ADVANCED MOBILE PAYMENT INC.

AMP EMV LEVEL 2

Developer Guide

v 2.1
PDV-5005-2.1-E

PROPRIETARY NOTICE

All pages of this document contain information proprietary to Advanced Mobile Payment (AMP) Inc. This document shall not be duplicated, transmitted, used, or otherwise disclosed to anyone other than the organization or specific individuals to which this document is delivered. This restriction is applicable to all sheets of this document. AMP reserves the right to have the recipient return all copies of this document at any time. AMP reserves the right to change the content of this document without prior notice.

DOCUMENT PROPERTIES

INFORMATION

ID	PDV-5005-2.1-E
Title	AMP EMV Level 2 - Developer Guide
Document Portal Path	/Development/AMP POS/AMP 3,5,7,9 Series /Development/AMP POS/AMP 6 Series /Development/AMP POS/AMP 8 Series
Category	AMP POS – 3,5,7,9 Series
Access Level	Development
NDA Required	Yes ☑ No☐

VERSION CONTROL

Template Version	1.3

Doc. Version	Date	Summary of Change	Updated by
V1.00	Oct 2016	Initial Draft	M.Tadili
V1.01	Oct 2016	Added new Interface: ClearMemoryAgnosConfig() GetRecordCAKeysXml() GetRecordCRLConfig() Added more information for Configuration APIs Added sample build and compiler settings	M.Tadili
V1.02	Nov 2016	Added build and compiler setup in Eclipse for DemoAppEMV application Added GetEMVTag() Modified CAKey, CRL sections Corrected UNABLE_TO_GO_ONLINE	M.Tadili
V1.03	Nov 2016	Modified Display Message callback where it now has “MessageId” to allows application layer to use MessageId instead of the actual translated	M.Tadili

		message. `unsigned short (*CBDisplayMessage) (unsigned char bClearScreen, unsigned char LangId, const char *szMessage, unsigned char MessageId);` Added list of MessageId for developer's reference Added notes for CardEntryTimeOut in CardEntryPolling()	
V1.04	Nov 2016	Added sample code for TERMINAL, PROCESSING, and ENTRY_POINT of DeleteRecordToAgnosConfig() API Added SetPollingTimeOut() API Added new SetFieldTagValueAgnosConfig() with TLV set of input parameters Modified AddRecordToAgnosConfig() with TLV set of input parameters Added sample for SetFieldTagValueAgnosConfig() and AddRecordToAgnosConfig() Added additional information for contactless kernels	M.Tadili
V1.05	Dec 2016	Added SetAmount() API description Added SetCashbackAmount() API description Added SetTransCurrCode() API description Corrected 9F01 information under PROCESSING section Added SetMaxPinLength() description	M.Tadili
V1.06	Dec 2016	Modified agnos.ini to include CurrencyCode=0840 parameter for default Transaction Currency Code (5F2A)	M.Tadili
V1.07	Dec 2016	Modified agnos.ini for ErrLEDDisplay=Yes parameter to enable RED LED display when error	M.Tadili

		encountered on contactless payment transaction Added ErrorLEDDisplay() API for LED display error processing Added Technical Notes/Limitations on Appendix 3 Corrected numbering of the document	
V1.08	Jan 2017	Changed GetLanguageCodeID() and SetLanguageCodeID() API functions to GetDefLanguageCodeID() and SetDefLanguageCodeID() respectively. Changed SetMaxPinLength() API function to SetExpectedPinLen() Modified CBAppSelection() API callback Modified CBSelectItemFromList() API callback Removed CBDisplayMaskedPin() API Modified CBOnlinePinEntry() API	M.Tadili
V2.0	Jan 2017	Modified CardEntryPolling() to support Mifare Desfire card detection Added SetMifareDetection() API to enable Mifare Desfire detection during CardEntryPolling() Modified agnos.ini file for new configurable parameters: USDebitAppSelect=Yes QuickChip=Yes EnableMifareDetect=No Added ENTRY_POINT configuration for Interac-Flash Modified ENTRY_POINT configuration for Discover-DPAS Added EMV Quick Chip Section Added US Debit Application Selection	M.Tadili

		Modified PowerOn() API return value Modified app file, added libJSpeedy.so and libFlash.so Added SetInitTransData() API to initialize transaction data needed to process payment transaction	
V2.1	May 2018	Imported to the new template Modified US Debit Final Selection section Modified Callback Section Added SetTranSeqCounter() API function Android Support	N.Abouelsaad M.Tadili K.Concha

SUPPORTED HARDWARE & SOFTWARE

Doc. Version	Software Title	Release	Supported Hardware Model
1.0 – 2.1	DemoAppEMV for Linux	v1.0.4	AMP 3,5,7,9 Series
1.0 – 2.0	AMPEMVL2	V2.0.2	AMP 3,5,7,9 Series
1.0 – 2.1	AMP 3,5,7,9 Series SDK Package	V1.0.1	AMP 3,5,7,9 Series
1.0 – 2.1	Agnos EMV Kernel	V3.0.1	AMP 3,5,7,9 Series
2.1	DemoAppEMV for Android	V1.2.0	AMP 6 and 8 Series
2.1	AMPEMVL2	V2.0.6	AMP 3,5,6,7,8,9 Series
2.1	AMP 8 Series SDK Package	V1.0.3	AMP 8 Series
2.1	AMP 6500 SDK Package	V1.0.0	AMP 6500
2.1	AMP 6700 SDK Package	V1.0.0	AMP 6700

EXTERNAL REFERENCES

URLs	--
E-Mail Addresses	--
Phone Numbers	--
Documents	--
Knowledge Base Articles	--

TABLE OF CONTENTS

TABLE OF FIGURES

1 ABOUT AMPEMVL2

The AMPEMVL2 library acts as an Application Development Kit component, in addition to being a framework which wraps the functionalities of the EMV contact Level-2 kernel and contactless entry point kernel processing. It primarily aims to help payment application developers to simplify the implementation of the EMV contact and contactless on their payment application development.

The figure below demonstrates the high level overview of the system from the application developer perspective. This allows developers to easily visualize how a library can be utilized and integrated on their payment application layer.

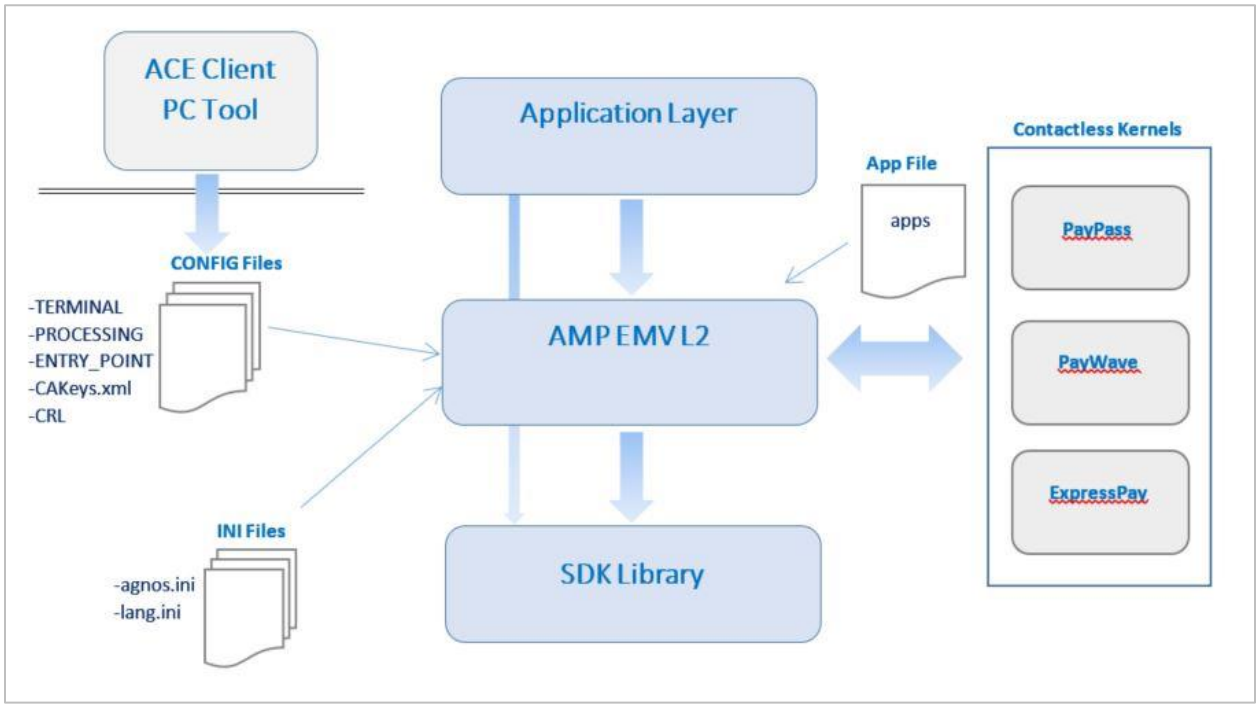

Figure 1 - AMPEMVL2 System Overview.

The system can be summarized to four core functions:

1) Agnos Framework (cAMPAgnos) – a framework that manages and compartmentalizes its main functionalities within the AMPEMVL2 library.

2) Agnos EMV Level-2 (cAMPAgnosEMV) – a module that handles EMV contact processing

3) Agnos Entry Point Contactless (cAMPAgnosEMVCtls) – a module that handles contactless processing

4) Agnos Configuration (cAMPAgnosConfig) – another module that allows the payment application developers to update the contents of the configuration file through API interfaces

1.1 THE AMPEMVL2 ARCHITECTURE

The AMPEMVL2 library can be visualized as shown in the following diagram, as a component which provides functional API interfaces that can be utilized by the application layer to build complete EMV contact and contactless transactions.

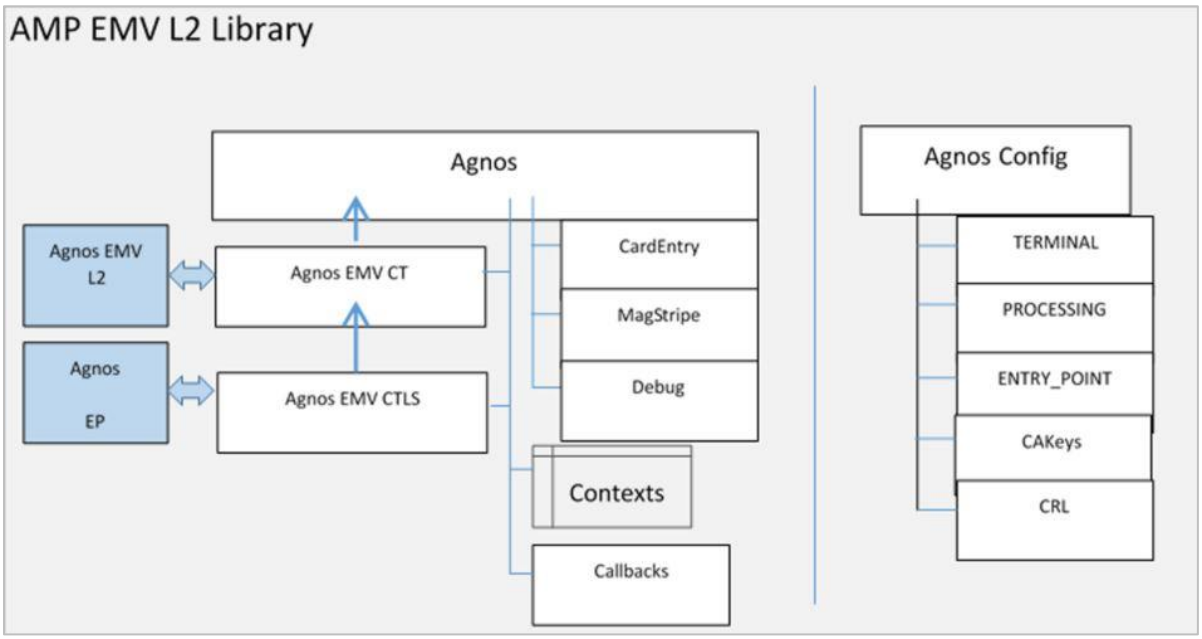

Figure 2 - AMPEMVL2 Architecture.

The AMEMVL2 is built on top of Agnos EMV L2 (Agnos) and Agnos Entry Point (AgnosEP) which are static libraries that are separately maintained by an expert team with expertise in library/kernel development conforming to EMVco EMV contact and contactless standards.

Looking back at the diagram, AMPEMVL2 also offers API functions as follows:

1) Card Entry Polling function which can be used to detect card entry from the user

2) Reading magnetic stripe data for swipe card entry

3) Debugging mechanism

2 AMPEMVL2 LIBRARY COMPONENT

The AMPEMVL2 library is composed of the following files that need to be included in the payment application build and compilation process.

AMPEMVL2

- 2.0.6
  - apps
    - libDPAS10.so
    - libEMVCo.so
    - libPayPass3x.so
    - libPayWave2x.so
    - libXPressPay3x.so
    - libJSpeedy.so
    - libFlash.so
  - config
    - agnos.ini
    - apps
    - CAKeys
    - ENTRY_POINT
    - lang.ini
    - PROCESSING
    - TERMINAL
  - lib
    - libAMPEMVL2.so
  - include
    - agnos.h
    - agnoserrors.h
    - agnostypes.h
    - AMPAgnos.h
    - AMPAgnosConfig.h
    - AMPAgnosConfigTagDefs.h
    - AMPAgnosEmv.h
    - AMPAgnosEmvCtls.h
    - AMPDataTypes.h
    - AMPEmvSizes.h
    - AMPReturnCodes.h
    - BER-TLV.h
    - datamngr.h
    - dataxchng.h
    - entrypoint.h
    - gpi.h
    - gpicad.h
    - gpidisplay.h
    - gpierrors.h
    - gpihsm.h
    - gpilog.h
    - gpiplatform.h
    - gpisped.h
    - gpiutils.h
    - paymenterrors. h

- paymentMW.h
- pos_interface.h
- selection.h
- sep.h
- tinyxml2.h
- tlv.h
- tlvdefs.h
- xgpiutils.h

3 DEMOAPPEMV BUILD AND COMPILER SETTINGS

This section discusses the build and compiler setup of the DemoAppEMV application that will serve as a reference for the application layer developers for the coding implementation, utilizing the AMPEMVL2's application program interface (API).

Here are the key items that need to be setup in the build process for properly compiling the DemoAppEMV.

1) AMPEMVL2 Location

C:/AMP7000/

- AMPEMVL2
  - 2.0.6 (see AMPEMVL2 Library Component section)
- sdk

C:/ApexProjects/DemoAppEMV

- .settings
- include
  - AMPUART.h
  - Debug.h
  - Defines.h
  - Display.h
  - EmvCallback.h
  - EmvConfig.h
  - EmvCtlsFunc.h
  - EmvFunc.h
  - Flow.h
  - Menu.h
  - PedFuncs.h
  - PinEntry.h
  - Transactions.h
- Source
  - EMV
  - EmvCallback.cpp
  - EmvConfig.cpp

    - EmvCtlsFunc.cpp
    - EmvFunc.cpp
    - EmvUtils.cpp
    - Flow
    - CardEntry.cpp
    - PedFuncs.c
    - PinEntry.c
    - Main
    - Main.c
    - Screen
    - Display.c
    - Font.c
    - Screens.c
    - Transaction
    - Transactions.c
- .cproject
- .project
- postBLD.mk

2) Environment variables:

```
AMPEMVL2_VER= 2.0.6
SDK_PATH = /cygdrive/c/AMP7000
```

3) C/C++ Compiler Settings:

- Include Paths (-I)

```
"${SDK_PATH}/AMPEMVL2/${AMPEMVL2_VER}/include"
```

4) C/C++ Linker Settings:

- Libraries (-l)
  AMPEMVL2d
- Library search path (-L)

```
"${SDK_PATH}/AMPEMVL2/${AMPEMVL2_VER}/lib"
```

3.1 ENVIRONMENT VARIABLES

The image below illustrates the Environment Settings in the Eclipse IDE. You may want to double check the following environment parameters:

- AMPEMVL2_VER → version number of the AMPEMVL2
- CROSS_COMPILE → location of the gcc, g++ compiler
- PROJ_DIR → location of the DemoAppEMV project directory

- SDK_PATH → location of the SDK directory

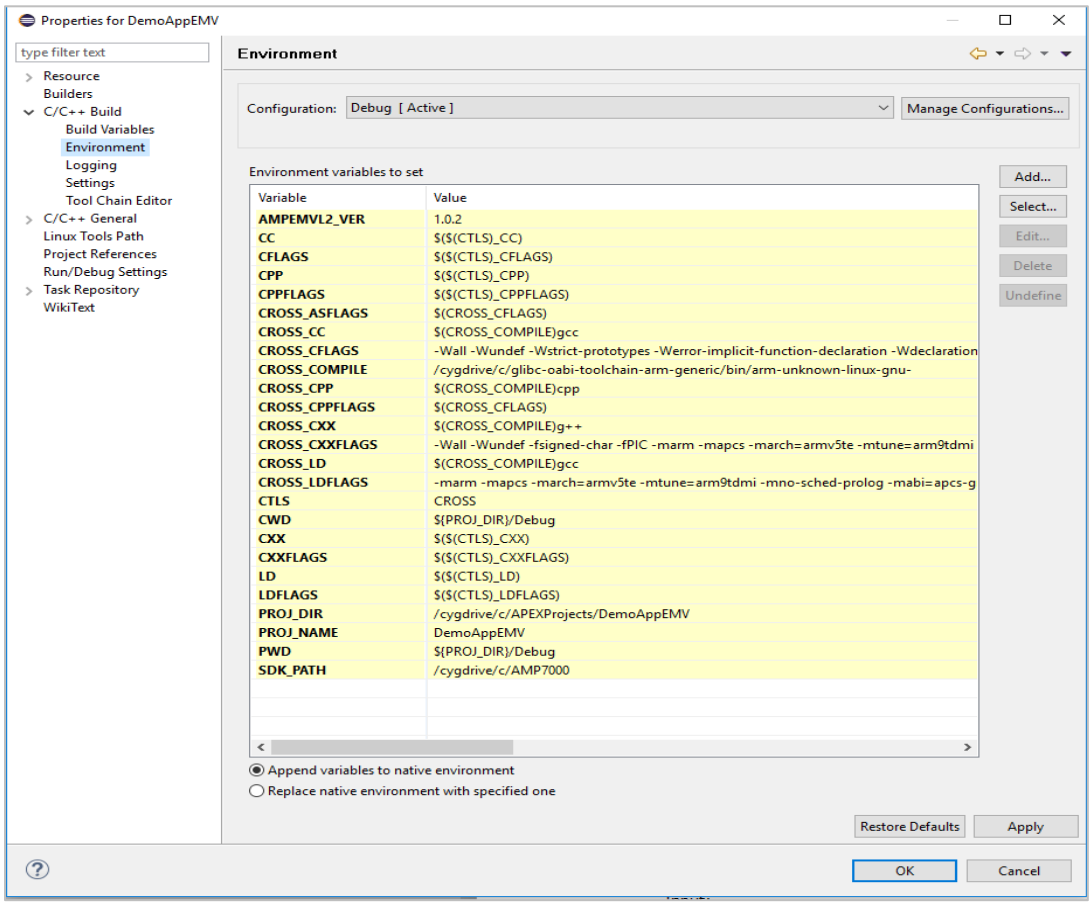

Figure 3 - Environment Settings in the Eclipse IDE.

3.2 COMPILER SETTINGS

The AMPEMVL2 library is written in C++ while DemoAppEMV is written in C/C++. Therefore, the project workspace environment should be in C and C++ like the one illustrated below.

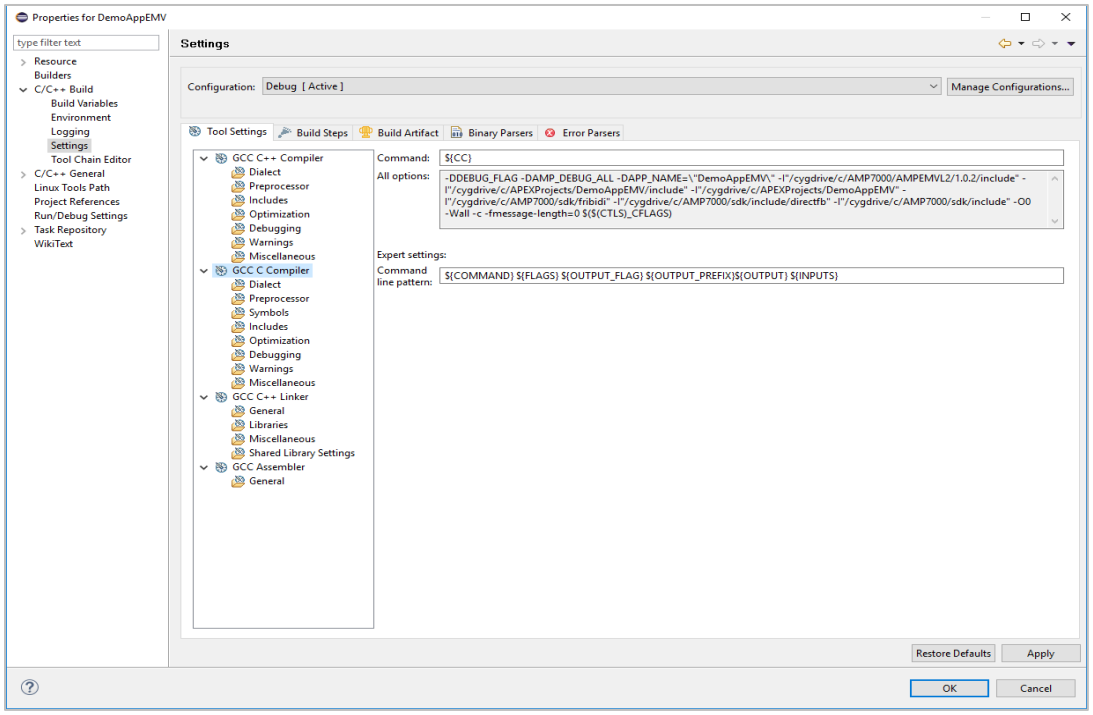

Figure 4 - The Workspace Environment in C and C++.

