
* @brief Serial communication data
* @param[in] chBaudRate :
*         1200                0
*         2400                1
*         4800                2
*         9600                3 (default)
*         14400               4
*         19200               5
*         38400               6
*         57600               7
*         115200              8
*         128000              9
*         256000              10
* @param[in] chStopBit :
*         ONESTOPBIT          0 (default)
*         ONE5STOPBITS        1
*         TWOSTOPBITS         2
* @param[in] chParity :
*         NOPARITY            0 (default)
*         ODDPARITY           1
*         EVENPARITY          2
*         MARKPARITY          3
*         SPACEPARITY         4
*/
struct SerialData {
          char szComm[10];                        /*Serial communication port number*/
          unsigned char chBaudRate;     /*Serial communication baudrate*/
          unsigned char chDataBit;      /*Serial communication data length*/
          unsigned char chStopBit;      /*Serial communication stop bit*/
          unsigned char chParity;                 /*Serial communication parity bit*/
};

/**
* @brief Open serial communication port
* @return
*   0 : Success
*  -1 : Internal error
*  -2 : Failed to flush data
*  -3 : Failed to set communication timeout
*  -4 : Failed to set communication option
*/
extern int ecrOpenSerialPort(struct SerialData* srSerialData);

/**
* @brief Send data to serial communication
* @param[in] szData : Buffer data to be send
* @param[in] inLen : Total data to be send
* @return
*   0 : Success
*  -1 : Internal error
*  -2 : Serial port not opened
*  -3 : Data sent not complete
*/
extern int ecrSendSerialPort(unsigned char* szData, unsigned int inLen);

/**
* @brief Receive data from serial communication
* @param[out] szData : Buffer data to receive
* @param[in] inSize : Size of buffer data to receive
* @return
* var : Data received
*  -1 : Internal error
*  -2 : Serial port not opened
*  -3 : Invalid length
*  -4 : Invalid lrc
*/
extern int ecrRecvSerialPort(unsigned char* szData, unsigned int inSize);

/**
* @brief Close serial communication port
*/
extern void ecrCloseSerialPort(void);

/**
* @brief Data structure for request transaction
* @note Refer to documentation
*/
struct ReqData {
          unsigned char chTransType;
          char szAmount[12];
          char szAddAmount[12];
          char szInvNo[12];
          char szCardNo[19];
};

/**
* @brief Pack message to be send for request transaction
* @param[out] szReqMsg : Buffer raw request message
* @return
* var : Raw request message length
*  -1 : Invalid parameters
*/
extern int ecrPackRequest(unsigned char* szReqMsg, struct ReqData* srReqData);

/**
* @brief Data structure for response transaction
* @note Refer to documentation
*/
struct RspData {
          unsigned char chTransType;
          char szTID[8];
          char szMID[15];
          char szBatchNumber[6];
          char szIssuerName[25];
          char szTraceNo[6];
          char szInvoiceNo[6];
          unsigned char chEntryMode;
          char szTransAmount[12];
          char szTransAddAmount[12];
          char szTotalAmount[12];
          char szCardNo[19];
          char szCardholderName[26];
          char szDate[8];
          char szTime[6];
          char szApprovalCode[8];
          char szResponseCode[2];
          char szRefNumber[12];
          char szBalancePrepaid[12];
          char szTopupCardNo[19];
          char szFiller[84];
};

/**
* @brief Parse message from receive for response transaction
* @param[in] szReqMsg : Buffer raw response message
* @return
*    0 : Success
*   -1 : Invalid length
*   -2 : Invalid lrc
*/
extern int ecrParseResponse(unsigned char* szRspMsg, struct RspData* srRspData);

#endif
```

Appendix C Android Library Class

```
Class CimbEcrLib(activity: Activity)

