PM500
EMV Kernel SDK Guide

Pointmobile Co., Ltd.
S/W R&D Team

PM500 EMV Kernel SDK Guide

Revision History

Revision	Date	Change Description	Author
V01	2020.08.25	Initial draft	Emma Jung

Table of Contents

1. Abstract

Introduction

This guide describes details of EMV kernel module API function for Android terminal. API function includes how to use below interfaces.

- Terminal and EMV kernel module parameters access interface
- CAPK management interface
- EMV App-List management interface
- App-EMV KERNEL callback function interface
- EMV App processing interface
  (Such as creating App list, reading App data, card data authentication, and online processing, etc.)

Some additional EMV related processing must be done in EMV / PBOC application, such as setting Terminal Capabilities, setting terminal capabilities, setting additional terminal capabilities, query and setting AID list, TAC and floor limit, query CAPK list and setting mode of communication to the host.

References

1) EMV v4.1 Book 1 ICC to Terminal Interface
2) EMV v4.1 Book 2 Security and Key Management
3) EMV v4.1 Book 3 Application Specification
4) EMV v4.1 Book 4 Other Interfaces

2. Application Programming Interfaces

Initialization Function

- **Functions**

1) Initialize EMV Library

```
public static native int EmvLib_Init();
```

2) Clear EMV Transaction Log

```
public static native void EmvLib_ClearTransLog();
```

3) Set Terminal Parameters

```
public static native int EmvLib_SetParam(EMV_PARAM tParam);
```

4) Set TLV Parameters

```
public static native int EmvLib_SetTLV(String Tag, byte[] DataIn,
int DataLen);
```

5) Add or Update CAPK

```
public static native int EmvLib_AddCapk(EMVCAPK capk);
```

6) Delete CAPK

```
public static native int EmvLib_DelCapk(byte KeyID, byte[] RID);
```

7) Add or Update Application List

```
public static native int EmvLib_AddApp(EMV_APPLIST App);
```

8) Add or Update Application List by Index

```
int EmvLib_AddAppByIndex(int Index, EMV_APPLIST App);
```

9) Delete Application List

```
public static native int EmvLib_DelApp(byte[] AID, int AIDLen);
```

10) Set Application File Path

```
public static native void EmvLib_SetFilePath(String path);
```

11) Remove the EMV Kernel Parameter File

```
int EmvLib_RemoveFile(byte FileIndex);
```

1) Initialize EMV Library

Function	`int EmvLib_Init();`
Parameter	void
Return Value	Refer to ERROR CODE
Note	Call it once in function MAIN() before all other EMV related processing

2) Clear EMV Transaction Log

EMV library module will write log to the file to save some latest transaction PAN and amount, at step “terminal risk management” in transaction flow, to process the total amount and floor limit. Call this function to delete those logs.

Function	`void EmvLib_ClearTransLog();`
Parameter	void
Return Value	Refer to ERROR CODE
Note	In practice, this function can be called after batch settlement.

3) Set Terminal Parameters

Set terminal parameters stored in EMV library module.

Function	`int EmvLib_SetParam(byte[] tParam);`
Parameter	tParam (byte[]) [in] – Pointer to data of EMV terminal parameter. Refer to EMV_PARAM
Return Value	Refer to ERROR CODE

4) Set TLV Parameters

Set and save a TLV element in EMV library.

Function	`int EmvLib_SetTLV(String Tag,`
	`byte[] DataIn,`
	`int DataLen);`

Parameter	Tag (String) [in] – Tag of the TLV to save
	DataIn (byte[]) [in] – Pointer to value of the TLV to save
	DataLen (int) [in] – Length of the TLV to save
Return Value	Refer to ERROR CODE

5) Add o Update CAPK

Add or update CAPK data in EMV library.

Function	`int EmvLib_AddCapk(EMVCAPK capk);`
Parameter	capk (EMVCAPK) [in] – Pointer to EMV_CAPK structure data
Return Value	Refer to ERROR CODE

6) Delete CAPK

Get the PCI version information of secure module and App.