Subsequently, library paths should be added to the compiler (–I) option. In the following example, the AMPEMVL2 header files should be included along with the SDK and application project headers.

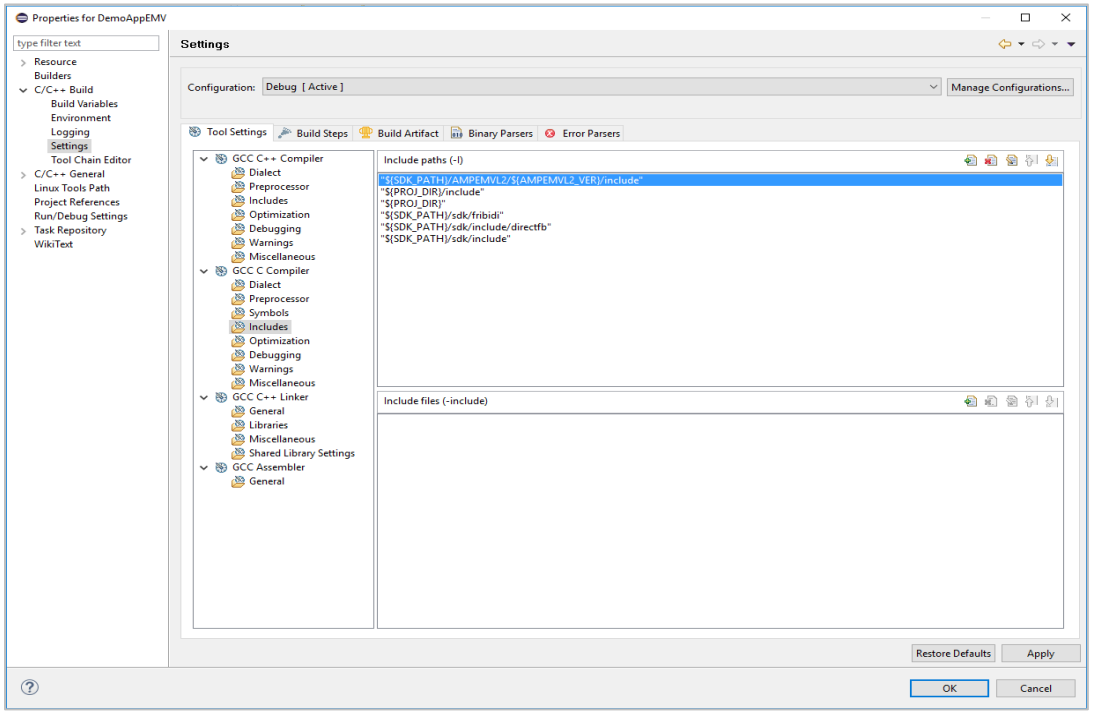

Figure 5 - Including the Library Paths in the Compiler (-I).

3.3 LINKER SETTINGS

The linker setup should resemble the sample figure.

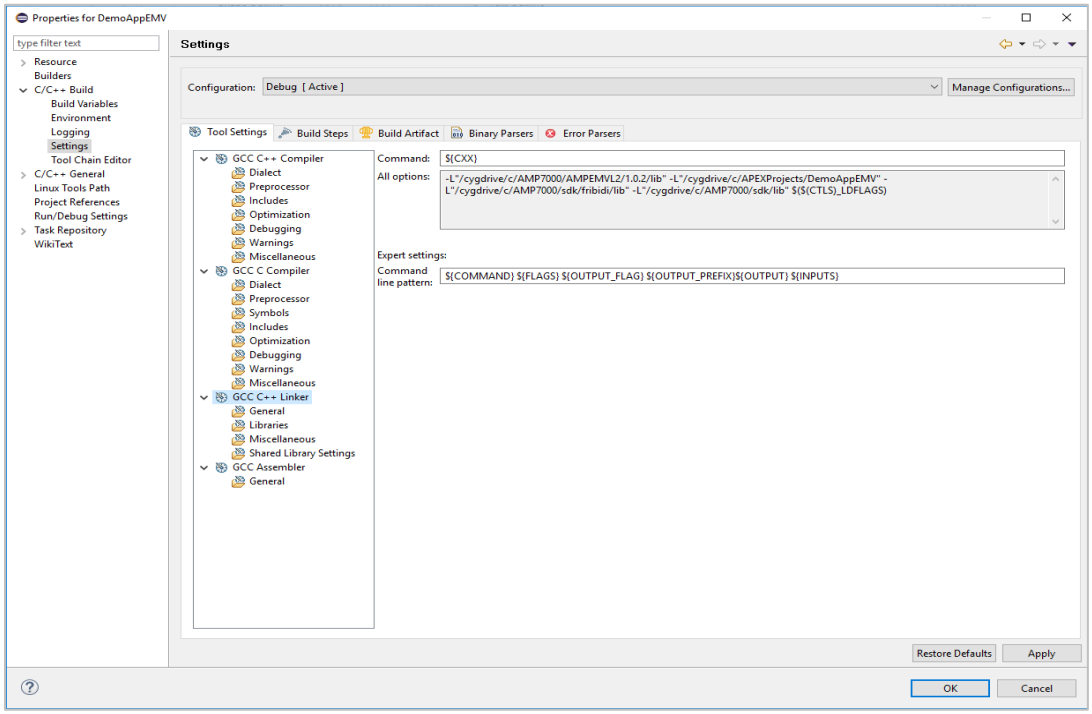

Figure 6 - The Linker Settings.

Setup the Libraries (-l), in this case we add the AMPEMVL2 library

- AMPEMVL2 – For the release version
- AMPEMVL2d – For the debug version

Additionally, the location of the AMPEMVL2 should be indicated in the Library Path (-L) linker option, as follows:

```
"${SDK_PATH}/AMPEMVL2/${AMPEMVL2_VER}/lib"
```

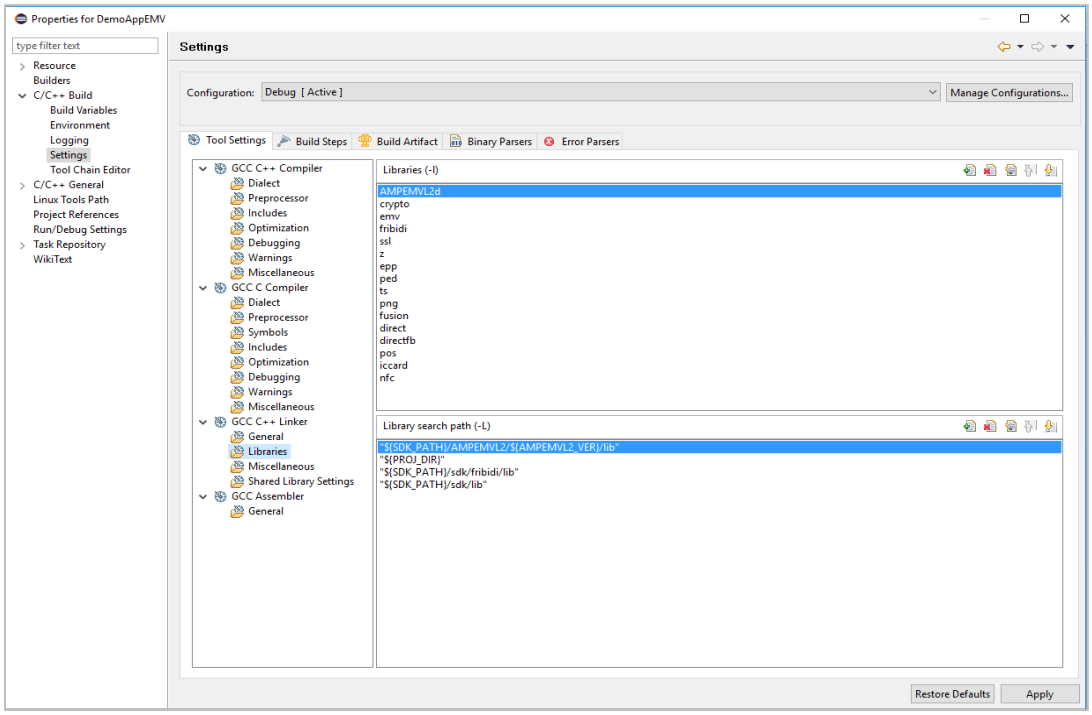

Figure 7 – Setting the Libraries (-l) and Updating the Library Path.

4 DEMOAPPEMV APPLICATION FILE TREE STRUCTURE

The AMPEMVL2 library also includes a demo application called DemoAppEMV that will serve as an application reference in conjunction with the document reference guide for the payment application developers.

Below are the files and structure that should be downloaded to the terminal for properly installing the demo application.

1) DemoAppEMV

- AGNOS
  - apps
  - CAKeys
  - CRL
  - ENTRY_POINT
  - PROCESSING
  - TERMINAL
  - libDPAS10.so
  - libEMVCo.so
  - libPayPass3x.so
  - libPayWave2x.so
  - libXPressPay3x.so

    - libFlash.so
    - libJSpeedy.so
- agnos.ini
- lang.ini
- CAKeys.xml
- DemoAppEMV
- QuickChip.cfg
- USDebit.cfg

2) lib

- libAMPEMVL2.so
- libfribidi.so

3) Desktop

- DemoAppEMV.desktop

4) Fonts

5 API REFERENCE – AMP EMV L2 INITIALIZATION

This section covers the initialization routines for the AMPEMVL2 library that needs to be called by the application layer in order to utilize its functionalities. It mainly covers the following six points:

1) Instantiation of the singleton instance (cAMPAgnos, cAMPAgnosEmv, and cAMPAgnosCtls) in the memory space
2) Initialization of the DEBUG port
3) Initialization of the EMVL2 and EntryPoint
4) Initialization of Language settings
5) Registration of callback functions
6) Initialization of the transaction

5.1 API INTERFACE

5.1.1 INSTANTIATION OF THE AMPEMVL2 LIBRARY

Prototype	`static cAMPAgnos* get_Instance(void);`
Function	The get_instance() API function is the actual instantiation of the singleton instance of the library.
Input	NA
Output	NA
Return	Pointer object of AMPAgnos

Notes	Here is a code snippet for proper instantiation:
	`//NOTE: Do not edit by order of creation of instances to maintain singleton of instance!`
	`cAMPAgnosEmvCtls *pAgnosEmvCtls = cAMPAgnosEmvCtls::get_Instance();`
	`cAMPAgnosEmv *pAgnosEmv = cAMPAgnosEmv::get_Instance();`
	`cAMPAgnos *pAgnos = cAMPAgnos::get_Instance();`

5.1.2 SETUP DEBUG LOG PORT

Prototype	`void SetDebugLogPort(`
	`const char *szAppName,`
	`int nLogPort);`

Function	The SetDebugLogPort() interface can be used to write debug logs in the specified port which will serve as a debugging tool for the application layer developers.

Input	szAppName – The application name
	nLogPort – Specifies the port in which logs will be written
	`#define PORT_COM1 0x00`
	`#define PORT_XX 0x06 // USB port`

Output	NA
Return	NA
Notes	NA

5.1.3 INITIALIZATION OF THE AMPEMVL2 LIBRARY

Prototype	`int Initialize(void);`
Function	This API has to be called during application initialization to perform its initialization routine such as reading and loading the library's agnos.ini and lang.ini files. Furthermore, it loads the contactless kernels as defined in the system's apps file.
Input	NA
Output	NA
Return	AMP_SUCESS
Notes	NA

5.1.4 REGISTRATION OF CALLBACK FUNCTIONS

Prototype	`void RegisterCallback(` `  AMPAGNOS_CB *callback);`
Function	The RegisterCallback() function has to be called during application initialization to override the built-in functions of the AMPEMVL2 library by the application layer.
Input	callback – and AMPAGNOS_CB structure that holds the callback function pointers
Output	NA
Return	NA
Notes	Please refer to **API Reference - Callback Functions** for more details.

5.1.5 INITIALIZATION OF LANGUAGE SETTINGS

Prototype	`unsigned char GetDefLanguageCodeID(` `  const char *langCode);`
Function	The AMPEMVL2 library has its own language support functionality with the use of lang.ini. Thus, the application layer may use the Get/SetDefLanguageCodeID() API to set the default and synchronize the language system of the application layer against the AMPEMVL2 framework.

Input	langCode – language code
Output	NA
Return	Language Id
Notes	NA

Prototype	`int SetDefLanguageCodeID(` `  const char *langCode);`
Function	The AMPEMVL2 library has its own language support functionality with the use of lang.ini. Thus, the application layer may use the Get/SetDefLanguageCodeID() API to set the default and synchronize the language system of the application layer against the AMPEMVL2 framework.
Input	langCode – Language Code (“ar”, “en”, “fr”)
Output	NA
Return	AMP_ERROR, AMP_SUCCESS
Notes	NA

5.1.6 INITIALIZATION OF TRANSACTION

Prototype	`int InitTrans(` `  AGN_TRANS_TYPE TransType);`
Function	The InitTrans() API has to be invoked for every transaction by the application layer to perform the transaction initialization routine indicating the transaction type as the input. During this function call, the AMPEMVL2 framework reads the configuration files (TERMINAL, PROCESSING, ENTRY_POINT) and initializes the contactless applications to prepare for the contact and contactless transaction card acceptance.
Input	TransType – The transaction type (refer to AGN_TRANS_TYPE)
Output	NA
Return	AMP_SUCCESS
Notes	NA

5.1.7 INITIALIZATION OF TRANSACTION DATA

Prototype	`int SetInitTransData(` `  const tInitTransData * InitTransData);`
Function	The SetInitTransData() API is used to initialize the value of the transaction data needed for a payment transaction.
Input	InitTransData – Initial Transaction Data structure (refer to **Initialization of Transaction**)
Output	NA
Return	AMP_SUCCESS

Notes	`typedef struct {`
	`  // Information provided by Level3`
	`unsigned long  mAmount;  // EMV Tag 9F02 =`
	`"Amount, Authorized" + "Amount, Other"`
	`unsigned long  mCashBack;  // EMV Tag 9F03 =`
	`"Amount, Other"`
	`AGN_TRANS_TYPE  mTransactionType;// EMV Tag 9C`
	`BYTE  mTransactionDate[3];// EMV Tag 9A(yymmdd)`
	`BYTE  mTransactionTime[3];// EMV Tag 9F21`
	`(hhmmss)`
	`BYTE  mMerchantCustomData[20];// EMV Tag 9F7C`
	`BYTE  mTransactionCurrencyCode[2];// EMV Tag`
	`5F2A`
	`BYTE  mTransactionCurrencyExponent;// EMV Tag`
	`5F36`
	`BYTE  mTransactionCategoryCode;// EMV Tag 9F53`
	`BYTE  mMerchantCategoryCode[2];// EMV Tag 9F15`
	`BYTE  mMerchantID[15]; // EMV Tag 9F16`
	`BOOL  mForcedOnline; // As per merchant`
	`decision`
	`unsigned long  mTransactionSequenceCounter; //EMV`
	`Tag9F41`
	`} tInitTransData;`

5.1.8 SET AMOUNT

Prototype	`int SetAmount(` `  unsigned long Amount);`
Function	This is used to set the Tag-9F02 of the EMV transaction amount.
Input	Amount – The transaction amount (tag 9F02) for the EMV payment transaction

Output	NA
Return	AMP_SUCCESS
Notes	SetAmount() has to be called prior to the actual payment transaction execution.

5.1.9 SET CASHBACK AMOUNT

Prototype	`Int SetCashbackAmount(` `  unsigned long CashbackAmount);`
Function	This is used to set the Tag-9F03 of the EMV transaction's other amount.
Input	CashbackAmount – The cashback amount or other amount (tag 9F03) of the EMV payment transaction
Output	NA
Return	AMP_SUCCESS
Notes	SetCashbackAmount() has to be called prior to the actual payment transaction execution.

5.1.10 SET TRANSACTION CURRENCY CODE

Prototype	`int SetTransCurrCode(` `  const BYTE CurrCode[2]);`
Function	SetTransCurrCode() has to be called prior to the actual payment transaction execution.
Input	CurrCode – A 2-bytes transaction currency code (tag 5F2A) of the EMV payment transaction
Output	NA
Return	AMP_SUCCESS

Notes	Code Sample:
	
	`int DoAgnosEmvCtlsInitiateFlow(void)`
	`{`
	`  int iRetVal = AMP_SUCCESS;`
	`  long lnAuthAmount = 0, lnCBAmount = 0;`
	`  int EmvCardDecision = 0;`

```
BYTE CurrCode[2];
tInitTransData InitTransData;

cAMPAgnosEmvCtls*pAgnosEmvCtls =
cAMPAgnosEmvCtls::get_Instance();

//1 Initial Setup for Agnos Transaction
//Temporary Values. Create Display for amount
entry
lnAuthAmount = atol(g_TranObj.szTranAmt);
lnCBAmount = atol(g_TranObj.szCashbackAmt);

//Set Initial Transaction Data
InitTransData.mAmount = lnAuthAmount;
InitTransData.mCashBack = lnCBAmount;
InitTransData.mTransactionType =
      (AGN_TRANS_TYPE) g_TranObj.TransType;
//InitTransData.mTransactionDate = currDate;
      yymmdd
//InitTransData.mTransactionTime = currTime;
      hhmmss
memset(InitTransData.mMerchantCustomData, 0xFF,
sizeof(InitTransData.mMerchantCustomData));
memcpy( InitTransData.mTransactionCurrencyCode,
"\x06\x82",
sizeof(InitTransData.mTransactionCurrencyCode))
;
InitTransData.mTransactionCurrencyExponent =
0xFF;
InitTransData.mTransactionCategoryCode = 0xFF;
memcpy( InitTransData.mMerchantCategoryCode,
"\x12\x34",
sizeof(InitTransData.mMerchantCategoryCode));
memcpy(InitTransData.mMerchantID,
"\x31\x32\x33\x34\x35\x36\x37\x38\x39\x41\x42\x
43\x44\x45\x46",
sizeof(InitTransData.mMerchantID));
InitTransData.mForcedOnline = FALSE;

//2 Set InitTransData
pAgnosEmv->SetInitTransData(&InitTransData);

//2 1. Setup Tag List
pAgnosEmvCtls->SetTagListCollection( gTagList,
sizeof(gTagList));

//2 2. Init Trans
iRetVal = pAgnosEmvCtls->InitTrans();
```

```
	iRetVal =
	pAgnosEmvCtls->ProcessCtlsEntryPoint();

	//2 Evaluate Outcome

	switch(iRetVal)
		{
		case ocNOT_EMV_CARD_POOLED: //MAG EMV
					    DATA
			break;
		case ocTRY_ANOTHER_INTERFACE: //To
					    fallback?
			BeepWarning();
			break;
		case ocTRANSACTION_CANCELLED:
			iRetVal = AMP_CANCEL;
			pAgnosEmvCtls->BeepError();
			break;
		case ocONLINE_REQUEST:
			EmvCardDecision = EMV_GO_ONLINE;
			break;
		case ocOFFLINE_APPROVED:
		case ocAPPROVED:
			pAgnosEmvCtls->BeepSuccess();
			EmvCardDecision = EMV_APPROVED;
			break;
		case ocOFFLINE_DECLINED:
		case ocDECLINED:
		case ocNOT_ACCEPTED:
		case ocOFFLINE_FAILED:
		case ocEND_APPLICATION:
		case ocFAILED:
		default:
			pAgnosEmvCtls->BeepError();
			EmvCardDecision = EMV_DECLINE;
			iRetVal = AMP_ERROR;
			break;
	}

	g_TranObj.EmvCardDecision = EmvCardDecision;

	pAgnosEmvCtls->RemoveTapCard();
	//1 10. Get Emv Tag Values
	DoAgnosEmvCtlsFirstReadTLVCollxn();

	return iRetVal;
}
```

5.1.11 SET TRANSACTION SEQUENCE COUNTER FOR TAG 9F41

Prototype	`int SetTransSeqCounter(` `  unsigned long SeqNo);`
Function	This API can be used to set the Tag 9F41-Transaction Sequence Counter.
Input	SeqNo – The Sequence Counter for Tag 9F41
Output	NA
Return	AMP_SUCCESS
Notes	SetTransSeqCounter() has to be called prior to the actual payment transaction execution.