implementation files('libs/ecr-lib-release.aar')
implementation files('libs/cimb-ecr-lib-release.aar')
```

CimbEcrLib is designed to enable proper and easy way to communication between applications and ecr terminal.

SUMMARY - Public methods

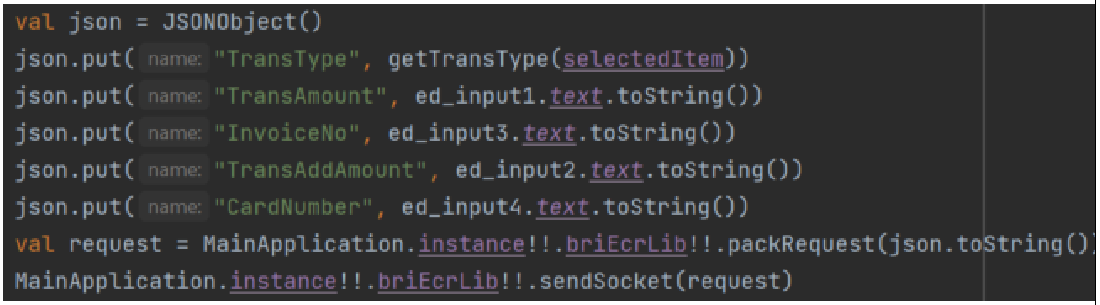
Modifier and Type	Method and Description
String	**getVersion()** Return version number of library.
ByteArray	**packRequest(reqMsg: String)** Used to pack message before send data to ecr terminal.
String?	**parseResponse(rspMsg: ByteArray)** Used to parse message after receive data from ecr terminal.
String	**getMessage()** Returns the status of the connection including parse process.
Boolean	**isConnected()** Returns the connection state of the socket or serial uart.
Boolean	**openSocket(ip: String, port: Int, ssl: Boolean)** Connects socket to specified port number on the named ip.
Boolean	**sendSocket(message: ByteArray)** Sends message to socket output stream.
ByteArray	**recvSocket()** Reads message from socket input stream according to specified format.

Unit	**closeSocket()** Closed the communication of socket.
Boolean	**openSerialPort(baudRate: Int, dataBits: Int, stopBits: Int, parity: Int)** Connects serial uart to specified serial device with serial settings.
Boolean	**sendSerialPort(message: ByteArray)** Sends message to serial uart output stream.
ByteArray	**recvSerialPort()** Reads message from serial uart input stream according to specified format.
Unit	**closeSerialPort()** Closed the communication of serial uart.

DETAILS - Public methods

getVersion
getVersion(): String get version of used library. **Returns:** string of library version
packRequest
packRequest(reqMsg: String): ByteArray pack the message to be sent to the ecr terminal. Please check section 5.1.2 Command Set - Request Message. **Parameters:** reqMsg - json string of request message **Returns:** empty ByteArray if invalid parameters

Snipped:



```
val json = JSONObject()
json.put( name: "TransType", getTransType(selectedItem))
json.put( name: "TransAmount", ed_input1.text.toString())
json.put( name: "InvoiceNo", ed_input3.text.toString())
json.put( name: "TransAddAmount", ed_input2.text.toString())
json.put( name: "CardNumber", ed_input4.text.toString())
val request = MainApplication.instance!!.briEcrLib!!.packRequest(json.toString()
MainApplication.instance!!.briEcrLib!!.sendSocket(request)
```

parseResponse

`parseResponse(rspMsg: ByteArray): String?`

parse the message receive from the ecr terminal. Please check section 5.1.2 Command Set – Response Message.

Parameters:
rspMsg – return value from recvSocket method

Returns:
json string if successful parsing
null if invalid parameters, detail error get using getMessage method

getMessage

`getMessage(): String`

returns the status of the connection including parse process. All process that generates status will use this to get human readable status.

Returns:
empty string if no status available

isConnected

`isConnected(): Boolean`

get the state of socket or serial uart connection.

Returns:
true – connection establish
false – disconnected

openSocket

openSocket(ip: String, port: Int, ssl: Boolean): Boolean

establish socket connection to ecr terminal.

Parameters:
ip – ecr terminal ip address
port – ecr terminal port number
ssl – true if secure connection, false if not secure

Returns:
true – connection establish successful
false – failed to establish connection

sendSocket

sendSocket(message: ByteArray): Boolean

send message from pack process to the ecr terminal through socket.

Parameters:
message – return value from packRequest method

Returns:
true – data sent successful
false – data failed to send

recvSocket

recvSocket(): ByteArray

receive data from ecr terminal through socket.

Returns:
empty ByteArray if no data received
when no data receive and getMessage method return "Connection is closed" that mean connection is closed by host

closeSocket

closeSocket()

close socket and also close associated streams from the socket.
openSerialPort
openSerialPort(baudRate: Int, dataBits: Int, stopBits: Int, parity: Int): Boolean establish serial uart connection to ecr terminal. **Parameters:** baudRate – ecr terminal serial uart speed • 1200 • 2400 • 4800 • 9600 • 14400 • 19200 • 38400 • 57600 • 115200 • 128000 dataBits – ecr terminal serial uart data bit • 6 • 7 • 8 • 9 stopBits – ecr terminal serial uart stop bit • STOP_BITS_1 • STOP_BITS_2 • STOP_BITS_1_5 parity – ecr terminal serial uart parity bit • PARITY_NONE • PARITY_ODD • PARITY_EVEN • PARITY_MARK • PARITY_SPACE **Returns:**

true – connection establish successful false – failed to establish connection
sendSerialPort
sendSerialPort(message: ByteArray): Boolean send message from pack process to the ecr terminal through serial uart. **Parameters:** message – return value from packRequest method **Returns:** true – data sent successful false – data failed to send
recvSerialPort
recvSerialPort(): ByteArray receive data from ecr terminal through serial uart. **Returns:** empty ByteArray if no data received
closeSerialPort
closeSerialPort() close serial uart connection.