Function	`int EmvLib_DelCapk(byte KeyID,` `byte[] RID);`
Parameter	KeyID (byte) [in] – Key index of CAPK to delete
	RID (byte[]) [in] – Pointer to RID of the CAPK to delete
Return Value	Refer to ERROR CODE

7) Add or Update Application List

Add or update application list.

Function	`int EmvLib_AddApp(EMV_APPLIST App);`
Parameter	App (EMV_APPLIST) [in] – Pointer to the application list item to add
Return Value	Refer to ERROR CODE
Note	EMV library can save at most 32 application list items. If more than 32 items be saved, it will return ERR_OVERFLOW.

8) Add or Update Application List by Index

Function	`int EmvLib_AddAppByIndex(int Index,` `EMV_APPLIST App);`
Parameter	Index (int) [in] – The location which the application list item will be saved
	App (EMV_APPLIST) [in] – Pointer to an application list item to add
Return Value	Refer to ERROR CODE
Note	EMV library can save at most 32 application list items. If more than 32 items be saved, it will return ERR_OVERFLOW.

9) Delete Application List

Delete one application list item.

Function	`int EmvLib_DelApp(byte[] AID,` `int AidLen);`
Parameter	AID (byte[]) [in] – Pointer to AID of App-list item to delete
	AIDLen (int) [in] – Length of AID
Return Value	Refer to ERROR CODE

10) Set Application File Path

EMV core use this function set the parameter file save path.

Function	`void EmvLib_SetFilePath(String path);`
Parameter	path (String) [in] – emv file save path
Return Value	Refer to ERROR CODE
Note	Call it before initialize emv library

11) Remove the EMV Kernel Parameter File

EMV core use this function to delete the parameter file.

Function	`int EmvLib_RemoveFile(byte FileIndex);`
Parameter	FileIndex (byte) [in] – The index of parameter file 0x01: Exception file 0x02: Trans log file 0x03: APP list file (EMV_APPLIST) 0x04: Terminal Parameter file (EMV_PARAM) 0x05: CAPK file (EMVCAPK) 0x07: The file of revocation CAPK 0x08: App program ID
Return Value	Refer to ERROR CODE
Note	If FileIndex=0x04, all the parameter data will be deleted.

EMV Application Trade Flow Interface

- Functions

1) Transaction Initialize

```
int EmvLib_TransInit();
```

2) Set trance amount and date time

```
public static native int EmvLib_BeforeTrans(long Amount,long
BackAmt, byte[] Tdate, byte[] Ttime);
```

3) Select Application In the App-List

```
public static native int EmvLib_AppSel(int slot, long TransNo);
```

4) Process Transaction before online

```
public static native int EmvLib_ProcTransBeforeOnline(byte[]
bIfGoOnline);
```

5) Online transaction complete

```
public static native int EmvLib_ProcTransComplete(byte
Result,byte[] RspCode,byte[] AuthCode,  int AuthCodeLen, byte[]
IAuthData, int IAuthDataLen, byte[] Script, int ScriptLen);
```

1) Transaction Initialize

Initialize the temporary variable before each trade.

Function	`int EmvLib_TransInit();`
Parameter	void
Return Value	Refer to ERROR CODE
Note	It will be called before each trade.

2) Set Trance Amount and Datetime

EMV core set trance amount and datetime

Function	`int EmvLib_BeforeTrans(long Amount,`
	`long BackAmt,`
	`byte[] Tdate,`
	`byte[] Ttime);`

Parameter	Amount (long) [in] – Authorized amount
	BackAmt (long) [in] – Other amount
	Tdate (byte[]) [in] – Trance date
	Ttime (byte[]) [in] – Trance time
Return Value	Refer to ERROR CODE

3) Select Application In The App-List

EMV transaction processing including select application, GPO, read application data.

Function	`int EmvLib_AppSel(int Slot,` `long TransNo);`
Parameter	Slot (int) – Card slot. 0: Contact card 1: Contactless card
	TransNo (long) – Transaction number
Return Value	Refer to ERROR CODE
Note	After create applist, the function should be called to select application.

4) Process Transaction before Online

EMV transaction processing including Offline data authentication, Terminal risk management, Cardholder verification, Terminal action analysis, Card action analysis and the first GAC.