5.2 INCLUDE HEADER LIBRARY

The APIs discussed above are defined in the header files as listed below, hence they should be included in the application layer code implementation.

- AMPAgnos.h
- AMPAgnosEmv.h
- AMPAgnosEmvCtls.h

5.3 CODE SAMPLE

The following code sample demonstrates the initialization of the AMPEMVL2 library.

```
void InitializeAMPEMVL2()
{
unsigned char LangId = 0;
char DefaultLang[256] = {0};

//Create Instance of EMV Object
//NOTE:  Do not edit by order of creation of instances to maintain
singleton of instance!
cAMPAgnosEmvCtls *pAgnosEmvCtls = cAMPAgnosEmvCtls::get_Instance();
cAMPAgnosEmv *pAgnosEmv = cAMPAgnosEmv::get_Instance();
cAMPAgnos *pAgnos = cAMPAgnos::get_Instance();

//Set Debug Log Application Name and Port
pAgnos->SetDebugLogPort(APP_NAME, PORT_COM1);

//Initialize Agnos Framework
pAgnos->Initialize();
```

```
//Register Callback
pAgnos->RegisterCallback(GetAgnosCallbackFunc());

//UpdateEMVLanguageSetting
LangID = pAgnos->GetLanguageCodeID(DefaultLang);
pAgnos->GetLanguageCodeID(DefaultLang, LangId);

}
```

6 API REFERENCE – AMP EMV L2 DATA TYPES AND STRUCTURES

6.1 TRANSACTION OUTCOME

This can be used to evaluate the result of the EMV contact and contactless main functions:

EmvInitiateApplication(), EmvCompletion(), ProcessCtlsEntryPoint() and ProcessCtlsCompletion()

```
typedef enum
{
      OUTCOME_NONE,
      OUTCOME_TRY_OTHER_INTERFACE,
      OUTCOME_OFFLINE_APPROVED,
      OUTCOME_OFFLINE_DECLINED,
      OUTCOME_OFFLINE_FAILED,
      OUTCOME_OFFLINE_NOT_ACCEPTED,
      OUTCOME_APPROVED,
      OUTCOME_DECLINED,
      OUTCOME_FAILED,
      OUTCOME_NOT_ACCEPTED,
      OUTCOME_CANCELLED,
      OUTCOME_SELECT_NXT_NOT_ACCEPTED,
      OUTCOME_SELECT_NXT_RETRY,
      OUTCOME_TRY_AGAIN,
      OUTCOME_ONLINE_REQUEST,
      OUTCOME_END_APPLICATION,
      OUTCOME_NOT_EMV_CARD_POOLED
}TRANS_OUTCOME;
```

6.2 CARD ENTRY POLLING OUTCOME

This is used as the return value from the CardEntryPolling() function

```
typedef enum
{
      CARD_ENTRY_NONE,
```

```
        CARD_ENTRY_MANUAL,
        CARD_ENTRY_SWIPE,
        CARD_ENTRY_INSERT,
        CARD_ENTRY_TAP,
        CARD_ENTRY_MIFARE_TAP
}CARD_ENTRY;
```

6.3 AGNOS TRANSACTION TYPE

This transaction type will be used as an input when calling the transaction initialization Initialize() function.

Note: A contactless transaction only supports AGN_PURCHASE, AGN_CASH, AGN_WITH_CASHBACK and AGN_REFUND.

```
typedef enum
{
        AGN_PURCHASE,
        AGN_CASH,
        AGN_WITH_CASHBACK,
        AGN_REFUND,
        AGN_MANUAL_CASH,
        AGN_QUASI_CASH,
        AGN_DEPOSIT,
        AGN_INQUIRY,
        AGN_PAYMENT,
        AGN_TRANSFER,
        AGN_ADMINISTRATIVE,
        AGN_HOUSE_KEEPING,
        AGN_RETRIEVAL,
        AGN_UPDATE,
        AGN_AUTHENTICATION,
        AGN_UNDEFINED
}AGN_TRANS_TYPE;
```

7 API REFERENCE – MISCELLANEOUS API FUNCTIONS

7.1 SET SUPPORTED CARD ENTRY

Prototype	`int SetSupprtdCardEntry(` `  unsigned int SupportedCardEntry);`
Function	SetSupprtdCardEntry() is used to set the card reader technology which needs to be detected by the CardEntryPolling() function.
Input	SupportedCardEntry – See the defined options as follows:

	//! Card Input Devices
	#define BASE_DEVCARD_MSR (1<<0) //-1
	#define BASE_DEVCARD_CTLS (1<<1) //-2
	#define BASE_DEVCARD_ICC (1<<2) //-4
	#define BASE_DEVCARD_MANUAL (1<<3) //-8
	#define
	BASE_DEVCARD_A(BASE_DEVCARD_MSR|BASE_DEVCARD_CTLS|BASE_DEVCARD_ICC|BASE_DEVCARD_MANUAL)

Output	NA
Return	Technology Polling defined in gpicad.h
Notes	NA

7.2 SET INTERNAL POLLING TIMEOUT

Prototype	int SetPollingTimeOut(const unsigned int CardEntryTimeOut);
Function	It sets the internal card polling timeout (in secs, default is 30sec, if not set) for the contactless entrypoint.
Input	CardEntryTimeOut – Specifies the timeout in seconds (s)
Output	NA
Return	AMP_SUCCESS
Notes	NA

7.3 CARD ENTRY POLLING

Prototype	int CardEntryPolling(const unsigned int CardEntryTimeOut, USHORT *AddlStatus);
Function	The CardEntryPolling() function can be used to detect the card readers for card entry. • CARD_ENTRY_MIFARE_TAP - Mifare Desfire contactless card • CARD_ENTRY_TAP - EMV contactless card • CARD_ENTRY_INSERT - EMV contact card • CARD_ENTRY_SWIPE - Magnetic Stripe card
Input	CardEntryTimeOut – Specifies the timeout in seconds (s)

Output	AddlStatus – Returns the keyed-in value during manual entry
Return	AMP_ERROR, AMP_TIMEOUT, AMP_CANCEL, CARD_ENTRY,
Notes	The CardEntryTimeOut is also used in an Entry Point's internal polling for the "Try Again" cases

7.4 MIFARE DESFIRE DETECTION

Prototype	`int SetMifareDetection(` `  BOOL bEnable);`
Function	SetMifareDetection() API enables the Mifare Desfire contactless card detection on the CardEntryPolling() function.
Input	bEnable = TRUE, to enable Mifare contactless card detection on the CardEntryPolling() function
Output	NA
Return	AMP_SUCCES
Notes	It can be also configured using the "EnableMifareDetect=Yes" configurable parameter defined in agnos.ini file

7.5 READ MAGNETIC STRIPE

Prototype	`int ReadMagstripe(`
	`  unsigned char *Track1, int *Track1Len,`
	`  unsigned char *Track2, int *Track2Len,`
	`  unsigned char *Track3, int *Track3Len);`

Function	The ReadMagstripe() function is used to read track data from the magnetic card.
Input	NA
Output	Track1, Track2, Track3 – Buffer to be filled-in by the function Track1Len, Track2Len, Track3Len – Returns the track data length
Return	AMP_SUCCESS, AMP_ERROR
Notes	NA

7.6 CLOSE DEVICE READER

Prototype	`int CloseGpiDeviceReader(` `  BYTE CardReader);`
Function	This function is used to close the specified card reader.

Input	CardReader options:
	`// EMV Card Reader Devices`
	`#define DEVICE_CARD_CONTACT 0x01`
	`#define DEVICE_CARD_CONTACTLESS 0x02`
	`#define DEVICE_CARD_ALL DEVICE_CARD_CONTACT | DEVICE_CARD_CONTACTLESS`

Output	NA
Return	AMP_SUCCESS
Notes	NA

7.7 GET EMV TAG VALUE

Prototype	`int EmvGetTag(`
	`  unsigned long TagName,`
	`  BYTE *value, int valuesize);`

Function	This function retrieves the value of the specified tag in the tag list collections. It searches the tag in the TLV stream data as a result of EMV contact and contactless processing.
Input	NA
Output	value – Returns the value of the specified tag valuesize – The size of the value buffer
Return	TLV Data Length
Notes	NA

7.8 GET EMV TAG COLLECTIONS

Prototype	`int GetEMVTlvData(`
	`  unsigned char *Buffer,`
	`  int BufferSize)`

Function	This function gets the TLV stream data as a result of the EMV contact and contactless processing.
Input	NA
Output	Buffer – Returns the filled-in value, which was set previously in SetTagListCollection() BufferSize – The buffer size
Return	TLV Data Length
Notes	NA

7.9 GET AID INFORMATION

Prototype	`int EmvGetAIDInfo(`
	`  BYTE *Buffer,`
	`  int BufferSize);`

Function	This function is used to get the AID information instead of using tag 4F. This interface was made available because the 4F tag is not included in the EMV kernel's tag collections.
Input	NA
Output	Buffer – Returns the Application Identifier (AID) of the EMV contact and contactless transaction BufferSize – The buffer size
Return	TLV Data Length, AMP_ERROR
Notes	NA

7.10 SET ALLOWED PIN LENGTH FOR OFFLINE PIN ENTRY

Prototype	`int SetExpectedPinLen (` `  const char* szMaxPINLength);`
Function	This function is used to set the allowed PIN lengths used during offline PIN Entry.
Input	szMaxPINLength – string list format with comma "," delimiter (ex: "4, 6, 12"). The minimum and maximum PIN lengths are 4 and 12

	respectively.
Output	NA
Return	AMP_SUCCESS, AMP_ERROR

Notes	Code Sample:
	`void AgnosCBDisplayPinEntryPrompt (unsigned char bClearScreen, unsigned char LangId, const char *szMessage)`
	`{`
	`int X, Y, Width, Height;`
	`int iTimeoutSec = 60;`
	`Width = PIN_ENTRY_DEF_WIDTH;`
	`Height = PIN_ENTRY_DEF_HEIGHT;`
	`X = PIN_ENTRY_DEF_X;`
	`Y = PIN_ENTRY_DEF_Y;`
	`//Set list of allowed PIN Length here`
	`cAMPAgnos *pAgnos = cAMPAgnos::get_Instance();`
	`pAgnos->SetExpectedPinLength("4,6,10");`
	`...`
	`}`

8 API REFERENCE – EMV CONTACT PROCESSING

The section covers the EMV contact API interfaces that will be used to construct the EMV transaction flow as described in Book 3 of EMVco Application Specification.

Refer to **Appendix B** for illustration and reference.

8.1 API INTERFACE

8.1.1 SET TAG LIST COLLECTION

Prototype	`int SetTagListCollection(`
	`BYTE *TagList,`
	`int TagListLen)`

Function	SetTagLisCollection() is used to setup the list of tag collections that will be filled-in through EMV contact and contactless processing
Input	TagList – The array of tags that will be filled-in after transaction

	completion TagListLen – The TagList length
Output	NA
Return	AMP_SUCCESS
Notes	NA

8.1.2 POWER ON THE CONTACT READER

Prototype	`int PowerOn(void);`
Function	This function is used to power ON the device contact reader. It returns ocTRY_ANOTHER_INTERFACE when the card is mute.
Input	NA
Output	NA
Return	AMP_SUCCESS, AMP_ERROR, ocTRY_ANOTHER_INTERFACE, ocEND_APPLICATION
Notes	NA

8.1.3 PERFORM SELECT APPLICATION

Prototype	`int EmvSelectApplication()`
Function	EMVSelectApplication() is used to perform the EMV application selection process as described in EMVco Book-1 ICC to Terminal Interface. It covers the building of the candidate list and performs the final selection. According to the INTERAC rules and the EMV standard, the final selection is generic for any selection processing worldwide. It will require a CBSelectItemFromList() callback if the application layer intends to override the built-in selection item display.
Input	NA
Output	NA
Return	TRANS_OUTCOME (see TRANS_OUTCOME enum values)

Notes	The actual final selection process can be overridden by the application layer through using the CBAppSelection() callback function (see **API Reference - Callback Functions**).

8.1.4 PERFORM SELECT LANGUAGE

Prototype	`int EmvSelectLanguage()`
Function	This function initiates language selection based on EMV Book-4 whereby a terminal supporting multiple languages shall compare the card's Language Preference with the languages supported by the terminal at the beginning of the transaction. It will require a CBSelectItemFromList() callback if the application layer intends to override the built-in selection item display.
Input	NA
Output	NA
Return	AMP_SUCCESS
Notes	The language selection process can be overridden by the application layer through using the CBLangSelection() callback function (see **API Reference - Callback Functions**).

8.1.5 SETUP TRANSACTIONAL CONTEXT

Prototype	`int EmvSetTransactionalContext()`
Function	This function has to be called to setup the transactional context extracted from the TERMINAL and PROCESSING configuration files which will be used to initiate contact kernel session.
Input	NA
Output	NA
Return	AMP_SUCCESS
Notes	NA

8.1.6 OPEN KERNEL SESSION

Prototype	`int EmvOpenKernelSession()`
Function	This opens a session and initializes the EMV kernel's data structure from a transaction context provided by the application level. Additionally, it resets internal parameters. The related EMV tags live until the session is explicitly closed.
Input	NA
Output	NA
Return	AMP_ERROR, AMP_SUCCESS
Notes	NA

8.1.7 PERFORM INITIATE APPLICATION

Prototype	`int EmvInitiateApplication()`
Function	This function begins actual EMV processing from Initiate Application, GPO, Read Application Data, Data Authentication, Processing Restrictions, CVM Processing, up to the 1st GenAC.
Input	NA
Output	NA
Return	TRANS_OUTCOME (see TRANS_OUTCOME enum values)
Notes	The application layer should implement the procedures for PIN Entry that is needed for card verification through CBDisplayMaskedPin(), CBDisplayPinEntryPrompt() and CBOnlinePinEntry() callbacks (refer to **API Reference - Callback Functions**) The tag collections information can be extracted through GetEMVTlvData() function defined in AMPAgnos.h

8.1.8 SET ONLINE RESPONSE

Prototype	`int EmvSetOnlineResponse(`
	`  unsigned char uARC[2],`
	`  unsigned char uCID,`
	`  const unsigned char *IssuerResponse,`
	`  int IssuerRespLen);`

Function	This sets the online host response needed prior to calling the EMVCompletion() function.

Input	uARC – The authorization response code
	uCID – The cryptogram Information data (CID)
	IssuerResponse – An issuer response data from the host
	IssuerRespLen – The length of the IssuerResponse data
	`//! ARC Response`
	`#define UNABLE_TO_GO_ONLINE "\x50\x50"`
	`#define TECHNICAL_ISSUE "\x60\x60"`
	`#define ISSUER_REFERRAL "\x30\x31"`
	`#define AUTH_ACCEPTED "\x30\x30"`
	`#define AUTH_DECLINED "\x35\x31"`
	`//! (CID) Cryptogram Information Data`
	`#define ARQC 0x80`
	`#define TC 0x40`
	`#define AAC 0x00`
	`#define UNDEFINED_CID 0xF7`

Output	NA
Return	AMP_SUCCESS

Notes	Sample Usage:
	`if( hostOnlineProcess() == HOSTCOMM_SUCCESS) //comms success`
	`{`
	`if(TransOnlineApproved())//host approved`
	`{`
	`uCID = TC;`
	`memcpy(uARC, AUTH_ACCEPTED, 2);`
	`}`
	`else //host declined`
	`{`
	`uCID = AAC;`
	`memcpy(uARC, AUTH_DECLINED, 2);`
	`}`
	`//Issuer Response`
	`IssuerRespLen = getHostIssuerResponse (IssuerResponse);`
	`}`
	`else //Unable to go online`
	`{`
	`memcpy(uARC, UNABLE_TO_GO_ONLINE, 2);`
	`// y1, z1, y3, z3`

	}
	//Set Online Response
	pAgnosEmv->EmvSetOnlineResponse(uARC, uCID, IssuerResponse, IssuerRespLen);

8.1.9 PERFORM EMV COMPLETION

Prototype	int EmvCompletion()
Function	The EMVCompletion() function performs 2nd GenAC, and extracts Tag 91, 71, and 72 from the issuer response, executes issuer script, and authenticates the issuer. When unable to go online, it calculates a CID to be provided to ICC and initializes EMV tag 8A.
Input	NA
Output	NA
Return	TRANS_OUTCOME (see TRANS_OUTCOME enum values)
Notes	NA

8.1.10 CLOSE KERNEL SESSION

Prototype	int EmvCloseKernelSesssion()
Function	It closes kernel session and closes the contact reader.
Input	NA
Output	NA
Return	AMP_SUCCESS
Notes	NA

8.2 INCLUDE HEADER LIBRARY

The EMV contact application program interfaces are defined in header/s:

- AMPAgnosEmv.h

8.3 BASIC FLOW

Here is the basic EMV contact transaction flow.

1) Amount Entry()

2) InitTrans()

3) CardEntryPolling()

4) OutCome= DoEmvInitiateFlow()

5) If Outcome=Offline Approved, Remove Card, then End Transaction

6) If Outcome=Requires Online

   a) OnlineProcessing()
   b) Set EmvSetOnlineResponse()
   c) OutCome= DoEmvCompletion()
   d) Remove Card

8.4 CODE SAMPLE

Here is the sample code implementation that demonstrates the EMV contact processing.

```
//Set up Tag List
static BYTE gTagList[] = {
0x9F, 0x39, 0x9F, 0x27, 0x9F, 0x1A, 0x9A, 0x9F, 0x26, 0x82, 0x9F, 0x36,
0x9F, 0x37, 0x95, 0x9B, 0x9C, 0x5F, 0x2A, 0x5F, 0x57, 0x9F, 0x41, 0x9F,
0x4E, 0x9F, 0x02, 0x9F, 0x03, 0x9F, 0x10, 0x5F, 0x34, 0x9F, 0x35, 0x9F,
0x34, 0x9F, 0x09, 0x84, 0x5A, 0x99, 0x9F, 0x1C, 0x9F, 0x1E, 0x9F, 0x15,
0x9F, 0x16, 0x4F, 0x9F, 0x12, 0x50, 0x5F, 0x30, 0x5F, 0x20, 0x5F, 0x24,
0x9F, 0x33, 0x5F, 0x25, 0x5F, 0x28, 0x57, 0x9F, 0x6E, 0x9F, 0x01, 0x9F,
0x53, 0x9F, 0x6D,0x8A
};


 int DoEmvInitiateFlow(void)
 {
       int iRetVal = AMP_SUCCESS;
       cAMPAgnosEmv *pAgnosEmv = cAMPAgnosEmv::get_Instance();

       //1. Setup Tag List
       pAgnosEmv->SetTagListCollection( gTagList, sizeof(gTagList));

       //2. Initiliaze EMV Agnos Data
       pAgnosEmv->InitTrans();
```

```
 //3. Power ON Card Reader
 iRetVal = pAgnosEmv->PowerOn();

 //3.b Close Contactless reader for touchscreen to work
 pAgnos->CloseGpiDeviceReader(DEVICE_CARD_CONTACTLESS);

 //4. Select Application
 iRetVal = pAgnosEmv->EmvSelectApplication(); //returns outcome

 //4.b Evaluate Outcome
 if(iRetVal == ocNOT_ACCEPTED || iRetVal == ocTRY_ANOTHER_INTERFACE
 || iRetVal == ocTRANSACTION_CANCELLED )
 {
      pAgnosEmv->RemoveChipCard();
      BeepWarning();

      if(iRetVal == ocTRANSACTION_CANCELLED)
            iRetVal = AMP_CANCEL;
      else if(iRetVal == ocNOT_ACCEPTED)
            iRetVal = AMP_ERROR;

            return iRetVal;
 }
//5. Select Language
iRetVal = pAgnosEmv->EmvSelectLanguage(); //retuns AMP_SUCCESS

//6. Setup Agnos Transactional Context
iRetVal    =    pAgnosEmv->EmvSetTransactionalContext();    //returns
AMP_SUCCESS

//7. Open Agnos Kernel Session
iRetVal = pAgnosEmv->EmvOpenKernelSession(); //returns AMP_SUCCESS,
AMP_ERROR

//8. Set EMV Mandatory Tags
iRetVal = pAgnosEmv->EmvSetMandatoryTags(); //returns AMP_SUCCESS

//9. Initiate Application
iRetVal = pAgnosEmv->EmvInitiateApplication(); //returns outcome

//9.b Evaluate Outcome
switch(iRetVal)
{
case OUTCOME_ONLINE_REQUEST:
      break;
case OUTCOME_OFFLINE_APPROVED:
      break;
}
}
```

```
int DoEmvCompletion(void)
{
int iRetVal = AMP_SUCCESS;
unsigned char uCID;
unsigned char uARC[2] ={0};
unsigned char IssuerResponse[ISSUER_RESPONSE_SIZE]={0};
int IssuerRespLen = 0;
cAMPAgnosEmv *pAgnosEmv = cAMPAgnosEmv::get_Instance();

//Setup uARC, uCID, IssuerResponse from the host
if(hostresponse == online_approved) //test online approved
{
uCID = TC; //set by host for approve
memcpy(uARC, AUTH_ACCEPTED, 2);
IssuerRespLen = getHosIssuerResp(IssuerResponse);

}
else if( hostresponse == online_declined)
{
uCID = AAC; //set by host for declined
memcpy(uARC, AUTH_DECLINED, 2);
IssuerRespLen = getHosIssuerResp(IssuerResponse);
}
else if(hostresponse == unable_to_go_online) // y1, z1, y3, z3
{
      memcpy(uARC, UNABLE_TO_GO_ONLINE, 2);
}

//1. Set Online Response
pAgnosEmv->EmvSetOnlineResponse(    uARC,    uCID,    IssuerResponse,
IssuerRespLen);

//2. Process Completion
iRetVal = pAgnosEmv->EmvCompletion();

//3. Evaluate OutCome
switch(iRetVal)
{
case OUTCOME_OFFLINE_APPROVED:
case OUTCOME_APPROVED:
      EmvCardDecision = EMV_APPROVED;
      iRetVal = AMP_SUCCESS;
      BeepInfo();
      break;
case OUTCOME_OFFLINE_DECLINED:
case OUTCOME_DECLINED:
case OUTCOME_NOT_ACCEPTED:
case OUTCOME_OFFLINE_FAILED:
```

```
case OUTCOME_END_APPLICATION:
case OUTCOME_FAILED:
case OUTCOME_NOT_EMV_CARD_POOLED:
case OUTCOME_CANCELLED:
default:
      EmvCardDecision = EMV_DECLINE;
      iRetVal = AMP_ERROR;
      break;
}

//4. Close EMV Kernel session
pAgnosEmv->EmvCloseKernelSession();

//5. Remove card
pAgnosEmv->RemoveChipCard();

return iRetVal;
}
```

9 EMV QUICK CHIP

This section delves into the EMV Quick Chip functionality.

9.1 QUICK CHIP NOTES

- Quick Chip is suitable for the "Online Only" (Terminal Type=21) terminal configuration as the transaction must be Authorize Online.
- The payment application will force the Terminal Floor Limit (Tag 9F1B) to zero (0) and the TAC-Online B4b1 Transaction Exceeds Floor Limit is set to one (1) to obtain ARQC
- The Quick Chip configuration shall be set per AID. The Quick Chip AIDs should be listed in the QuickChip.cfg file
- Issuer Authentication is not necessary for Quick Chip transaction; hence, the payment application shall discard any Issuer Authentication Data or Issuer Scripts in the authorization response
- Card Removal shall be prompted before Online Authorization processing

9.2 API INTERFACE

9.2.1 ENABLING QUICK CHIP

Prototype	`int EmvSetEnableQuickChip(` `  BOOL bEnableFlag )`
Function	This API is used to enable EMV Quick Chip transaction. All Application Identifiers (AIDs) listed in QuickChip.cfg will execute EMV

	Quick Chip payment flow.
Input	bEnableFlag – TRUE, to enable Quick Chip
Output	NA
Return	AMP_SUCCESS
Notes	An alternative way for enabling Quick Chip is through setting the configurable parameter "QuickChip=Yes" defined in the agnos.ini file **QuickChip.cfg** A0000000031010 A0000000041010 A000000025010801

9.2.2 CHECKING IF QUICK CHIP TRANSACTION

Prototype	`BOOL EMVIsQuickChipTrans(void);`
Function	This API can be used to determine if the transaction performed was an EMV Quick Chip or an EMV standard transaction.
Input	NA
Output	NA
Return	TRUE – Quick EMV Chip transaction FALSE – EMV transaction
Notes	NA

10 US DEBIT APPLICATION SELECTION

This section describes the US Debit Automatic Application Selection functionality.

10.1 US DEBIT APPLICATION NOTES

10.1.1 CONFIGURATION (AGNOS.INI)

Here is the list of configurable parameters defined in the agnos.ini file that are being used for U.S. Debit Selection

- USDebitAppSelect – A configurable parameter that is used to enable/disable the US Debit Application Selection feature

    - Yes – Enables US Debit (default)
    - No – Disables the US Debit feature
- USDebitModeCT – The US Debit application selection methods for EMV contact

    - 0 – EMV standard selection
    - 1 – Automatic US Debit AID selection
    - 2 – Manual US Debit Selection Process (default), see also "USDebitAIDListMode" parameter

- USDebitModeCTLS – US Debit application selection methods for EMV contactless

    - 0 – EMV contactless standard selection where the highest application priority indicator (Tag 87) shall be automatically selected (default)
    - 1 – Automatic US Debit selection
- USDebitAIDListMode – A configurable parameter that allows the merchant to select whether the US Debit or the Global Debit applications will be shown to the cardholder when the funds (IIN values) are the same

    - 0 – All
    - 1 – Retains US Debit and eliminates Global Debit in the candidate list (default)
    - 2 – Eliminates US Debit and retains Global Debit in the candidate list

Note: This is only applicable when USDebitModeCT is set to "2" – Manual US Debit Selection Process.

10.1.2 USDEBIT.CFG

A configuration file that contains the list of US Debit Application Identification (AID). This will be used in the US Debit Selection Process to identify if the AID is a US Common debit.

- A0000000042203
- A0000000980840

10.1.3 EMV CONTACT US DEBIT SELECTION METHODS

- Automatic US Debit Selection – POS solution should always select the US Common Debit AID when present.

- Manual US Debit Selection Process – If AIDs in the candidate list have the same Tag5f55=US and IIN, POS should allow the merchant to choose either "Global Debit" or "US Common Debit" based on their preferred routing choice.

    Points to remember:

- EMV POS solution must NOT assume that if a U.S. Common Debit AID is present then this AID can be automatically selected
- Issuer Country Code (Tag 5F55) with value of 0x5553 (US)
- Issuer Identification Number (IIN) (Tag 42)
- The transaction limits that will be evaluated to set the CVM configuration
  - NO CVM Limit (callback function)
  - CVM Limit (callback function)
- Credit/Debit Selection prompt (callback function)
- CVM configuration in Terminal Capabilities (Tag 9F33 Byte-2)
  - All CVM Config – to support all CVM including NO CVM (Byte-2 = "F8")
  - No CVM Only Config – to support only NO CVM (Byte-2 = "F8")

1) Callbacks

The application should prepare new callbacks in order for AMPEMVL2 to correctly process the Manual US Debit Selection

```
typedef struct
{
      int (*CBAppSelection) (tADFList *mutualList, BYTE *MLIndex,
            const tTerminalContext *terminal, BOOL CLMode);
      int (*CBLangSelection) (unsigned char LangId, unsigned char
            StringId, int TimeOut, unsigned char *SelectedId);
      unsigned short (*CBDisplayMessage) (unsigned char
            bClearScreen, unsigned char LangId, const char
            *szMessage, unsigned char MessageId);
      unsigned short   (*CBSelectItemFromList) (const char
            *arrItems[], unsigned char arrCount, const char *Title,
            int TimeOut, unsigned char *SelectedId);
      unsigned short   (*CBGpiSwitchLED) (unsigned char Led,
            unsigned char Flag, unsigned char Colour);
      void (*CBDisplayPinEntryPrompt) (unsigned char bClearScreen,
            unsigned char LangId, const char *szMessage);
      unsigned short (*CBOnlinePinEntry) (const char *PAN, int
            PANlen, unsigned char *PinBlock, int *PinBlockLen);
      void (*CBPinRetries) (int PinRetries);

      unsigned long (*CBGetCvmLimit) (const BYTE *byAid, BYTE
            AidLen);
      unsigned long (*CBGetNoCvmLimit) (const BYTE *byAid, BYTE
            AidLen);
      int   (*CBSelectDebitCredit)(const BYTE *byAid, BYTE AidLen);

      #if defined(ANDROID)
      unsigned short (*CBOfflinePinEntry) (char bEnciphered,
            unsigned int lan, int timeout, tEMVPubKey* pubkey,
            unsigned char *IccRandom, unsigned char *RespData);
      #endif
```

```
}AMPAGNOS_CB;
```

2) Logic

1- Start from the Candidate List

2- For each AID in the Candidate List,

3- If the AID has Tag5F55 (CountryCode) and Tag42 (IIN) proceed to Step-4, otherwise go to Step-7

4- If Tag5F55 is 0x5553 (US) proceed to Step-5, otherwise go to Step-7

5- Check AID if it has the same IIN (Tag42) value with another AID

6- If AID has same IIN, select the AID that should be eliminated, either “US Debit” or “Global Debit”

- Check the USDebitAIDListMode value in agnos.ini
  - 1- retains US Debit and eliminates Global Debit in the candidate list
  - 2- eliminates US Debit and retains Global Debit in the candidate list
- If the AID is listed in the USDebit.cfg, this means the AID is “US Debit”, otherwise a “Global Debit”

7- If the AID end record in the Candidate List, proceed to Step-8, otherwise next AID, go to Step-3

8- If only one AID is left in the Candidate List, continue to Step-9, else go to Step-12

9- If Amount < CVM_LIMIT, set Tag9F33 (Terminal Capabilities) to “No CVM Only Config”, then go to Step-12, else proceed to Step-10

10- If Amount >= NO_CVM_LIMIT, ask for Debit/Credit selection

11- If Debit was selected set Tag9F33 (Terminal Capabilities) to “All CVM Config”, otherwise set to “No CVM Only Config”

12- Display the Final Candidate List for cardholder AID selection

13- If AID selected is US Debit proceed to Step-14, otherwise go to Step-15

14- If the Amount >=NO_CVM_LIMIT, set Tag9F33 (Terminal Capabilities) to “All CVM Config”, otherwise set to “No CVM only Config”

15- End Manual US Debit Selection Process

10.1.4 EMV CONTACTLESS US DEBIT SELECTION METHOD

- Automatic Selection by Application Priority Indicator – EMV POS Solution should automatically select the AID in the Candidate List with the highest Application Priority Indicator (Tag 87).

- Automatic US Debit Selection – the POS solution will select the U.S. Common Debit AID when present.

10.2 API INTERFACE

10.2.1 ENABLING U.S. DEBIT AUTOMATIC APPLICATION SELECTION

Prototype	`int EmvSetEnableUSDebitFinalAppSelect(` `  BOOL bEnableFlag )`
Function	This API is used to enable US Debit Auto Selection. All Application Identifiers (AIDs) listed in USDebit.cfg will be treated as US Common Debit AID that will be auto selected.
Input	bEnableFlag – TRUE, to enable US Debit Auto Selection
Output	NA
Return	AMP_SUCCESS
Notes	NA

10.2.2 MODE SETTING OF U.S. DEBIT AUTOMATIC APPLICATION SELECTION

Prototype	`int EmvSetUSDebitMode(`
	`  BOOL bCTLS,`
	`  unsigned char Mode )`

Function	This API can be used to set the mode to be applied for US Debit Auto Selection.
Input	bCTLS – Indicates a contactless transaction Contact: (bCTLS=FALSE) Mode=0 – EMV Standard Selection Mode=1 – US Debit AID will be automatically selected Mode=2 – Manual US Debit selection process

	Contactless: (bCTLS=TRUE) Mode=0 – Highest priority will be automatically selected Mode=1 – US Debit AID will be automatically selected
Output	NA
Return	AMP_SUCCESS
Notes	NA

10.2.3 LIST OF AID TO BE DISPLAYED TO THE CARDHOLDER

Prototype	`EmvSetUSDebitAIDListMode( unsigned char Mode )`
Function	This API is used to select either the US Debit or the Global Debit applications will be shown to the cardholder when the funds (IIN values) are the same.
Input	Mode=0 – All Mode=1 – Retail "US Debit" and eliminate "Global Debit" in the candidate list Mode=2 – Eliminate "US Debit" and retail "Global Debit" in the candidate list
Output	NA
Return	AMP_SUCCESS
Notes	This is only applicable when USDebitModeCT is set to "2" – Manual US Debit Selection Process

11 API REFERENCE – EMV CONTACTLESS PROCESSING

This section covers the API interface for EMV contactless card transactions

Refer to **Appendix B** for illustration and reference.

11.1 API INTERFACE

11.1.1 PROCESS ENTRY POINT

Prototype	`int ProcessCtlsEntryPoint()`
Function	This function is basically the Entry Point as described in EMVco Contactless Book-A. It is responsible for the pre-processing, discovery and selection of a contactless application which is supported by both the card and the reader, in addition to the appropriate kernel activation.
Input	NA
Output	NA
Return	TRANS_OUTCOME (refer to TRANS_OUTCOME enum values)
Notes	When a kernel provides an OUTCOME, the control is passed back to Entry Point which immediately handles certain parameters.

11.1.2 PROCESS COMPLETION

Prototype	`int ProcessCtlsCompletion()`
Function	This function is used to execute Entry Point with STARTD entry. STARTD is used when the Entry Point is restarted within a transaction in order to process the data provided in response to an online request, for risk management purposes.
Input	NA
Output	NA
Return	TRANS_OUTCOME (refer to TRANS_OUTCOME enum values)
Notes	NA

11.1.3 LED STATUS DISPLAY FOR ERROR OUTCOME

Prototype	`int ErrorLEDDisplay(` `  int Option)`
Function	It can be used to light the LED for Card read ERROR scenarios as described in Book A EMVco under the Common User Interface

	Guidelines for Card Read Processing Error.
Input	Option – Indicates option 1 or 2 Where: 1= LED-1 to LED-4 are OFF 2= LED-1 to LED-3 are OFF, LED-4 is RED
Output	NONE
Return	AMP_SUCCESS
Notes	NA

11.2 DEFAULT TAG LIST COLLECTION OF CONTACTLESS PROCESSING

After contactless processing ProcessCtlsEntryPoint() or ProcessCtlsCompletion(), a TLV stream data will be generated based on the default internal tag list listed below. This occurs when the tag list collection is not set in the application layer.

```
// PayWave Data Record
static const tByte EMVDR[] =
{0x9F,0x02,0x9F,0x03,0x9F,0x26,0x82,0x9F,0x36,0x5F,0x34,0x9F,0x7C,0x9F,0x6
E,0x9F,0x10,
0x9F,0x39,0x9F,0x33,0x9F,0x1A,0xDF,0X17,0x95,0x5F,0x2A,0x9A,0x9C,0x9F,0x37
,0x57,0x9F,
0x06,0x9F,0x5D,0x9F,0x27,0x9F,0x66,0x9F,0x21,0x5A,0x5F,0x20,0x5F,0x24};

// PayPass Data Record
static const tByte EMVDR[] =
{0x9F,0x39,0x9F,0x02,0x9F,0x03,0x9F,0x26,0x5F,0x24,0x82,0x50,0x5A,0x5F,0x3
4,0x9F,0x12,
0x9F,0x36,0x9F,0x09,0x9F,0x27,0x9F,0x34,0x84,0x9F,0x1E,0x9F,0x10,0x9F,0x11
,0x9F,0x33,
0x9F,0x1A,0x9F,0x35,0x95,0x57,0x9F,0x53,  0x5F,0x2A,0x9A,0x9C,0x9F,0x37};

// ExpressPay Data Record
static const tByte EMVDR[] =
{0x9F,0x39,0x50,0x56,0x57,0x5A,0x84,0x95,0x9A,0x9C,0x82,0x9F,0x02,0x9F,0x0
3,0x9F,0x26,
0x9F,0x27,0x9F,0x10,0x5F,0x24,0x5F,0x34,0x9F,0x36,0x9F,0x37,0x9F,0x09,0x9F
,0x6D,0x9F,
0x34,0x9F,0x1E,0x9F,0x33,0x9F,0x1A,0x9F,0x35,0x9F,0x6B,0x5F,0x2A,0x9F,0x16
,0x9F,0x70,
0x9F,0x6E};
```

```
// DPass Data Record
static const tByte EMVDR[] =
{0x9F,0x02,0x9F,0x03,0x9F,0x26,0x82,0x9F,0x36,0x5F,0x34,0x9F,0x7C,0x9F,0x6
E,0x9F,0x10,
0x9F,0x39,0x9F,0x33,0x9F,0x1A,0x95,0x5F,0x2A,0x9A,0x9C,0x9F,0x37,0x57,0x9F
,0x06,0x9F,
0x5D,0x9F,0x27,0x9F,0x66,0x9F,0x21,0x5A,0x5F,0x20,0x5F,0x24};
```

11.3 INCLUDE HEADER LIBRARY

The EMV contactless application program interfaces are defined in the header/s:

- AMPAgnosEmvCtls.h

11.4 BASIC FLOW

1) Amount Entry()
2) InitTrans()
3) CardEntryPolling()
4) OutCome=ProcessCtlsEntryPoint()
5) If Outcome=Offline Approve, End Transaction
6) If Outcome=Requires Online
   a) OnlineProcessing()
   b) Set EmvSetOnlineResponse()
   c) OutCome=ProcessCtlsCompletion()

11.5 CODE SAMPLE

```
void OnTap()
{
TRANS_OUTCOME OutCome= OUTCOME_NONE;
OutCome = DoEmvCtlsInitiateFlow();
}

TRANS_OUTCOME DoEmvCtlsInitiateFlow()
{
      //Get CTLS instance
cAMPAgnosEmvCtls *pAgnosEmvCtls = cAMPAgnosEmvCtls::get_Instance();

//1. Setup Tag List
```

```
pAgnosEmvCtls->SetTagListCollection( gTagList, sizeof(gTagList));

//2. Contactless Init Trans
iRetVal = pAgnosEmvCtls->InitTrans();

//3. Process Entry Point
iRetVal = pAgnosEmvCtls->ProcessCtlsEntryPoint();

//4. Evaluate Outcome
switch(iRetVal)
{
case ocNOT_EMV_CARD_POOLED: //other contactless card
      break;
case ocTRY_ANOTHER_INTERFACE: //To fallback?
      BeepWarning();
      break;
case ocTRANSACTION_CANCELLED:
      iRetVal = AMP_CANCEL;
      BeepWarning();
      break;
case ocONLINE_REQUEST:
      EmvCardDecision = EMV_GO_ONLINE;
      break;
case ocOFFLINE_APPROVED:
case ocAPPROVED:
      EmvCardDecision = EMV_APPROVED;
      BeepInfo();
      break;
case ocOFFLINE_DECLINED:
case ocDECLINED:
case ocNOT_ACCEPTED:
case ocOFFLINE_FAILED:
case ocEND_APPLICATION:
case ocFAILED:
default:
      EmvCardDecision = EMV_DECLINE;
      iRetVal = AMP_ERROR;
      BeepError();
      break;
}
}
```

12 API REFERENCE - CALLBACK FUNCTIONS

12.1 CALLBACK API FUNCTIONS

12.1.1 APPLICATION SELECTION CALLBACK

Prototype	`int (*CBAppSelection) (`
	`  tADFList *mutualList,`
	`  BYTE *MLIndex,`
	`  const tTerminalContext *terminal,`
	`  BOOL CLMode);`

Function	A callback function that allows the application layer to override the library's built-in Final Selection Application process.
Input	mutualist – Holds the Mutual AID list for display and selection terminal – Holds the terminal parameters defined in the TERMINAL config CLMode – Indicates if the contactless mode is applied
Output	MLIndex – The selected index number in the Mutual List
Return	payTRANSACTION_TERMINATED, payTRANSACTION_CANCELLED, payNO_ERROR
Notes	NA

12.1.2 LANGUAGE SELECTION CALLBACK

Prototype	`int (*CBLangSelection) (`
	`  unsigned char LangId,`
	`  unsigned char StringId,`
	`  int TimeOut,`
	`  unsigned char *SelectedId);`

Function	A callback function that allows the application layer to override the library's built-in Language Selection process.
Input	LangId – The current language ID StringID – Title display ID (LANGUAGE = 0x83) TimeOut – The list display timeout (in seconds)
Output	SelectedId – The index number of the selected language referencing lang.ini

Return	payNO_ERROR
Notes	NA

12.1.3 DISPLAY ITEM LIST CALLBACK

Prototype	`unsigned short (*CBSelectItemFromList) (`
	`const char *arrItems[],`
	`unsigned char arrCount,`
	`const char *Title,`
	`int TimeOut,`
	`unsigned char *SelectedId);`

Function	A callback function that allows the application layer to override the library's built-in display of selection item list menu that is being used during Application Selection and Language Selection processing.
Input	arrItems – Holds the list of string items arrCount – The count of items in arrItems Title – Title of the list display Timeout – Display timeout
Output	SelectedId – The selected index item in arrItems (Base-0)
Return	lstCANCEL, lstNO_ERROR
Notes	NA

12.1.4 DISPLAY MESSAGE CALLBACK

Prototype	`unsigned short (*CBDisplayMessage)(`
	`unsigned char bClearScreen,`
	`unsigned char LangId,`
	`const char *szMessage,`
	`unsigned char MessageId);`