Function	`int EmvLib_ProcTransBeforeOnline(byte[] bIfGoOnline);`
Parameter	bIfGoOnline (byte[]) [out] – Transaction whether need online
Return Value	Refer to ERROR CODE
Note	If the value of `bIfGoOnline` is 1, you must call the complete interface `EmvLib_ProcTransComplete`.

5) Online Transaction Complete

Process transaction to complete after online process

Function	`int EmvLib_ProcTransComplete(byte Result,`
	`byte[] RspCode,`
	`byte[] AuthCode,`
	`int AuthCodeLen,`
	`byte[] IAuthData,`
	`int IAuthDataLen,`
	`byte[] Script,`
	`int ScriptLen);`

Parameter	Result (byte) – Online result
	RspCode (byte[]) – Response code
	AuthCode (byte[]) – Authorized code
	AuthCodeLen (int) – Authorized code length
	IAuthData (byte[]) – Authentication data
	IAuthDataLen (int) – Authentication data length
	Script (byte[]) – Issuer script
	ScriptLen (int) – Issuer script length
Return Value	Refer to ERROR CODE

EMV Core Get Data

- **Functions**

1) Get EMV Core Version

```
public static native int EmvLib_GetVer();
```

2) Get EMV Process Path

```
public static native byte EmvLib_GetPath();
```

3) Get e_cash Balance

```
public static native int EmvLib_GetBalance(byte[] BcdBalance);
```

4) Get TLV Parameters

```
public static native int EmvLib_GetTLV(String Tag, byte[]
DataOut, int[] outLen);
```

5) Get Terminal Parameter

```
public static native int EmvLib_GetParam(EMV_PARAM tParam);
```

6) Get CAPK

```
public static native int EmvLib_GetCapk(int Index, EMVCAPK
capk);
```

7) Get Application List

```
public static native int EmvLib_GetApp(int Index, EMV_APPLIST
App);
```

8) Get Issuer Script Result

```
public static native int EmvLib_GetScriptResult(byte[] Result,
int[] RetLen);
```

9) Get The Flag of Signature

```
public static native int EmvLib_GetPrintReceiptFlag();
```

10) Get The Flag of Advice

```
public static native int EmvLib_GetAdviceReqFlag();
```

11) Application selection for reading log:

```
public static native int EmvLib_AppSelForLog(int slot);
```

12) Read transaction log

```
public static native int EmvLib_ReadLogRecord(int RecordNo);
```

13) Get the data of transaction log

```
public static native int EmvLib_GetLogItem(String Tag, byte[]
DataOut, int[] outLen);
```

14) Read load log

```
public static native int EmvLib_ReadLoadLogRecord(int RecordNo);
```

15) Get common data of load log

```
public static native int EmvLib_GetLoadLogCommon(byte[] Data,
int[] DataLen);
```

16) Get data from load log

```
public static native int EmvLib_GetLoadLogItem(String Tag,byte[]
TagData, int[] TagLen);
```

1) Get EMV Core Version

Function	`int EmvLib_GetVer();`
Parameter	void
Return Value	Integer version number. (2601 represents as 2.6.0.1)

2) Get EMV Process Path

EMV core get card process path.

Function	`byte EmvLib_GetPath();`
Parameter	void
Return Value	0: PBOC 1: qPBOC 3: qVSDC 5: MChip 6: Mag Stripe

3) Get e_cash Balance

Function	`int EmvLib_GetBalance(byte[] BcdBalance);`
Parameter	BcdBalance (byte[]) – Return Amt of Balance with BCD format
Return Value	Refer to ERROR CODE

4) Get TLV Parameter

Find an TLV element by parameter `Tag` in EMV library.

Function	`int EmvLib_GetTLV(String Tag,`
	`byte[] DataOut,`
	`int[] OutLen);`

Parameter	Tag (String) [in] – Tag of the TLV to find
	DataOut (byte[]) [out] – Pointer to length of the TLV found
	OutLen (int[]) [out] – Pointer to value of the TLV found
Return Value	Refer to ERROR CODE

5) Get Terminal Parameters

Read terminal parameters stored in EMV library.