Function	A callback function that overrides the built-in screen display of the message. This allows the application layer to control the appearance of the displayed message during EMV contact and contactless processing.
Input	bClearScreen – Indicator to clear the screen LangId – The message's language Id

	szMessage – The message to be prompted on the screen display MessageId – The equivalent MessageId of the szMessage that is defined in gpidisplay.h while being used in lang.ini. A FF value means there is no equivalent MessageId for the input szMessage
Output	NA
Return	admNO_ERROR
Notes	The AMPEMVL2 library messages are based on the standard messages described in EMVco Book 4. MessageId defined in gpiddisplay.h

```
//------------------------------------------------------
------
//                Definitions
//------------------------------------------------------
------
//See standard messages, EMVCo Book IV, section 11.2
#define     NO_MESSAGE                      0x00
#define     AMOUNT                          0x01
#define     AMOUNT_OK                       0x02
#define     APPROVED                        0x03
#define     CALL_YOUR_BANK                  0x04
#define     CANCEL_OR_ENTER                 0x05
#define     CARD_ERROR                      0x06
#define     DECLINED                        0x07
#define     ENTER_AMOUNT                    0x08
#define     ENTER_PIN                       0x09
#define     INCORRECT_PIN                   0x0A
#define     INSERT_CARD                     0x0B
#define     NOT_ACCEPTED                    0x0C
#define     PIN_OK                          0x0D
#define     PLEASE_WAIT                     0x0E
#define     PROCESSING_ERROR                0x0F
#define     REMOVE_CARD                     0x10
#define     USE_CHIP_READER                 0x11
#define     USE_MAGSTRIPE                   0x12
#define     TRY_AGAIN                       0x13

#define     WELCOME                         0x14
#define     PRESENT_CARD                    0x15
#define     PROCESSING                      0x16
#define     CARD_READ_OK                    0x17
#define     PLEASE_INSERT_OR_SWIPE          0x18
#define     PLEASE_PRESENT_ONE_CARD         0x19
#define     APPROVED_PLEASE_SIGNED          0x1A
```

```
#define     AUTHORIZING                     0x1B
#define     ERROR_USE_OTHER_CARD            0x1C
#define     PRESENT_CARD_AGAIN              0x1D
#define     CLEAR_DISPLAY                   0x1E
//#define   RFU                             0x1F
#define     SEE_PHONE                       0x20
#define     RETRY                           0x21
#define     INSERT_SWIPE_OR_OTHER_CARD      0x22

#define     SELECT_ACCOUNT                  0x81
#define     USING_MAGSTRIPE                 0x82
#define     LANGUAGE                        0x83
#define     TRANSACTION_CANCELLED           0x84
#define     SWIPE_CARD                      0x85
#define     TRANSACTION_FAILED              0x86
#define     CASH_BACK                       0x87
#define     CASH_BACK_OK                    0x88
#define     REFERRAL                        0x89
#define     THANKS                          0x8A
#define     TXN_NOT_PERMITTED               0x8B
#define     SELECT                          0x8C
#define     DEFAULT_ACCOUNT                 0x8D
#define     SAVINGS_ACCOUNT                 0x8E
#define     CHECKING_ACCOUNT                0x8F
#define     CREDIT_ACCOUNT                  0x90
```

12.1.5 PIN ENTRY CALLBACKS

Prototype	`void (*CBDisplayPinEntryPrompt)(`
	`  unsigned char bClearScreen,`
	`  unsigned char LangId,`
	`  const char *szMessage);`

Function	This callback is used to customize the actual PIN entry prompt message where the application layer can add information freely to the display like the total amount presented on screen. This is used for Offline PIN entry.
Input	bClearScreen – An indicator to clear the screen LangId – Language ID szMessage – “Enter PIN” message
Output	NA
Return	NA

Notes	NA

Prototype	`unsigned short (*CBOnlinePinEntry) (`
	`  const char *PAN,`
	`  int PANlen,`
	`  unsigned char *PinBlock`
	`  int *PinBlockLen);`

Function	This callback is mandatory when using the AMPEMVL2 library for EMV contact processing. It is expected that the application layer will take care of the Online PIN encryption through the secured PED module.
Input	PAN – The PAN used for encryption PANlen – PAN length
Output	PinBlock – Output encrypted PIN PinBlockLen – PIN Block length
Return	sepNO_ERROR, sepTIME_OUT, sepCANCEL
Notes	NA

Prototype	`void (*CBPinRetries) (int PinRetries);`
Function	This callback can be used by the application layer to get the remaining PIN retries during Offline PIN Entry.
Input	PinRetries – The number of remaining retries
Output	NA
Return	void
Notes	NA

12.1.6 CONTACTLESS LED INDICATORS CALLBACK

Prototype	`unsigned short (*CBGpiSwitchLED) (`
	`  unsigned char Led,`
	`  unsigned char Flag,`
	`  unsigned char Colour);`

Function	Contactless kernel invokes the GpiSwitchLED() callback which allows the application layer to manage the LED indicators to display their status during contactless transaction processing.
Input	Led – LED index Flag – Status Colour - Color
Output	NA
Return	adm_NO_ERROR

Notes	`// LED Definitions`
	`//LED index`
	`#define LED_1 0x01`
	`#define LED_2 0x02`
	`#define LED_3 0x03`
	`#define LED_4 0x04`
	`#define LED_ALL_ID 0x05`
	
	`//Status Flag`
	`#define LED_OFF 0x00`
	`#define LED_ON 0x01`
	`#define LED_FLASH 0x02`
	
	`//Colour`
	`#define LED_NONE 0x00`
	`#define LED_BLUE 0x01`
	`#define LED_YELLOW 0x02`
	`#define LED_RED 0x03`
	`#define LED_GREEN 0x04`
	`#define LED_ALL_COLOR 0x05`

12.1.7 U.S. DEBIT CVM LIMIT CALLBACKS

Prototype	`unsigned long (*CBGetNoCvmLimit) (`
	`const BYTE *byAid,`
	`BYTE AidLen);`

Function	This is used to pass the NO CVM LIMIT of the given AID from the application layer to AMPEMVL2 for the Manual U.S. Debit Selection Process.
Input	byAid – The card's AID information AidLen – The AID data length

Output	NA
Return	NO CVM LIMIT amount
Notes	See the **US Debit Application Selection**

Prototype	`unsigned long (*CBGetCvmLimit) (`
	`  const BYTE *byAid,`
	`  BYTE AidLen);`

Function	This is being used to pass the CVM LIMIT of the given AID from the application layer to AMPEMVL2 for the Manual U.S. Debit Selection Process.
Input	byAid – The card's AID information AidLen – The AID data length
Output	NA
Return	The CVM Limit amount
Notes	See the **US Debit Application Selection**

12.2 CALLBACK REGISTRATION

Prototype	`void RegisterCallback(`
	`  AMPAGNOS_CB *callback);`
	
	`typedef struct`
	`{`
	`  int (*CBAPPSelection) (...);`
	`  int (*CBLanguageSelection) (...);`
	`  unsigned short (*CBDisplayMessage)(...);`
	`  unsigned short (*CBSelectItemFromList) (...);`
	`  unsigned short (*CBGpiSwitchLED) (...);`
	`  void (*CBDisplayPinEntryPrompt) (...);`
	`  unsigned short (*CBOnlinePinEntry) (...);`
	`  void (*CBPinRetries) (...);`
	`  unsigned long (*CBGetCvmLimit) (...);`
	`  unsigned long (*CBGetNoCvmLimit) (...);`
	`  int (*CBSelectDebitCredit) (...);`
	`}AMPAGNOS_CB;`

Function	The AMPAGNOS_CB callback structure can be filled in with the application layer's equivalent function pointer members in order to

	call these functions rather than the built-in library functions.
Input	NA
Output	NA
Return	NA

Notes	`static AMPAGNOS_CB gAgnosCB =`
	`{`
	`NULL, //myCBAppSelection,`
	`NULL, //myCBLangSelection,`
	`myCBDisplayMessage,`
	`myCBSelectItemFromList,`
	`myCBGpiSwitchLED,`
	`myCBDisplayPinEntryPrompt,`
	`myCBOnlinePinEntry,`
	`myCBPinRetries,`
	`myCBGetCvmLimit,`
	`myCBGetNoCvmLimit,`
	`myCBSelectDebitCredit`
	`};`
	
	`cAMPAgnos *pAgnos = cAMPAgnos::get_Instance();`
	`//Register Callback`
	`pAgnos->RegisterCallback(&gAgnosCB);`

12.3 INCLUDE HEADER LIBRARY

Callback functions are defined in the header/s:

- Agnos.h

12.4 SAMPLE CODE

```
int myCBAppSelection(
tADFList *mutualList,
BYTE *AIDIndex,
BYTE *MLIndex,
const tTerminalContext *terminal,
BOOL CLMode)
{
 //Put your own App Selection routines

iRetVal = payTRANSACTION_TERMINATED,
iRetVal = payTRANSACTION_CANCELLED
iRetVal = payNO_ERROR
```

```
 //Set MLIndex
 *MLIndex = selectedIndx;

return iRetVal;
}

int myCBLangSelection(
unsigned char LangId,
unsigned char StringId,
int TimeOut,
unsigned char *SelectedId);
{
 //Put your own Language Selection routine

iRetVal = payNO_ERROR

 //Set SelectedId
 *SelectedId = selectedIndx;

return iRetVal;
}

unsigned short  myCBSelectItemFromList (
const char *arrItems[],
const unsigned char arrCount,
const char *Title,
const int TimeOut,
unsigned char *SelectedId)
{
int iIdx = 0;

//Put your own list item display screen
for(iIdx=0; iIdx < arrCount; iIdx__)
{
 display_item("%s", arrItems[iIdx]);
}

//set SelectedId
*SelectedId = idx;

return lstNO_ERROR;

}
```

13 API REFERENCE – CONFIGURATION PROCESSING

13.1 DATA TYPES AND STRUCTURES

13.1.1 CONFIGURATION

```
typedef enum
{
      TERMINAL_CFG,
      PROCESSING_CFG,
      ENTRY_POINT_CFG,
      CAKEYS_CFG,
      CRL_CFG
}tConfigType;
```

13.1.2 PROCESSING TYPE

```
typedef enum
{
NO_PROC_TYPE,
      CT_PROC_TYPE,
      CTLS_PROC_TYPE
}tProcessingType;
```

13.1.3 CONFIGURATION TRANSACTION TYPE

```
typedef enum
{
      CFG_TRANS_PURCHASE,
      CFG_TRANS_CASH,
      CFG_TRANS_WITHCASHBACK,
      CFG_TRANS_REFUND
}tConfigTransType;
```

13.1.4 CAP KEYS STRUCTURE

```
typedef struct _tCAPK_
{
      BYTE RID[RID_SIZE];
      BYTE Index;
      BYTE ExpoLen;          //0x01 or 0x03 as per EMV specs
      BYTE ModLen;           //Max 248 as per EMV specs
      BYTE Exponent[MAX_EXPONENT_SIZE];
      BYTE Modulus[MAX_MODULUS_SIZE];
      BYTE Expiry[3];        //ddmmyy
      BYTE Checksum[CHECKSUM_SIZE];
```

```
        struct _tCAPK_ *pnext;
}tCAPK;
```

13.1.5 CRL STRUCTURE

```
typedef struct _tCRL_
{
        BYTE RID[RID_SIZE];
        BYTE Index;
        BYTE SerialNumber[3];

        struct _tCRL_ *pnext;
}tCRL;
```

13.2 CONFIGURATION API FUNCTIONS

13.2.1 READS CONFIGURATION FILE AND LOADS TO MEMORY

Prototype	`int LoadAgnosConfig ( tConfigType CfgType);`
Function	This function reads the specified configuration file type as follows: • TERMINAL_CFG → ./AGNOS/TERMINAL • PROCESSING_CFG → ./AGNOS/PROCESSING • ENTRY_POINT_CFG → ./AGNOS/ENTRY_POINT • CAKEYS_CFG → ./CAKeys.xml • CRL_CFG → ./AGNOS/CRL
Input	tConfigType (see tConfigType enum values)
Output	NA
Return	AMP_SUCCESS, AMP_ERROR
Notes	Memory must be cleared after processing the configuration by calling the UpdateAgnosConfig() or ClearMemoryAgnosConfig().

13.2.2 WRITES OR UPDATES CONFIGURATION FILE FROM THE DATA MEMORY

Prototype	`int UpdateAgnosConfig ( tConfigType CfgType);`
Function	This function updates or saves the data loaded in memory to the specified configuration file.

	It also performs memory cleanup routine.
Input	tConfigType (see tConfigType enum values)
Output	NA
Return	AMP_SUCCESS, AMP_ERROR
Notes	NA

13.2.3 SET FIELD VALUE OF THE SPECIFIC RECORD

Prototype	`int SetFieldTagValueAgnosConfig (`
	`  tConfigType CfgType,`
	`  unsigned char KernelID,`
	`  BYTE *AIDKey,`
	`  unsigned char uAIDKeyLen,`
	`  tProcessingType ProcType,`
	`  tConfigTransType TransType,`
	`  unsigned long ulTag,`
	`  BYTE *Value,`
	`  unsigned long ulLen);`
	
	`int SetFieldTagValueAgnosConfig (`
	`  tConfigType CfgType,`
	`  unsigned char KernelID,`
	`  BYTE *AIDKey,`
	`  unsigned char uAIDKeyLen,`
	`  tProcessingType ProcType,`
	`  tConfigTransType TransType,`
	`  BYTE *TlvParams,`
	`  unsigned long ulTlvParamsLen);`

Function	It sets the value of the configurable parameter, refer to the **TERMINAL**, **PROCESSING** and **ENTRY_POINT** sections for the equivalent tags of its parameters.
Input	tConfigType (see tConfigType enum values) KerneID – Kernel identifier, refer to **ENTRY_POINT** configuration section AIDKey – Used for PROCESSING and ENTRY_POINT uAIDKeyLen – Length of the input AID ProcType –Indicates if the AID is for contact or contactless for the PROCESSING configuration (see tProcessingType enum values)

	TransType – For the ENTRY_POINT configuration (refer to tConfigTransType enum values) ulTag – The equivalent tag of the parameter that needs to set the value (refer to the **TERMINAL**, **PROCESSING** and **ENTRY_POINT** sections) Value – The desired value of the parameter ulLen – The length of the input Value TlvParams – TLV stream of the parameters set ulTlvParamsLen – The length of TlvParams
Output	NA
Return	AMP_SUCCESS, AMP_ERROR

Notes	Sample:
	
	`int DoAgnosTerminalConfig()`
	`{`
	`int iRetVal = AMP_SUCCESS;`
	`cAMPAgnosConfig *pAgnosConfig =`
	`cAMPAgnosConfig::get_Instance();`
	`BYTE byValue[MAX_PARAM_DATA_SIZE]={0};`
	`unsigned long byValueLen=0;`
	`int iRec=0;`
	`//Set of parameters`
	`char szTlvParams[] =`
	`"9f1a020056" //country code`
	`"df7305cc00000000" //tac default`
	`"df74050000000000" //tac denial`
	`"df7505cc00000000" //tac online`
	`;`
	
	`BYTE byTlvParams[MAX_PARAM_DATA_SIZE]={0};`
	`unsigned long ulTlvParamsLen=0;`
	
	`//add TERMINAL record`
	`pAgnosConfig->AddRecordToAgnosConfig(TERMINAL_CFG, 0, NULL, 0, NO_PROC_TYPE, (tConfigTransType) 0 );`
	
	`//convert string szTlvParams to bytes`
	`ulTlvParamsLen= pAgnosConfig->Asc2Hex((unsigned char*)szTlvParams, byTlvParams);`
	
	`//configure set of tag value`

	pAgnosConfig->SetFieldTagValueAgnosConfig(
	TERMINAL_CFG, //config type
	0, //kernel id
	NULL, //AIDKey
	0, //AIDKeyLen
	NO_PROC_TYPE, //ProcessingType
	(tConfigTransType) 0, //ConfigTransType
	byTlvParams, //TlV params
	ulTlvParamsLen //Tlv params len
	);
	
	//Get Record for Debugging
	pAgnosConfig->GetRecordAgnosConfig(TERMINAL_CFG, 1, byValue, &byValueLen);
	#ifdef DEBUG_AGNOS_CONFIG
	_LogHEXBuffer((unsigned char*)byValue, byValueLen, "TERMINAL TLV Data");
	#endif
	
	//Save to TERMINAL configuration file
	pAgnosConfig->UpdateAgnosConfig(TERMINAL_CFG);
	
	return iRetVal;
	}

13.2.4 GET FIELD VALUE OF THE SPECIFIC RECORD

Prototype	int GetFieldTagValueAgnosConfig (tConfigType CfgType, unsigned char KernelID, BYTE *AIDKey, unsigned char uAIDKeyLen, tProcessingType ProcType, tConfigTransType TransType, unsigned long ulTag, BYTE *Value, unsigned long *ulLen);
Function	It retrieves the value of the configurable parameter, refer to **TERMINAL, PROCESSING** and **ENTRY_POINT** sections for the equivalent parameters' tags.
Input	tConfigType (see tConfigType enum values) KernelID – Kernel identifier, refer to the **ENTRY_POINT** configuration section

	AIDKey – Used for PROCESSING and ENTRY_POINT uAIDKeyLen – Length of the input AID ProcType – Indicates if the AID is for contact or contactless transactions for the PROCESSING configuration (see tProcessingType enum values) TransType – For ENTRY_POINT configuration (refer to tConfigTransType enum values) ulTag – The equivalent tag of the parameter that needs to set the value (refer to the **TERMINAL**, **PROCESSING** and **ENTRY_POINT** sections)
Output	Value – Returns the parameter's value ulLen – The length of the output Value
Return	AMP_SUCCESS, AMP_ERROR
Notes	NA

13.2.5 GET RECORD COUNT OF CONFIGURATION

Prototype	`int GetRecordCountAgnosConfig(` `  tConfigType CfgType );`
Function	It returns the record count of the specified configuration type that can be used for record iteration.
Input	tConfigType (see tConfigType enum values)
Output	NA
Return	record count

Notes	`//! Get Record Count`
	`iRecCount = pConfig->GetRecordCountAgnosConfig(`
	`  PROCESSING_CFG);`
	
	`//Traverse records`
	`for(iIndex=1; iIndex<=iRecCount; iIndex++)`
	`{`
	`  pConfig->GetRecordAgnosConfig(`
	`  PROCESSING_CFG, iIndex, Data, &DataLen);`
	`}`
	
	The TERMINAL has only one (1) record that contains the parameters

	set.

13.2.6 GET RECORD VALUE BY INDEX OR RECORD NUMBER

Prototype	`int GetRecordAgnosConfig(`
	`  tConfigType CfgType,`
	`  const int RecNumber,`
	`  BYTE *Data,`
	`  unsigned long *DataLen );`

Function	This function returns the data record from configuration (TERMINAL, PROCESSING, ENTRY_POINT) files by specifying the index number.
Input	tConfigType (see tConfigType enum values) RecNumber – Records the index number
Output	Data – Returns the data record DataLen – Length of the data record
Return	AMP_SUCCESS, AMP_ERROR

Notes	TERMINAL record:
	
	`Data (113):`
	`0000 9F 1A 02 00 56 DF 79 01 01 9F 35 01 22 DF 0A 01`
	`- ....V.y...5."...`
	`0010 01 9F 33 03 E0 F0 C8 9F 40 05 F0 00 F0 A0 01 DF`
	`- ..3.....@.......`
	`0020 55 01 01 DF 0B 01 00 DF 27 01 0F DF 06 01 00 DF`
	`- U.......'.......`
	`0030 08 01 00 DF 7A 01 01 DF 0D 01 00 DF 10 03 00 00`
	`- ....z...........`
	`0040 00 DF 7B 01 01 DF 07 01 00 DF 09 01 01 DF 73 05`
	`- ..{...........s.`
	`0050 DC 50 84 00 00 DF 74 05 00 10 00 00 00 DF 75 05`
	`- .P....t.......u.`
	`0060 C4 00 00 00 00 DF 53 01 00 DF 54 01 00 DF 7C 01`
	`- ......S...T...|.`
	`0070 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00`
	`- ................`
	
	PROCESSING record:
	
	`Data (115):`
	`0000 9F 01 06 F2 00 00 00 00 00 4F 07 A0 00 00 00 04`
	`- .........O......`
	`0010 10 10 DF 7E 01 01 9F 09 02 00 00 DF 11 01 00 DF`
	`- ...~............`

```
0020 12 01 00 DF 13 01 00 DF 14 01 00 DF 15 01 00 DF
- ................
0030 20 05 00 00 00 00 00 DF 21 05 00 00 00 00 00 DF
-  .......!.......
0040 22 05 00 00 00 00 00 9F 1B 04 00 00 00 00 DF 70
- ".............p
0050 01 00 DF 6E 03 00 00 00 DF 6F 01 00 DF 01 01 00
- ...n.....o......
0060 DF 71 00 DF 02 01 00 DF 72 00 5F 2A 02 00 00 5F
- .q......r._*..._
0070 36 01 00 00 00 00 00 00 00 00 00 00 00 00 00 00
- 6...............
```

ENTRY_POINT record:

```
Data (165):
0000 DF 0E 03 02 01 00 DF 0F 81 9B 5F 57 00 9F 01 00
- .........._W....
0010 9F 40 05 00 00 00 00 00 9F 09 02 00 02 DF 17 01
- .@..............
0020 00 DF 18 01 60 DF 19 01 08 DF 1A 03 9F 6A 04 DF
- ....`........j..
0030 1B 01 20 DF 0C 01 02 DF 1E 01 10 DF 2C 01 00 DF
- .. .........,...
0040 1C 02 00 00 DF 1D 01 00 9F 15 02 01 30 9F 16 00
- ............0...
0050 9F 4E 00 9F 7E 00 DF 23 06 00 00 00 01 00 00 DF
- .N..~..#........
0060 24 06 00 00 00 03 00 00 DF 25 06 00 00 00 05 00
- $..............
0070 00 DF 26 06 00 00 00 00 10 00 DF 03 01 08 DF 20
- ..&............
0080 05 00 00 00 00 00 DF 21 05 00 00 00 00 00 DF 22
- .......!........"
0090 05 00 00 00 00 00 9F 33 00 9F 1A 02 00 56 9F 1C
- .......3.....V..
00A0 00 9F 35 01 22 00 00 00 00 00 00 00 00 00 00 00
- ..5."...........
```

CAKeys record:

```
Data (137):
0000 A0 00 00 00 04 FE 01 80 03 A6 53 EA C1 C0 F7 86
- .........S.....
0010 C8 72 4F 73 7F 17 29 97 D6 3D 1C 32 51 C4 44 02
- .rOs.)..=.2Q.D.
0020 04 9B 86 5B AE 87 7D 0F 39 8C BF BE 8A 60 35 E2
- ...[..}.9....`5.
0030 4A FA 08 6B EF DE 93 51 E5 4B 95 70 8E E6 72 F0
- J..k...Q.K.p..r.
```

	0040 96 8B CD 50 DC E4 0F 78 33 22 B2 AB A0 4E F1 37 - ...P...x3"...N.7 0050 EF 18 AB F0 3C 7D BC 58 13 AE AE F3 AA 77 97 BA - ....<}.X.....w.. 0060 15 DF 7D 5B A1 CB AF 7F D5 20 B5 A4 82 D8 D3 FE - ..}[.... ..... 0070 E1 05 07 78 71 11 3E 23 A4 9A F3 92 65 54 A7 0F - ...xq.>#....eT.. 0080 E1 0E D7 28 CF 79 3B 62 A1 00 00 00 00 00 00 00 - ...(.y;b..... CRL record: Data (9): 0000 A0 00 00 00 25 01 01 10 10 00 00 00 00 00 00 00 - .....