Function	`void EmvLib_GetParam(EMV_PARAM tParam);`
Parameter	tParam (EMV_PARAM) [out] – Pointer to data of EMV terminal parameter
Return Value	Refer to ERROR CODE
Note	Applications can be obtained through this function EMV terminal parameter value of library, to modify, are updated with the following Settings interface.

6) Get CAPK

Fine one CAPK element by parameter `Index` in EMV library.

Function	`int EmvLib_GetCapk(int Index,` `EMVCAPK capk);`
Parameter	Index (int) [in] – Key index of CAPK to find
	capk (EMVCAPK) [out] – Pointer to the CAPK data found
Return Value	Refer to ERROR CODE

7) Get Application List

Get application list item by parameter `Index`.

Function	`int EmvLib_GetApp(int Index, EMV_APPLIST App);`
Parameter	Index (int) [in] – Index of the application list item in files
	App (EMV_APPLIST) [out] – Pointer to find application list item
Return Value	Refer to ERROR CODE

8) Get Issuer Script Result

Function	`int EmvLib_GetScriptResult(byte[] Result, int[] RetLen);`
Parameter	Result (byte[]) [out] – Pointer to data of script result
	RetLen (int[]) [out] – Pointer to length of script result
Return Value	Refer to ERROR CODE

9) Get the Flag of Signature

Function	`int EmvLib_GetPrintReceiptFlag();`
Parameter	void
Return Value	0: Don’t need a signature 1: Need a signature

10) Get the Flag of Advice

Function	`int EmvLib_GetAdviceReqFlag();`
Parameter	void
Return Value	0: Don’t need advice 1: Need advice

11) Application Selection for Reading Log

Function	`int EmvLib_AppSelForLog(int slot);`
Parameter	slot (int) 0: Contact 1: Contactless
Return Value	Refer to ERROR CODE

12) Read Transaction Log

Function	`int EmvLib_ReadLogRecord(int RecordNo);`
Parameter	RecordNo (int) – Start from No.1
Return Value	Refer to ERROR CODE
Note	Call `EmvLib_AppSelForLog` first

13) Get the Data of Transaction Log

Function	`int EmvLib_GetLogItem(String Tag,`
	`byte[] DataOut,`
	`int[] outLen);`

Parameter	Tag (String) – Label
	DataOut (byte[]) – Label length
	outLen (int[]) – Length of Label data
Return Value	Refer to ERROR CODE
Note	Call `EmvLib_AppSelForLog, EmvLib_ReadLogRecord` first

14) Read Load Log

Function	`int EmvLib_ReadLoadLogRecord(int RecordNo);`
Parameter	RecordNo (int) – Start from No.1
Return Value	Refer to ERROR CODE
Note	Call `EmvLib_AppSelForLog` first

15) Get Common Data of Load Log

Function	`int EmvLib_GetLoadLogCommon(byte[] Data, int[] DataLen);`
Parameter	Data (byte[]) – Data
	DataLen (int[]) – Length of data
Return Value	Refer to ERROR CODE
Note	Call `EmvLib_AppSelForLog`, `EmvLib_ReadLoadLogRecord` first.

16) Get Data from Load Log

Function	`int EmvLib_GetLoadLogItem(String Tag, byte[] TagData, int[] TagLen);`
Parameter	Tag (String) – Label
	TagData (byte[]) – Label data
	TagLen (int[]) – Length of label data
Return Value	Refer to ERROR CODE
Note	Call `EmvLib_AppSelForLog`, `EmvLib_ReadLoadLogRecord`, `EmvLib_GetLoadLogCommon` first.

EMV Kernel Callback Interface

- **Functions**

1) Multiple application options

```
int cEmvLib_WaitAppSel(int TryCnt, String[] Appname, int AppNum,
byte timeout);
```

2) PIN enter callback interface

```
int cEmvLib_GetHolderPwd(int TryCnt,int RemainCnt, byte
pinEntryModel, byte[] pk, byte timeout);
```

1) Multiple Application Options

Multiple application display and select application.

Function	`int cEmvLib_WaitAppSel(int TryCnt,`
	`String[] Appname,`
	`int AppNum,`
	`byte timeout);`

Parameter	TryCnt (int) – The number of times already selected
	Appname (String[]) – Application names
	AppNum (int) – The numbers of application
	timeout (byte) – Must be returned within `timeout` second
Return Value	0: Select the first application 1: Select the second application and so on

2) PIN Enter Callback Interface

Enter PIN.