13.2.7 ADD RECORD TO THE CONFIGURATION

Prototype	`int AddRecordToAgnosConfig(`
	`tConfigType CfgType,`
	`unsigned char KernelID,`
	`BYTE *AIDKey,`
	`unsigned char uAIDKeyLen,`
	`tProcessingType ProcType,`
	`tConfigTransType TransType,`
	`BYTE *TlvParams=NULL,`
	`unsigned long ulTlvParamsLen=0 );`

Function	This function adds a record to the record list of the configuration with the internal default values of its parameters.
Input	tConfigType (see tConfigType enum values) KerneID – Kernel identifier, refer to the **ENTRY_POINT** configuration section AIDKey – Used for PROCESSING and ENTRY_POINT uAIDKeyLen – Length of the input AID ProcType – Indicates if the AID is for contact or contactless transactions for the PROCESSING configuration (see tProcessingType enum values) TransType – For ENTRY_POINT configuration (refer to tConfigTransType enum values)

	TlvParams – TLV stream of the parameters set ulTlvParamsLen – Length of TlvParams
Output	NA
Return	AMP_SUCCESS, AMP_ERROR

Notes	When creating a new record, the parameter should be modified as shown in the sample:
	
	`//Add record`
	`AddRecordToAgnosConfig(…)`
	`//configure the parameter`
	`SetFieldTagValueAgnosConfig(...)`
	`SetFieldTagValueAgnosConfig(...)`
	`SetFieldTagValueAgnosConfig(...)`
	`The function is only applicable for TERMINAL, PROCESSING, and ENTRY_POINT`
	
	Sample:
	
	`int DoAgnosProcessingConfig()`
	`{`
	`int iRetVal = AMP_SUCCESS;`
	`cAMPAgnosConfig *pAgnosConfig =`
	`cAMPAgnosConfig::get_Instance();`
	
	`BYTE Data[MAX_PARAM_DATA_SIZE]={0};`
	`unsigned long DataLen=0;`
	`unsigned long byValueLen=0;`
	`int iRec=0;`
	`int iRecCount=0;`
	
	`typedef struct _myAID_`
	`{`
	`unsigned char szAID[16+1];`
	`BYTE KernelId;`
	`}myAID;`
	
	`BYTE szAIDCtlsDefault[] =`
	`"DF7E0101" //application selection indicator`
	`"9F09020001" //avn`
	`"DF110100" //skip TAC/IAC default supported`
	`"DF120100" //Random transaction selection`
	`"DF130100" //Velocity checking`
	`"DF140100" //Floor limit checking`

```
            "DF150100"       //TAC supported
            "DF20050000000000"     //TAC Default
            "DF21050000000000"     //TAC Denial
            "DF22050000000000"     //TAC Online
            "9F1B0400000000"      //Floor limit
            "DF700100"       //Target Percentage
            "DF6E03000000"         //Threshold value
            "DF6F0100"       //Max Target Percentage
            "DF010100"       //Default DDOL supported
            "DF7100"         //Default DDOL
            "DF020100"       //Default TDOL supported
            "DF7200"         //Default TDOL
            "5F2A020840"           //Transaction
Currency Code
            "5F360100";            //Transaction
Currency Exponent

      BYTE byTlvParams[MAX_PARAM_DATA_SIZE]={0};
      unsigned long ulTlvParamsLen=0;
            //convert string params to hex
      ulTlvParamsLen = pAgnosConfig->Asc2Hex(
      szAIDCtlsDefault, byTlvParams);

      myAID AIDKeys[]=
      {
            //contactless
            { "A0000000041010",          2 },
            { "A0000000043060",          2 },
            { "A0000000031010",          3 },
            { "A0000000999090",          3 },
            { "A000000025010403",        4 },

            //contact
            { "A0000000031010",          0 },
            { "A000000010",              0 },
            { "A0000000041010",          0 },
            { "A0000000046000",          0 },
            { "A0000000043060",          0 },
            { "A000000004101001",        0 },
            { "A000000004101002",        0 },
            { "A00000002501",            0 },
            { "A0000001523010",          0 },
      };
      int AidKeysLen =sizeof(AIDKeys)/sizeof(myAID);
      BYTE byAID[15]={0};
      int iAIDLen=0;
      tProcessingType ProcType;

      //3 Add PROCESSING records
```

```
      for(iRec=0; iRec < AidKeysLen; iRec++)
      {
            //convert string AID to hex
pAgnosConfig->Asc2Hex(AIDKeys[iRec].szAID, byAID) ;
            iAIDLen =
strlen((char*)AIDKeys[iRec].szAID)/2;
            if (AIDKeys[iRec].KernelId > 0 )
                  ProcType = CTLS_PROC_TYPE;
            else
                  ProcType = CT_PROC_TYPE;

pAgnosConfig->AddRecordToAgnosConfig(PROCESSING_CFG,
AIDKeys[iRec].KernelId, byAID, iAIDLen, ProcType,
(tConfigTransType)NULL, byTlvParams, ulTlvParamsLen);

            //Add record then change default value
for contactless
            if( ProcType == CTLS_PROC_TYPE)
            {
pAgnosConfig->AddRecordToAgnosConfig(PROCESSING_CFG,
AIDKeys[iRec].KernelId, byAID, iAIDLen, ProcType,
(tConfigTransType)NULL, byTlvParams, ulTlvParamsLen);

            }
            //Add record and use default parameters
else
{
pAgnosConfig->AddRecordToAgnosConfig(PROCESSING_CFG,
AIDKeys[iRec].KernelId, byAID, iAIDLen, ProcType,
(tConfigTransType)NULL);
            }

      }
```

13.2.8 DELETE RECORD IN THE CONFIGURATION

Prototype	`int DeleteRecordToAgnosConfig(`
	`  tConfigType CfgType,`
	`  unsigned char KernelID,`
	`  BYTE *AIDKey,`
	`  unsigned char uAIDKeyLen,`
	`  tProcessingType ProcType,`
	`  tConfigTransType TransType );`

Function	Deletes the record in the record list by specifying the record key. • PROCESSING – AIDKey, uAIDKeyLen, ProcType • ENTRY_POINT – KernelID, AIDKey, uAIDKeyLen, TransType

Input	tConfigType – (see tConfigType enum values) KerneID – Kernel identifier, refer to the **ENTRY_POINT** configuration section AIDKey – Used for PROCESSING and ENTRY_POINT uAIDKeyLen – Length of input AID ProcType – Indicates if the AID is for contact or contactless PROCESSING configuration (see tProcessingType enum values) TransType – For the ENTRY_POINT configuration (refer to tConfigTransType enum values)
Output	NA
Return	AMP_SUCCESS, AMP_ERROR

Notes	Only applicable for TERMINAL, PROCESSING and ENTRY_POINT
	Example:
	TERMINAL:
	`DeleteRecordToAgnosConfig(TERMINAL_CFG, 0, 0, 0, 0, 0);`
	PROCESSING:
	`DeleteRecordToAgnosConfig(PROCESSING_CFG, 0, AIDKey, AIDKeyLen, ProcType, 0);`
	ENTRY_POINT:
	`DeleteRecordToAgnosConfig(ENTRY_POINT_CFG, KernelID, AIDKey, AIDKeyLen, 0, TransType);`

13.2.9 CLEAR CONFIGURATION DATA IN MEMORY

Prototype	`int ClearMemoryAgnosConfig(` `  tConfigType CfgType);`
Function	Clears the configuration data loaded in the memory. This should be used when you load the configuration with no intention to update the configuration file, to free up the memory by destroying the storage structure.
Input	CfgType – (refer to tConfigType enum values)

Output	NA
Return	AMP_SUCCESS, AMP_ERROR
Notes	NA

13.3 CAP KEYS ADD/DELETE RECORD API FUNCTIONS

13.3.1 ADD CAPK RECORD TO CAKEYS LIST

Prototype	`int AddCapkToCAKeys(` `  tCAPK *Capk);`
Function	This function adds the CAPK record to the CAKeys list. It validates the input checksum and expiry first before adding to the list.
Input	Capk – the CAPK data to be added to the list (refer to tCAPK structure)
Output	NA
Return	AMP_SUCCESS, AMP_ERROR
Notes	NA

13.3.2 DELETE CAPK RECORD TO CAKEYS LIST

Prototype	`int DeleteCapkToCAKeys(`
	`  BYTE *RID,`
	`  BYTE Index);`

Function	It searches the RID and Index as input keys to delete the record in the CAKeys list.
Input	RID – 5 bytes data Index – RID Index
Output	NA
Return	AMP_SUCCESS, AMP_ERROR
Notes	NA

13.3.3 GET CAPK RECORD FROM CAKEYS LIST

Prototype	`tCAPK* GetRecordCAKeysXml( const int iRecNumber );`
Function	It returns the CAPK structure from the CAKeys list to retrieve the CAPK information.
Input	iRecNumber – The record index number
Output	NA
Return	CAPK structure (refer to the **CAP Keys structure** to see its members' information)
Notes	NA

13.3.4 WRITE CAKEYS.XML FROM CAKEYS IN MEMORY

Prototype	`int WriteCAKeysXML();`
Function	It overwrites the CAKeys.xml file based on the CAKeys list loaded in the memory.
Input	NA
Output	NA
Return	AMP_SUCCESS, AMP_ERROR
Notes	NA

13.4 CRL CERTIFICATE REVOCATION LIST- ADD/DELETE RECORD API FUNCTIONS

13.4.1 ADD CRL RECORD TO CRL LIST

Prototype	`int AddCrlToCRLConfig( tCRL *pCrl);`
Function	This function adds the CRL record to the CRL list.
Input	pCrl – the CRL data to be added to the certificate revocation list (refer to **CRL structure**)
Output	NA

Return	AMP_SUCCESS, AMP_ERROR
Notes	NA

13.4.2 DELETE CRL RECORD FROM CRL LIST

Prototype	`int DeleteCrlToCRLConfig(`
	`  BYTE *RID,`
	`  BYTE Index,`
	`  BYTE *SerialNo);`

Function	It searches the RID, Index and Serial as input keys to delete the record in the CRL list.
Input	RID – 5 bytes data Index – RID Index SerialNo – Serial No
Output	NA
Return	AMP_SUCCESS, AMP_ERROR
Notes	NA

13.4.3 GET CRL RECORD FROM CRL CERTIFICATE REVOCATION LIST

Prototype	`tCRL* GetRecordCRLConfig(` `  const int iRecNumber );`
Function	It returns the CRL structure from the CRL list to retrieve the CRL information.
Input	iRecNumber – The record index number
Output	NA
Return	CRL structure (refer to the **CRL structure** to see its members' information)
Notes	NA

13.5 OTHER API FUNCTIONS

13.5.1 READ DATA FROM CONFIGURATION FILE

Prototype	`int ReadTlvDataFromConfig(`
	`tConfigType CfgType,`
	`BYTE *TlvDataBuff,`
	`unsigned long *ulDataSize);`

Function	It reads data stream to the configuration file. This function is responsible for version and checksum validation in the header. It then retrieves the data from the configuration file.
Input	CfgType – See tConfigType enum values
Output	TlvDataBuff – The actual data stream to be read in the file ulDataSize – Data length
Return	AMP_SUCCESS, AMP_ERROR
Notes	NA

13.5.2 WRITE DATA TO CONFIGURATION FILE

Prototype	`int WriteTlvDataToConfig(`
	`tConfigType CfgType,`
	`BYTE *TlvDataBuff,`
	`unsigned long ulDataSize);`

Function	It writes the data stream to the configuration file. This function allows the application layer to take care of the actual data stream to be written to the configuration file as the interface takes care of managing the header file.
Input	CfgType – See tConfigType enum values TlvDataBuff – The actual data stream to be written to the file ulDataSize – Data length
Output	NA
Return	AMP_SUCCESS, AMP_ERROR
Notes	NA

13.5.3 GET TAG VALUE FROM TLV STREAM

Prototype	`int GetTlvTagValue(`
	`  const BYTE *TlvStream,`
	`  const unsigned long ulTlvStreamLen,`
	`  const unsigned long ulTag,`
	`  BYTE *Value,`
	`  unsigned long ulValueSize);`

Function	A utility function that gets the value of the specified tag from the input TVL stream data.
Input	TlvStream – The input TLV stream to be parsed ulTlvStreamLen – TLV stream length ulTag – The tag to be searched ulValueSize – The size of the value buffer
Output	Value – The value retrieved from the input TLV stream
Return	Value length
Notes	NA

14 CONFIGURATION FILES

14.1 TERMINAL

The TERMINAL configuration file contains the set of configurable terminal specific parameters that are needed to process the EMV contact transaction

14.1.1 STRUCTURE

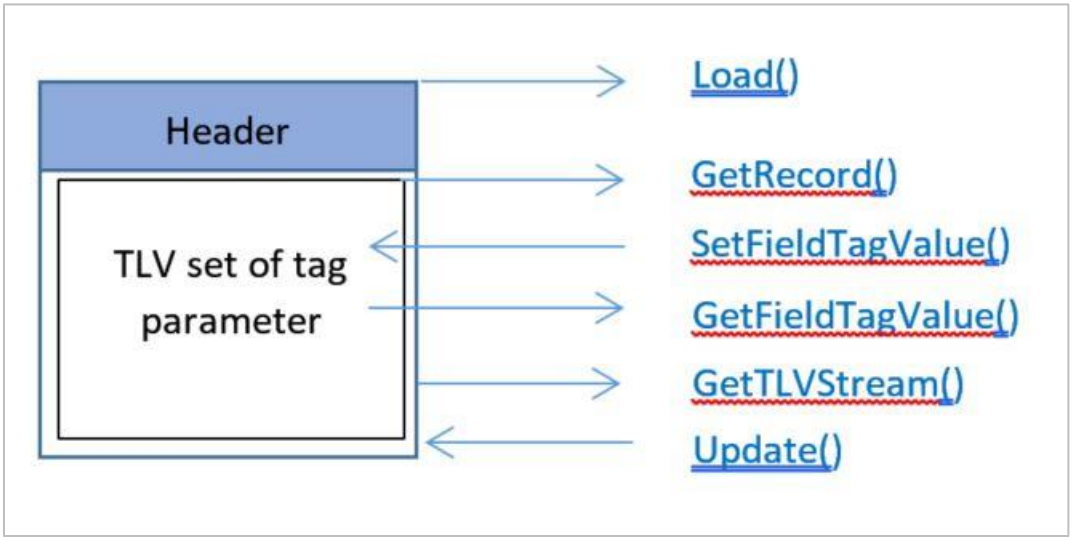

Figure 8 - Terminal Configuration File Structure.

Below is the terminal header table:

Tag	Length	Description
Version	unsigned char	Value between 0xAA and 0xAF
SHA-1	unsigned char[20]	SHA-1 calculation of the TERMINAL Tags

The following is the terminal tags table:

Tag	Length	Default	Description
9F1A	02	0826	Country Code
DF79	01	01	Cardholder Confirmation
9F35	01	22	Terminal Type
DF0A	01	01	EMV contact supported
9F33	03	E0F0C8	Terminal Capabilities
9F40	05	F000F0A001	Additional Terminal Capabilities
DF55	01	01	EMV contactless supported
DF0B	01	00	Magstripe supported

DF27	01	0F	Pin Timeout (sec)
DF06	01	00	Batch managed
DF08	01	00	Advice managed
DF7A	01	01	PSE Supported
DF0D	01	00	AutoRun mode (for contactless vending machine)
DF10	03	000000	Predefined amount for AutoRun mode
DF7B	01	01	Pin bypass supported
DF07	01	00	Referral managed
DF09	01	01	Default TAC supported
DF73	05	DC50840000	Default TAC-Default
DF74	05	0010000000	Default TAC-Denial
DF75	05	C400000000	Default TAC-Online
DF53	01	00	Random Transaction Selection not supported
DF54	01	00	Velocity checking not supported
DF7C	01	00	CDA type (mode 2 or 3 only)

14.1.2 API USAGE

- Read the TERMINAL configuration file for further processing
- Use Get() and Set() Application Program Interfaces (API) to configure the desired parameter
- Update the TERMINAL configuration file

14.1.3 MISCELLANEOUS USAGE

- Add record
- Delete record
- Get record TLV set of tag parameters

14.1.4 CODE SAMPLE

1) Sample 1

```
//Load TERMINAL file in memory for configuration processing
LoadAgnosConfig(TERMINAL_CFG);
```

```
//Set the value of the specified tag parameter
ulTag = 0x9f1a; //Terminal Country Code
sprintf(szValue, "0826");
AsciiHexToBinHex ( szValue, byteValue, ulLen);
SetFieldTagValueAgnosConfig(TERMINAL_CFG, NULL, NULL, NULL, NULL,
NULL, ulTag, byteValue, ulLen);

//Get the value of the desired tag parameter
ulTag = 0x9f1a; //Terminal Country Code
GetFieldTagValueAgnosConfig(TERMINAL_CFG, NULL, NULL, NULL, NULL,
NULL, ulTag, byteValue, &ulLen);
BinHexToAsciiHex( byteValue, szValue, ulLen);

//Writes the data loaded in memory to the TERMINAL file
UpdateAgnosConfig(TERMINAL_CFG));
```

2) Sample 2

```
//Add TERMINAL record
AddRecordToAgnosConfig(TERMINAL_CFG, NULL, NULL, 0, 0, 0);

//Delete TERMINAL record
DeleteRecordToAgnosConfig(TERMINAL_CFG, NULL, NULL, 0, 0, 0);

//Get TLV parameter set of TERMINAL record
GetTLVStreamAgnosConfig(TERMINAL_CFG, byteBuffer, ulBufferSize,
&ulLen);
```

14.2 PROCESSING

The PROCESSING configuration file holds the list of AIDs and its configurable parameters that the system supports for EMV contact and contactless cards.

Note: The sequence order of AID records should include AID-Contactless first then AID-Contact.

14.2.1 STRUCTURE

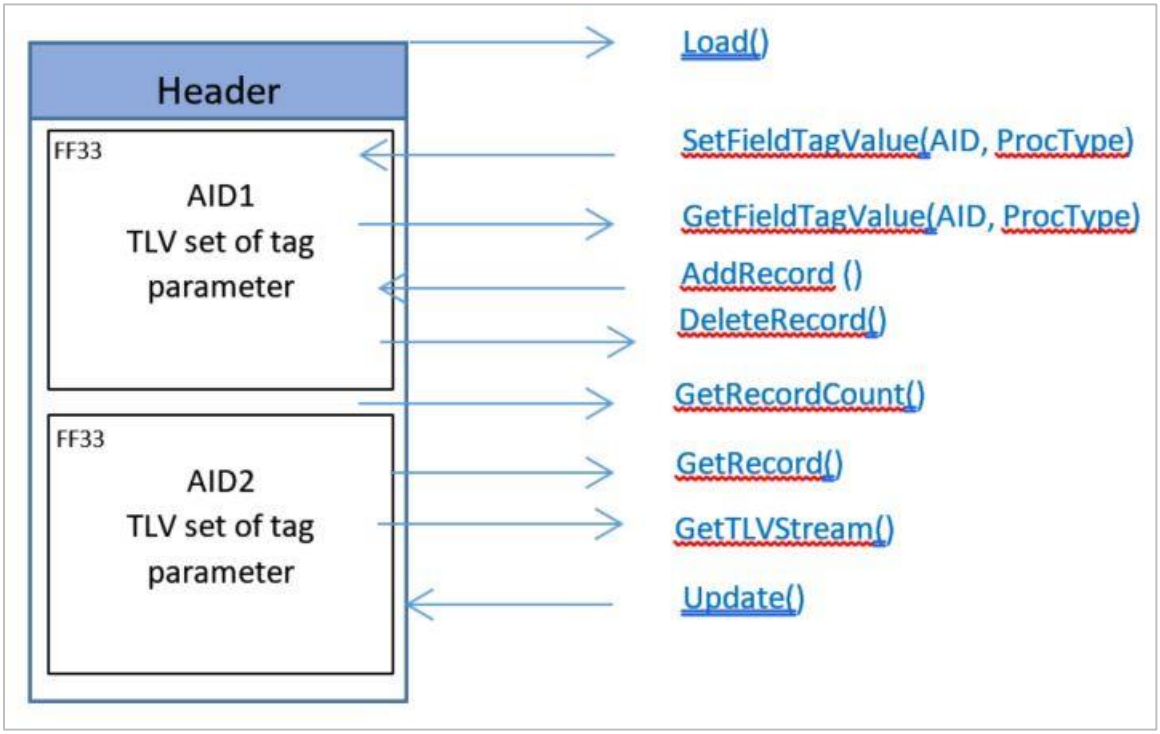

Figure 9 - Processing Configuration File Structure.

Below is the processing header table:

Tag	Length	Description
Version	unsigned char	Value between 0xAA and 0xAF
SHA-1	unsigned char[20]	SHA-1 calculation of the PROCESSING Tags

The following is the processing tags table:

Tag	Length	Default	Description
FF33	Variable		AID record delimiter, FF33 is a constructed data object that contains the set of AID parameters (9F01-5F36)
9F01		F20000000000 – PayPass F30000000000 –	Acquirer Identifier 0xF0|KernelID – For contactless AID

	06	PayWave F40000000000 – ExpressPay F50000000000 – JSpeedy F60000000000 – D-Pas F70000000000 – CUP F80000000000 – Flash 000000000000 – EMV Contact	
4F	05-16	A0000000041010	Application Identification (AID)
DF7E	01	01	Application Selection Indicator (ASI)
9F09	02	0000 – contactless '0001 – contact	Application Version Number (AVN)
DF11	01	00 – contactless '00 – contact	Skip TAC/IAC default supported (contact only) – A flag indicator on whether to skip the TAC/IAC-Default terminal process for UNABLE_TO_GO_ONLINE scenario
DF12	01	00 – contactless '01 – contact	Random transaction selection supported (contact only)
DF13	01	00 – contactless '01 – contact	Velocity Checking (contact only)
DF14	01	00 – contactless '01 – contact	Floor limit checking supported (contact only)
DF15	01	00 – contactless '01 – contact	TAC supported (contact only)
DF20	05	0000000000 – contactless 'C800000000 –	TAC-Default (contact only)

		contact	
DF21	05	0000000000	TAC-Denial (contact only)
DF22	05	0000000000 – contactless 'C800000000 – contact	TAC-Online (contact only)
9F1B	04	00000000 – contactless '000003E8 – contact (1000 in decimal)	Floor Limit (contact only)
DF70	01	00 – contactless '63 – contact (99% in decimal)	Target Percentage (contact only)
DF6E	03	000000 – contactless '0001F4 – contact (500 in decimal)	Threshold value (contact only)
DF6F	01	00 '63 – contact (99% in decimal)	Maximum Target Percentage (contact only)
DF01	01	00 – contactless '01 – contact	Default DDOL supported (contact only)
DF71	00 - 252	9F3704 - contact	Default DDOL (contact only)
DF02	01	00 – contactless '01 – contact	Default TDOL supported (contact only)
DF72	00 - 252	9F0802 – contact	Default TDOL (contact only)
5F2A	02	0000 – contactless '0840 – contact	Transaction Currency Code (contact only)
5F36	01	00 – contactless '02 – contact	Transaction Currency Exponent (contact only)

14.2.2 API USAGE

1) Read the PROCESSING configuration file for further processing
2) Get the AID record for Tag parameter settings
3) Use Get() and Set() Application Programmable Interfaces (API) to configure the desired parameter
4) Update the PROCESSING configuration file

14.2.3 MISCELLANEOUS USAGE

1) Add AID record
2) Delete AID record
3) Get record TLV set of tag parameters

14.2.4 CODE SAMPLE

1) Sample 1

```
//Load PROCESSING file in memory for configuration processing
LoadAgnosConfig(PROCESSING_CFG);

//Set the value of the specified tag parameter
ulTag = 0x5f2a; //Transaction Currency Code
sprintf(szAID, "A0000000041010");
AsciiHexToBinHex(szAID, byteAID, AidLen);
sprintf(szValue, "0840");
AsciiHexToBinHex ( szValue, byteValue, ulLen);
SetFieldTagValueAgnosConfig(PROCESSING_CFG, NULL, byteAID,   AidLen,
CT_PROC_TYPE, NULL, ulTag, byteValue, ulLen);

//Get the value of the desired tag parameter
ulTag = 0x5f2a; //Transaction Currency Code
GetFieldTagValueAgnosConfig(PROCESSING_CFG, NULL, byteAID,   AidLen,
NULL, CT_PROC_TYPE, ulTag, byteValue, &ulLen);
BinHexToAsciiHex( byteValue, szValue, ulLen);

//Writes the data loaded in memory to the PROCESSING file
UpdateAgnosConfig(PROCESSING_CFG));
```

2) Sample 2

```
//Add PROCESSING record with processing type of Contact
sprintf(szAID, "A0000000000010");
AsciiHexToBinHex(szAID, byteAID, AidLen);
```

```
AddRecordToAgnosConfig(PROCESSING_CFG,    NULL,    byteAID,    AIDLen,
CT_PROC_TYPE, 0);

//Delete PROCESSING record
DeleteRecordToAgnosConfig(PROCESSING_CFG, NULL, NULL, 0, 0, 0);

//Get record count
RecCount = GetRecordCountAgnosConfig(PROCESSING_CFG);

//Get TLV set of parameter of PROCESSING record
RecNo = 2;
GetRecordAgnosConfig(PROCESSING_CFG, RecNo, byteBuffer, &ulLen);
```

14.3 ENTRY_POINT

The ENTRY_POINT configuration file contains the list of COMBINATION of KernelID-AIDIndex-TransactionType (KAT) that will be used for EMV Contactless card processing,

14.3.1 STRUCTURE

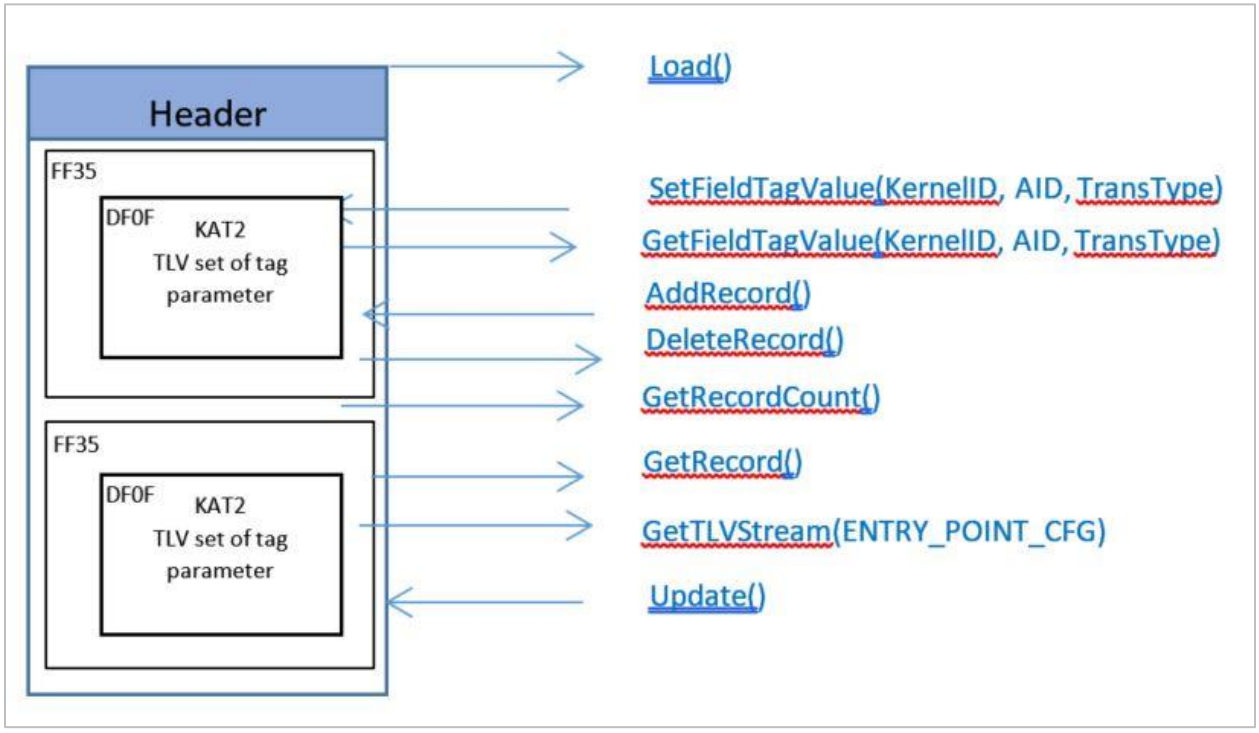

Figure 10 - Entry Point Configuration File Structure.

Below is the ENTRY_POINT Header table:

Tag	Length	Description
Version	unsigned char	Value between 0xAA and 0xAF
SHA-1	unsigned char[20]	SHA-1 calculation of the TERMINAL Tags

The following is the ENTRY_POINT Tags table:

Tag	Length	Default	Description
FF35	variable		KAT record delimiter, FF35 is a constructed data object that contains a set of KAT parameters (DF0E-DF0F)
DF0E	3	020100 – Paypass-AIDIndex1-Purchase	KAT Combination: Byte 1 – Kerned Identifier - 0x01: VISA/JCB - 0x02: PayPass Mastercard - 0x03: payWave VISA - 0x04: ExpressPay Amex - 0x05: J/Speedy JCB - 0x06: D-PAS Discover - 0x07: QuickPass CUP - 0x08: Flash Interac Byte 2 – the contactless AID Index listed in the PROCESSING file Byte 3 – Transaction Identifier - 0x00 for Purchase - 0x01 for Cash - 0x02 for Purchase with Cashback - 0x03 for Refund
DF0F	variable	variable	Is a TLV stream that contains a set of configurable parameters depending on the Kernel

1) Kernel-1 Pure Entry Point

N/A

2) Kernel-2 PayPass Mastercard

The table below shows the default set of parameters for Kernel-2:

Tag	Length	Default	Description
FF35	variable		KAT record delimiter, FF35 is a constructed data object that contains a set of KAT parameters (DF0E-DF0F)
DF0E	3	020100 – Paypass-AIDIndex1-Purchase	Paypass Configuration
DF0F	variable	variable	TLV stream that contains the set of configurable parameters below
DF1A	0 - 250	9F6A04	Default UDOL
DF0C	01	02	Kernel ID
9F6D	02	0001	Magstripe AVN
DF1E	01	F0	Magstripe CVM capabilities – CVM required b8-b5: 0000: No CVM 0001: Obtain Signature 0010: Online PIN 1111: N/A Other: RFU b4-b1: RFU
DF2C	01	F0	Magstripe CVM capabilities – no CVM required

			b8-b5: 0000: No CVM 0001: Obtain Signature 0010: Online PIN 1111: N/A Other: RFU b4-b1: RFU
DF23	06	000000010000	RCFL reader contactless floor limit
DF24	06	000000030000	RCTL reader contactless transaction limit
DF25	06	000000050000	RCTL On-Device CVM reader contactless transaction limit
DF26	06	000000001000	RCRL reader CVM required limit
9F1A	02	0056	Terminal country code
9F7E	00		Mobile support indicator
9F40	05	0000000000	Additional Terminal Capabilities
9F09	02	0002	AVN
DF17	01	00	Card data input capabilities b8: Manual Key Entry b7: Magnetic stripe b6: ICC with contacts b5-b1: RFU
DF18	01	60	CVM capabilities – CVM required b8: Plaintext PIN for ICC verification b7: Enciphered PIN for online verification b6: Signature (paper) b5: Enciphered PIN for offline verification

			b4: No CVM Required b3-b1: RFU
DF19	01	08	CVM capabilities - No CVM required b8: Plaintext PIN for ICC verification b7: Enciphered PIN for online verification b6: Signature (paper) b5: Enciphered PIN for offline verification b4: No CVM Required b3-b1: RFU
DF1B	01	20	Kernel Configuration b8: only EMV supported b7: only Magstripe supported b6: On device cardholder verification supported b5: Relay resistance protocol supported b4-b1: RFU
DF1C	02	012C – (300 in decimal)	Max Lifetime transaction (in secs)
DF1D	01	00	Max number of torn transaction
DF03	01	00	Security capabilities. b8: SDA b7: DDA b6: Card capture b5: RFU b4: CDA b3-b1: RFU Note: *PayPass supports CDA only*

DF20	05	0000000000	TAC-Default
DF21	05	0000000000	TAC-Denial
DF22	05	0000000000	TAC-Online
9F35	01	22	Terminal Type
DF2D	03	00000D (13 in decimal)	Message hold time in 100ms
9F15	02	0000	Merchant Category Code

Note: In the contact of PayPass (refer to EMV Contactless Specification for Payment Systems Book C-2), all DFxx tags must be understood as DF81xx except for DF03 which corresponds to DF811F.

MasterCard PayPass Settings/Conditions:

Tag	**Comments**	**Source**
DF04	Balance before GenAC • If present, offline is fetched from the card	ENTRY_POINT
DF05	Balance after GenAC • If present, offline balance is fetched from the card	ENTRY_POINT
DF20	TAC-Default • Similar to EMV standard • If not AAC and terminal offline only then TAC-Default is used • If 9F0D (IAC-Default) exists • If((DF20 OR 9F0D) AND TVR-95) NOT 0000000000: AAC • If((DF20 OR 9F0D) AND TVR-95) = 0000000000: TC • Else • If TVR-95 NOT 0000000000:AAC • If TVR-95 = 0000000000:TC	ENTRY_POINT
DF21	TAC – Denial • Similar to EMV standard • If 9F0E (IAC-Denial) does not exist, then 9F0E is set to 0000000000 • If (DF21 OR 9F0E) AND TVR-95) NOT 0000000000: AAC	ENTRY_POINT

DF22	TAC-Online • Similar to EMV standard • If not AAC and terminal not offline only, then TAC-Online is used • If 9F0F (IAC-Online) exists • If((DF22 OR 9F0F) AND TVR-95) NOT 0000000000: ARQC • If((DF22 OR 9F0F) AND TVR-95) = 0000000000: TC • Else • If TVR-95 NOT 0000000000:ARQC • If TVR-95 = 0000000000:TC	ENTRY_POINT
DF23	RCFL - Reader Contactless Floor Limit • If transaction amount > DF23 • TVR B4b8 set to 1 (Trans exceed Limit)	ENTRY_POINT
DF24	RCTL – (No On-Device CVM) • If AIP B1b2=0 OR DF1B B1b6=0; (AIP-On-Device cardholder verification supported, Kernel Config=0) • Transaction's RCTL, set to DF24 • If Transaction amount > Transaction's RCTL then • Next application selected (max limit exceeded)	ENTRY_POINT
DF25	RCTL – (On-Device CVM) • If AIP B1b2=1 AND DF1B B1b6=1; (AIP-On-Device cardholder verification supported, Kernel Config=0) • Transaction's RCTL, set to DF24 • If Transaction amount > Transaction's RCTL then • Next application selected (max limit is exceeded)	ENTRY_POINT

DF26	RCRL – Reader CVM Required Limit
	• If On-device cardholder verification is supported (AIP-B1b2=1)
	  ○ If transaction amount > DF26
	  ▪ CVM set to CONFIRMATION CODE VERIFIED
	• Else
	Magstripe mode
	• If transaction amount <=DF26
	  ○ DF2C is used as for CVM (DF2C- Magstripe CVM Capabilities No-CVM Required)
	  ○ Receipt is printed, if signature is required
	• Else
	  ○ DF1E is used as for CVM (DF1E- Magstripe CVM
		ENTRY_POINT

	Capabilities CVM Required) o Receipt is always printed EMV mode • If transaction amount <=DF26 o 9F33 B2 value set from DF19 for CVM processing • Else o 9F33 B2 value from DF18 for CVM processing o Receipt is always printed Where: DF18 – CVM Capabilities CVM Required DF19 – CVM Capabilities No CVM Required 9F33 – Terminal capabilities (not configurable)	
DF55	EMV Contactless Supported • If DF55=0 then terminal's behavior is forced to Magstripe mode	TERMINAL

Application Interchange Profile (Tag-82, Template-77, Length-2, Format-b):

Byte	**Bit**	**Definition**
Byte 1	b8	RFU
	b7	SDA supported
	b6	DDA supported
	b5	Cardholder verification supported
	b4	Terminal risk management is to be performed
	b3	Issuer Authentication is supported
	b2	On device cardholder verification is supported
	b1	CDA supported
Byte 2	b8	EMV mode is supported
	b7-2	Each bit RFU
	b1	Relay resistance protocol is supported

Kernel Configuration (Tag-DF1B, Tag-DF811B, Length-1, Format-b):

Byte	Bit	Definition
Byte 1	b8	Mag-stripe mode for contactless transactions is not supported
	b7	EMV mode for contactless transactions is not supported
	b6	On device cardholder verification is supported
	b5	Relay resistance protocol is supported
	b4-b1	Each bit RFU

3) Kernel-3 PayWave VISA

The following table shows the default set of parameters for Kernel-3:

Tag	Length	Default	Description
9F1A	02	0056	Terminal country code
DF1B	02	F000	Kernel Configuration Byte1 = b8: CVN17 fallback to MSD Legacy b7: Enable MSD CVN17 b6: MSD Formatting Track2 Data b5: MSD Constructing Track1 Data b4-b2: RFU b1: UL2 Byte2 = b8: DRL set option b7: Not Reset TVR b6-b1: RFU

DF2D	03	00000A (10 in decimal)	Message hold time in 100ms
9F09	02	0001	AVN
9F33	03	000040	Terminal capabilities
9F40	05	6000000000	Additional Terminal Capabilities
DF30	01	78	Bitmap Entry point b8: Status Check Support Flag b7: Zero Amount Allowed Flag b6: Reader Contactless Transaction Limit b5: Reader Contactless Floor Limit b4: Reader CVM Required Limit b3-b1: RFU
DF32	01	01	Status zero amount allowed flag 0x01: option1 = online cryptogram request 0x02: option2 = not allowed
9F35	01	22	Terminal Type Byte1 = b8-b7: RFU b6-b5: 01: Financial Institution 10: Merchant 11: Cardholder b4: RFU b3-b1: 001: Attended – Online Only 001: Attended – Offline with Online Capabilities

9F66	04	20000000	Terminal Transaction Qualifier (TTQ) Byte1 = b8: Magstripe mode b7: RFU b6: EMV Mode b5: EMV Contact Chip b4: Offline Only b3: Online PIN b2: Signature b1: RFU Byte2 = b8: Online Cryptogram (not configurable) b7: CVM Required (not configurable) b6: Offline PIN (not configurable) b5-b1: RFU Byte3 = b8: Issuer Script b7: Mobile CVM b6-b1: RFU Byte4 = b8-b1: RFU
9F1B	04	00000000	Terminal floor limit
DF23	06	000000010000	RCFL reader contactless floor limit
DF24	06	000000030000	RCTL reader contactless transaction limit
DF26	06	000000001000	RCRL reader CVM required limit

VISA-PayWave Settings/Conditions:

Tag	Comments	Source

9F1B	Floor Limit
	• If DF23 is not present
	  ○ If transaction amount > Floor Limit
	  ▪ TTQ B2b8 set to 1 (Online Cryptogram)
		ENTRY_POINT

9F33	Terminal Capabilities • Similar to EMV standard • Use SDA only for TTIG terminal	ENTRY_POINT

DF23	RCFL - Reader Contactless Floor Limit
	• If DF23 present
	  ○ If transaction amount > DF23
	  ▪ TTQ B2b8 is set to 1 (Online Cryptogram)
		ENTRY_POINT

DF24	RCTL • If Transaction amount > DF24 ○ Contactless application is not allowed	ENTRY_POINT
DF26	RCRL – Reader CVM Required Limit • If transaction amount > DF26 ○ TTQ B2b7 is set to 1 (CVM Required)	ENTRY_POINT
DF55	EMV Contactless Supported • If DF55=0 then terminal's behavior is forced to Magstripe mode	ENTRY_POINT

4) Kernel-4 ExpressPay Amex

The table below shows the default set of parameters for Kernel-4:

Tag	Length	Default	Description
9F6D	01	C0	Contactless reader capabilities b8-b7: 00: Expresspay 1.0 01: Expresspay 2.0 and Expresspay >= 3.x