Function	`int cEmvLib_GetHolderPwd(int TryCnt,`
	`int RemainCnt,`
	`byte pinEntryModel,`
	`byte[] pk,`
	`byte timeout);`

Parameter	TryCnt (int) – The number of attempts, only used by offline pin
	RemainCnt (int) – The number of attempts remain
	pinEntryModel (byte) 1: Offline plain password 2: Offline encryption password 3: Online pin

	pk (byte[]) – The data of public key *(Mould length (1 byte)* + *Mould* + *Exponent length (1 byte)* + *Exponent)*, this data only used for offline cipher pin
	timeout (byte) – Must be returned within `timeout` second
Return Value	0: Success Other: Refer to ERROR CODE

ERROR_CODE

public static final int EMV_OK	(-0)	Success
public static final int ERR_EMVRSP	(-1)	Return code error
public static final int ERR_APPBLOCK	(-2)	Application locked
public static final int ERR_NOAPP	(-3)	No application in card
public static final int ERR_USERCANCEL	(-4)	User cancel
public static final int ERR_TIMEOUT	(-5)	Timeout
public static final int ERR_EMVDATA	(-6)	Card data error
public static final int ERR_NOTACCEPT	(-7)	Transaction not accept
public static final int ERR_EMVDENIAL	(-8)	Transaction declined
public static final int ERR_KEYEXP	(-9)	The key expired
public static final int ERR_NOPINPAD	(-10)	No pinpad
public static final int ERR_NOPIN	(-11)	No PIN input
public static final int ERR_CAPKCHECKSUM	(-12)	Check sum error
public static final int ERR_NOTFOUND	(-13)	No data found
public static final int ERR_NODATA	(-14)	No data found
public static final int ERR_OVERFLOW	(-15)	Over flow
public static final int ERR_NOTRANSLOG	(-16)	No log
public static final int ERR_NORECORD	(-17)	No record
public static final int ERR_NOLOGITEM	(-18)	No log item
public static final int ERR_ICCRESET	(-19)	ICC reset error
public static final int ERR_ICCCMD	(-20)	ICC command error
public static final int ERR_ICCBLOCK	(-21)	ICC locked
public static final int ERR_ICCNORECORD	(-22)	ICC no record
public static final int ERR_USECONTACT	(-23)	RFID failed
public static final int ERR_APPEXP	(-24)	qPBOC card expired
public static final int ERR_BLACKLIST	(-25)	qPBOC black list card
public static final int ERR_GPORSP	(-26)	Error from GPO

public static final int ERR_USE_OTHER	(-27)	
public static final int ERR_LASTREAD	(-28)	Read last recode error
public static final int ERR_TRANSEXCEEDED	(-29)	Trance exceeded
public static final int ERR_NULL	(-30)	Path error
public static final int ERR_NOAMT	(-31)	No Amount Error
public static final int ERR_PINBLOCK	(-32)	PIN locked
public static final int EMV_FILE_ERR	(-33)	EMV file error
public static final int EMV_DATA_EXIST	(-34)	Data has existed
public static final int ERR_APPPATH	(-35)	Path error
public static final int ERR_PAYSCHEME	(-37)	Refer your mobile payment
public static final int ERR_NOTALLOWED	(-38)	Application is not allowed

3. Structure Description

EMV_PARAM

```
public class EMV_PARAM {
    private byte[] MerchName = new byte[64];        // Merchants Name (9F4E)
    private byte[] MerchCateCode = new byte[2];     // Merchant category code (9F15)
    private byte[] MerchId = new byte[15];          // Merchants ID (9F16)
    private byte[] TermId = new byte[8];            // Terminal ID (9F1C)
    private byte TerminalType;                      // Terminal Type (9F35)
    private byte[] Capability = new byte[3];        // Terminal capability (9f33)
    private byte[] ExCapability = new byte[5];      // Additional Terminal Capabilities (9F40)
    private byte TransCurrExp;                      // Transaction currency exponent (5F36)
    private byte ReferCurrExp;                      // Transaction Reference currency exponent
                                                       (9F3D)
    private byte[] ReferCurrCode = new byte[2];     // Transaction Reference Currency Code
                                                       (9F3C)
    private byte[] CountryCode = new byte[2];       // Terminal country code (9F1A)
    private byte[] TransCurrCode = new byte[2];     // Transaction currency code (5F2A)
    private long ReferCurrCon;                      //Transaction Reference Currency Conversion.
                                                      Factor used in the conversion from the
                                                      Transaction Currency Code to the Transaction
                                                      Reference Currency Code
    private byte bBatchCapture;                     // Whether Support batch capture
    private byte bSupportAdvices;                   // Whether Support Advice
    private byte TransType;                         // Transaction type (9C)
    private byte ForceOnline;                       // 1: force the transaction online
    private byte GetDataPIN;                        // Whether support to get the remaining count
                                                       before offline PIN execution,
                                                       1: Support, 0: Not support
    private byte SurportPSESel;                     // Whether support PPSE
    private byte[] TermTransQuali = new byte[4];    // Terminal transaction quality (9f66)
    private byte ECTSI;                             // Not used for EMV
```

```
    private byte EC_bTermLimitCheck;          // Not used for EMV
    private long EC_TermLimit;                // Not used for EMV
    private byte CL_bStatusCheck;             // Not used for EMV contact
    private long CL_FloorLimit;               // Not used for EMV contact
    private long CL_TransLimit;               // Not used for EMV contact
    private long CL_CVMLimit;                 // Not used for EMV contact
    private byte SMTSI;                       // Not used for EMV contact
    private byte bExceptionFile;              // Whether the exception file is supported,
                                                 1: support, 0: not support
    private byte[] IFD_SN=new byte[9];        // Interface device serial number (9F1E)
}
```

EMVCAPK

```
public class EMVCAPK {
    private byte[] RID = new byte[5];         // Application Registrar ID
    private byte KeyID;                       // Key index
    private byte HashInd;                     // HASH algorithm index
    private byte ArithInd;                    // RSA algorithm index
    private byte ModulLen;                    // Modul len
    private byte[] Modul = new byte[248];     // Modul
    private byte ExponentLen;                 // Exponent len
    private byte[] Exponent = new byte[3];    // Exponent
    private byte[] ExpDate = new byte[3];     // Expire date(YYMMDD)
    private byte[] CheckSum = new byte[20];   // Key check sum
}
```

EMV_APPLIST

```
public class EMV_APPLIST {
    private byte[] AID = new byte[17];        // AID
    private byte AidLen;                      // Length of AID
    private byte SelFlag;                     // Select sign (Partial matching / full matching)
```

```
    private byte Priority;                          // Priority flag
    private byte TargetPer;                         // Target percentage
    private byte MaxTargetPer;                      // Maximum target percentage
    private byte FloorLimitCheck;                   // Whether the floor limit checking is supported
    private byte RandTransSel;                      // Whether to make random trade selection
    private byte VelocityCheck;                     // Whether Velocity detection is performed
    private long FloorLimit;                        // Floor limit (9f1B)
    private long Threshold;                         // Threshold value
    private byte[] TACDenial = new byte[6];         // Terminal behavior code (decline), 5 byte
    private byte[] TACOnline = new byte[6];         // Terminal behavior code (online), 5 byte
    private byte[] TACDefault = new byte[6];        // Terminal behavior code (default), 5 byte
    private byte[] AcquierId = new byte[7];         // Acquire ID 6 byte
    private byte[] dDOL = new byte[256];            // Default DDOL, B0: the value length
    private byte[] tDOL = new byte[256];            // Default TDOL, B0: the value length
    private byte[] Version = new byte[3];           // Application version, 2 byte
    private byte[] RiskManData = new byte[10];      // Risk management data, B0: the value length
    private byte EC_bTermLimitCheck;                // NONE
    private long EC_TermLimit;                      // NONE
    private byte CL_bStatusCheck;                   // NONE
    private long CL_FloorLimit;                     // NONE
    private long CL_TransLimit;                     // NONE
    private long CL_CVMLimit;                       // NONE
}
```