			Magstripe 10: Expresspay 2.0 EMV and Magstripe 11: Expresspay>= 3.x Magstripe and EMV b6-b5: Used by EMVCo b4: Not to be configured (run-time setting for Expresspay >=3.x) 0: CVM is not required 1: CVM is required b3-b1: used by EMVCo
9F1A	02	0840	Terminal country code
9F1E	00		TID IFD Serial number
DF1B	02	3C21	Kernel configuration Byte1 = seed for random number calculation Byte2 = b8: (XpressPay3.1 only else RFU) 1: DRL set option b7: (XpressPay3.1 only else RFU) 1: EMVCo entrypoint pre-processing (C4) b6: (XpressPay3.1 only else RFU) 1: Use TAC default if unable to go online b5: no EMV consistency check performed b4: (ExpressPay2.0.2 only else RFU) 1: patch ODA is applied b3: Consistency check is performed as per EMV b2: Delayed authorization b1: ExpressPay2.02 else RFU

			0: full EMV (second genAC supported) 1: EMV partial mode Byte3 = hold time for SEE PHONE cases (in seconds) Byte4 = no holding time on approval/decline Note 1: B2b1 is always set to partial mode for rapid payment and for consistency with other contactless payment brands (i.e. no 2nd GenAC performed). From ExpressPay 3.x, B2bit1 is forced to 1, regardless of the configuration value. Note 2: tag 9F6E is used by Agnos ExpressPay 3.x Implementation: in Agnos, this tag is not a configuration data because it is built from 9F6D, 9F33 and 9F35 tags to avoid inconsistencies and to maintain upward compatibility.
9F35	01	22	Terminal type b8-b7: RFU b6-b5: 01: Financial Institution 10: Merchant 11: Cardholder b4: RFU b3-b1: 001: Attended - Online Only 001: Attended - Offline with Online Capabilities 001: Attended - Offline Only 001: Unattended - Online Only 101: Unattended - Offline with Online

			Capabilities 110: Unattended - Offline Only
9F40	05	6000000000	Additional terminal capabilities
9F09	02	0001	AVN
9F33	03	000088	Terminal capabilities
DF20	05	DC50840000	TAC-Default
DF21	05	0000000000	TAC-Denial
DF22	05	C400000000	TAC-Online
DF2D	03	00000D (13 in decimal)	Message hold time in 100ms
DF30	01	38	Bitmap entry point b8: StatusCheckSupportFlag b7: ZeroAmountAllowedFlag b6: ReaderContactlessTransactionLimit b5: ReaderContactlessFloorLimit b4: ReaderCVMRequiredLimit b3..b1: RFU
DF32	01	01	The status zero amount allowed flag 0x01: option1 = online cryptogram request 0x02: option2 = not allowed
DF23	06	000000001500	RCFL reader contactless floor limit
DF24	06	000000000000	RCTL reader contactless transaction limit
DF26	06	000000001000	RCRL reader CVM required limit
DF27	01	0F (15 in decimal)	Timeout value

Amex-ExpressPay Settings/Conditions:

Tag	Comments	Source

DF20	TAC-Default
	• Similar to EMV standard
	• If not AAC and terminal is offline only then TAC-Default is used
	• If 9F0D (IAC-Default) exists
	  o If((DF20 OR 9F0D) AND TVR-95) NOT 0000000000: AAC
	  o If((DF20 OR 9F0D) AND TVR-95) = 0000000000: TC
	• Else
	  o If TVR-95 NOT 0000000000:AAC
	  o If TVR-95 = 0000000000:TC
		ENTRY_POINT

DF21	TAC – Denial • Similar to EMV standard • If 9F0E (IAC-Denial) does not exist, then 9F0E is set to 0000000000 • If (DF21 OR 9F0E) AND TVR-95) NOT 0000000000: AAC	ENTRY_POINT

DF22	TAC-Online
	• Similar to EMV standard
	• If not AAC and terminal is not offline only then TAC-Online is used
	• If 9F0F (IAC-Online) exists
	  o If((DF22 OR 9F0F) AND TVR-95) NOT 0000000000: ARQC
	  o If((DF22 OR 9F0F) AND TVR-95) = 0000000000: TC
	• Else
	• If TVR-95 NOT 0000000000:ARQC
	• If TVR-95 = 0000000000:TC
		ENTRY_POINT

DF23	RCFL - Reader Contactless Floor Limit
	• During pre-processing, if not EMVCo entry point
	  o If DF23 present and DF24 absent and RCFL flag (DF30) = 1
	  ▪ If the transaction amount > DF23 and terminal not online capable
	  • Transaction is not allowed
	• Else
	  o See PayWave Entry Point Settings
	• During processing
	  o If transaction amount > DF23
	  ▪ TVR B4b8 raised (Transaction exceed floor
		ENTRY_POINT

	limit)	

DF24	RCTL – (No On-Device CVM)
	• During pre-processing, if not EMVCo entry point
	  ○ if DF24 is present
	  ▪ If the transaction amount > DF24 and RCTL flag (DF30) = 1
	  • Transaction is not allowed
	• Else
	  ○ See PayWave Entry Point Settings
		ENTRY_POINT

DF26	RCRL – Reader CVM Required Limit
	• During pre-processing, if not EMVCo entry point
	  ○ if DF26 present and DF24 absent and DF23 absent and RCRL flag (DF30) = 1
	  ▪ If the transaction amount > DF26 and CVM processing is not supported
	  • Transaction is not allowed
	• Else
	  ○ See PayWave Entry Point Settings
	• During processing
	  ○ If the transaction amount > RCRL
	  ▪ If supported switch to another interface
	  ▪ Else, stop transaction
		ENTRY_POINT

DF27	Timeout Value (ms)	ENTRY_POINT
DF55	EMV Contactless Supported • If DF55=0 then the terminal's behavior is forced to Magstripe mode	ENTRY_POINT

5) Kernel-5 J/Speedy JCB

N/A

6) Kernel-6 D-Pas Discover

The following table shows the default set of parameters for Kernel-6:

Tag	**Length**	**Default**	**Description**
9F09	02	0001	AVN

9F1A	02	0840	Terminal country code
DF1B	01	00	Kernel configuration b8: Exception list supported b7: Check PID Limit b6: Entry point supports extended selection b5-b1: RFU
9F35	01	22	Terminal type
5F2A	02	0840	Transaction Currency Code
9F33	03	20F8C8	Terminal Capabilities
DF30	01	38	Bitmap entry point b8: StatusCheckSupportFlag b7: ZeroAmountCheckFlag b6: ReaderContactlessTransactionLimit b5: ReaderContactlessFloorLimit b4: ReaderCVMRequiredLimit b3..b1: RFU
DF32	01	01	Status zero amount allowed flag 0x01: option1 = online cryptogram request Other: not allowed
9F66	04	24000000	Terminal Transaction Qualifier (TTQ) Byte1 = b8: Magstripe mode b7: RFU b6: EMV Mode b5: EMV Contact Chip

			b4: Offline Only b3: Online PIN b2: Signature b1: RFU Byte2 = b8: Online Cryptogram (not configurable) b7: CVM Required (not configurable) b6: Offline PIN (not configurable) b5-b1: RFU Byte3 = b8: Issuer Script b7: Mobile CVM b6-b1: RFU Byte4 = b8-b1: RFU
9F1B	04	0000000A	Terminal floor limit
DF23	06	000000001000	RCFL reader contactless floor limit
DF24	06	000000001500	RCTL reader contactless transaction limit
DF26	06	000000001000	RCRL reader CVM required limit

Discover-D-PAS Settings/Conditions:

Tag	Comments	Source

9F1B	Floor Limit
	• If DF23 is not present
	  ○ If the transaction amount > Floor Limit
	  ▪ TTQ B2b8 set to 1 (Online Cryptogram)
		ENTRY_POINT

DF23	RCFL - Reader Contactless Floor Limit
	• If DF23 present
	  ○ If the transaction amount > DF23
	  ▪ TTQ B2b8 set to 1 (Online Cryptogram)
		ENTRY_POINT

DF24	RCTL • If the transaction amount >= DF24 ○ Contactless application not allowed	ENTRY_POINT
DF26	RCRL – Reader CVM Required Limit • If the transaction amount >= DF26 ○ TTQ B2b7 set to 1 (CVM Required)	ENTRY_POINT

7) Kernel-7 QuickPass CUP

N/A

8) Kernel-8 Interac-Flash

The table below shows the default set of parameters for Kernel-8:

Tag	**Length**	**Default**	**Description**
9F09	02	0001	AVN
9F1A	02	0840	Terminal country code
9F1B	04	0000000A	Terminal floor limit
DF1B	02	0294	Kernel configuration 1st Byte – Interac Retry Limit 2nd Byte: b8: Interac Contact b7: Interac on Other Terminal b6: Mobile NFC b5: CL Card b4: Legacy Floor Limit

			b3: Always Discretionary Data b2: v1.4 Flash Terminal b1: RFU
9F35	01	22	Terminal type
9F33	03	604808	Terminal Capabilities
9F40	05	0000003000	Additional Terminal Capabilities
9F58	01	03	Merchant Type Indicator (MTI) 01, 02, 03, 04, 05
9F52	0C	000000001000 000000002000	MTI Limit-1 1st 6-bytes – Contactless per transaction limit 2nd 6-bytes – Contactless CVM Required Limit
9F54	0C	000000001000 000000002000	MTI Limit-2
9F55	0C	000000001000 000000002000	MTI Limit-3
9F56	0C	000000001000 000000002000	MTI Limit-4
9F57	0C	000000001000 000000002000	MTI Limit-5
9F5A	01	00	Terminal Transaction Type (TTT) 00 – Sale 01 – Refund
9F5D	06	000000002000	Terminal Contactless Receipt Required Limit (RRL)
9F5E	02	0000	Terminal Option Status (TOS) 1st Byte: b8: Use Interface If Different

			Currency b7: Use Interface If Different Country Code b6: Use Interface If Domestic Transaction with Different Currency Code b5-b1: RFU 2nd Byte – RFU
9F5F	06	000000001000	Reader Contactless Floor Limit
DF20	05	0000000000	TAC-Default
DF21	05	0000000000	TAC-Denial
DF22	05	0000000000	TAC-Online
9F49	00-FF	00	DDOL
DF2D	03	00000D	Message Hold Time (100 of ms)
DF6E	03	000000	Threshold Value for Biased Random Selection (binary format)
DF6F	01	00	Max Target Percentage for Biased Random Selection (binary format)
DF70	01	00	Target Percentage for Biased Random Selection
9F59	03	Auto Generated	Terminal Transaction Information (TTI) - Set 9F33 Terminal Capabilities - Set 9F40 Additional Terminal Capabilities - Set DF1B Kernel Configuration 9F59:

			Byte-1: b8: Reader with display capability b7: Interac Contact application available b6: Interac Contact application at other Terminal b5: CDA Supported b4: Offline Capable terminal (0 means online only terminal) b3: Online PIN Supported b2-b1: RFU Byte-2: b8-b7: 00 – Contactless only Capable 01 – Contactless & Mag stripe-read capable 10 – Contactless, Contact Chip & Mag stripe-read capable 11 – Contactless & Contact Chip capable b6-b4: RFU b3: Mobile NFC Device (FFI=03) accepted b2: Contactless Card (FFI= 00, 01 or 02) accepted b1: Always 1. It indicates acceptance data is present in this version of TTI.

Interac-Flash Settings/Conditions:

Tag	Comments	Source
9F52 9F54 9F55 9F56 9F57	Contactless Per Transaction Limit as indicated in Tag-9F58 MTI • If CVR B2b3=0 Offline limit must be skipped? o If the transaction amount >= Contacltess Per Trans Limit ▪ Set CVR B4b1=1; Contactless Per Trans Limit Exceeded Contactless CVM Required Limit • If CVR B2b3=0 Offline limit must be skipped? o If the transaction amount >= Contactless CVM Required Limit ▪ Set CTI B4b1=1; CVM Required by Card ▪ Set CVR B4b2=1; Contactless CVM Required Limit Exceeded	ENTRY_POINT
9F5D	Terminal Contactless Receipt Required Limit • If Transaction amount >= Receipt Required Limit o The terminal must automatically print a transaction record	ENTRY_POINT
9F5F	Reader Contactless Floor Limit • If transaction amount > Reader Contactless Floor Limit o Set TVR B4b1; Transaction Exceeds Floor Limit	ENTRY_POINT

Limits:

Limit Used	Reference Comparison	Outcome
Contactless Per Transaction Limit	Transaction Amount	Use Other Interface
Contactless CVM Required Limit	Transaction Amount	Process CVM or not
Contactless Card Floor Limit	Transaction Amount	Online Transaction
Lower Total Contactless Offline Amount Limit	Card Permitted Contactless Spend Accumulator	Online Transaction
Upper Total Contactless Offline Amount Limit	Card Permitted Contactless Spend Accumulator	Online Transaction

14.3.2 API USAGE

- LoadAgnosConfig(ENTRY_POINT_CFG) – loads ENTRY_POINT config in memory
- GetRecordCountAgnosConfig(ENTRY_POINT_CFG) – get record count
- GetRecordAgnosConfig(ENTRY_POINT_CFG, iIndex, Data, &DataLen)
- UpdateAgnosConfig(ENTRY_POINT_CFG) – Updates ENTRY_POINT config file

14.3.3 CODE SAMPLE

```
//!Configure ENTRY_POINT of Agnos
int DoAgnosEntryPointConfig(void)
{
	int iRetVal = AMP_SUCCESS;
	cAMPAgnosConfig *pConfig = cAMPAgnosConfig::get_Instance();

	unsigned long ulTag=0x00;
	unsigned long ulLen=0;
	int iRecCount=0;
	BYTE Data[MAX_PARAM_DATA_SIZE]={0};
	int iIndex=0;
	unsigned long DataLen=0;

	//3  Load the ENTRY_POINT config file for processing
	//Reads ENTRY_POINT file (./AGNOS/ENTRY_POINT)
	pConfig->LoadAgnosConfig(ENTRY_POINT_CFG);

	//3 Add routines for Get/Set of Tags in ENTRY_POINT
	//! Get Record Count
	iRecCount = pConfig->GetRecordCountAgnosConfig(
		ENTRY_POINT_CFG);

	//Traverse records
	for(iIndex=1; iIndex<=iRecCount; iIndex++)
	{
		pConfig->GetRecordAgnosConfig(
			ENTRY_POINT_CFG, iIndex, Data, &DataLen);
	}

	//3 Writes the loaded data to ENTRY_POINT config file
	(./AGNOS/ENTRY_POINT), it also clears data in memory
		pConfig->UpdateAgnosConfig(ENTRY_POINT_CFG);

		return iRetVal;
	}
```

14.4 CAKEYS.XML AND CAKEYS

14.4.1 STRUCTURE

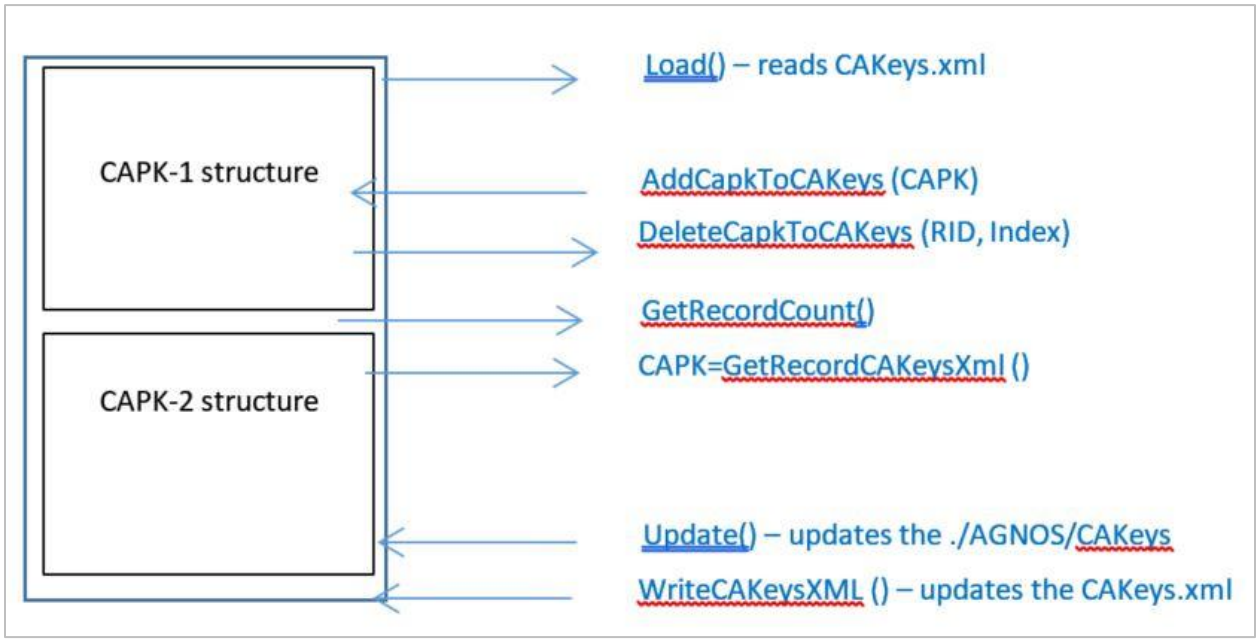

Figure 11 - CA Key File Structure.

1) CAKeys.xml

The CAKeys.xml contains a list of CAPK in readable XML format that can be used to update CAKeys binary file.

```
<CAPKLIST>
      <COUNT>1</COUNT>
      <_1>
            <RID>A000000004</RID>
            <INDEX>FE</INDEX>
<MODULUS>A653EAC1C0F786C8724F737F172997D63D1C3251C44402049B865BAE877
D0F398CBFBE8A6035E24AFA086BEFDE9351E54B95708EE672F0968BCD50DCE40F783
322B2ABA04EF137EF18ABF03C7DBC5813AEAEF3AA7797BA15DF7D5BA1CBAF7FD520B
5A482D8D3FEE105077871113E23A49AF3926554A70FE10ED728CF793B62A1</MODUL
US>
            <EXPONENT>03</EXPONENT>
            <EXPDATE>311220</EXPDATE>

      <CHECKSUM>9A295B05FB390EF7923F57618A9FDA2941FC34E0</CHECKSUM>
      </_1>
</CAPKLIST>
```

The AMPAgnosConfig module checks the EXPDATE (ddmmyy) against the current date to check if the CAPK item is already expired. If it is expired, it will not be added to the CAKeys binary file.

The AMPAgnosConfig also validates the input CHECKSUM against the calculated checksum, if it does not match then it will not be added to the CAKeys binary file.

2) CAKeys

The CAKeys file is the actual file that the AMPEMVL2 library reads during data authentication for contact and contactless processing. It can be generated and updated via the CAKeys.xml API interface.

Field	Length	Notes
RID	unsigned char [5]	
Index	unsigned char	
Exponent Length	unsigned char	1 or 3 as per EMV specification
Modulus Length	unsigned char	Max 248 bytes as EMV specification
Exponent	unsigned char [ExpoLen]	
Modulus	unsigned char [ModulusLen]	

14.4.2 CHECKSUM CALCULATION

The checksum calculation is a SHA-1 20bytes algorithm of the Data Buffer as an input

Where: DataBuffer = RID+Index+Modulus+Exponent

14.4.3 API USAGE

- LoadAgnosConfig(CAKEYS_CFG) - reads CAKeys.xml
- AddCapkToCAKeys(&Capk) - adds CAPK in CAKeys list
- GetRecordCountAgnosConfig(CAKEYS_CFG) - gets record count
- GetRecordAgnosConfig(CAKEYS_CFG, iIndex, Data, &DataLen) - gets data buffer
- WriteCAKeysXML() - updates CAKeys.xml
- UpdateAgnosConfig(CAKEYS_CFG) - updates CAKeys

14.4.4 CODE SAMPLE

```
int DoAgnosCAKeysConfig(void)
{
	int iRetVal = AMP_SUCCESS;
	cAMPAgnosConfig *pConfig = cAMPAgnosConfig::get_Instance();
	tCAPK Capk;
	int iIndex=0;
	int iRecCount=0;
	BYTE Data[MAX_PARAM_DATA_SIZE]={0};
```

```
unsigned long DataLen=0;

//Reads the CAKeys.xml to update CAKeys config file
//Reads CAKeys XML file (./CAKeys.xml)
pConfig->LoadAgnosConfig(CAKEYS_CFG);

// Add routines for add/delete of CAPK record
//
//setup Capk

//Add to capk list
//pConfig->AddCapkToCAKeys(&Capk);

//Get Capk Record Count
iRecCount = pConfig->GetRecordCountAgnosConfig(CAKEYS_CFG);

//Traverse Record
for(iIndex=1; iIndex<=iRecCount; iIndex++)
{
      pConfig->GetRecordAgnosConfig(
            CAKEYS_CFG, iIndex, Data, &DataLen);
}

//Generate XML file if you wish to update XML
//    pConfig->WriteCAKeysXML();

//Writes   the   loaded   data   to   CAKeys   config   file
(./AGNOS/CAKeys), it also clears data in memory
      pConfig->UpdateAgnosConfig(CAKEYS_CFG);

      return iRetVal;
}
```

14.5 CRL

The CRL file contains the Certification Revocation List which can be used to revoke the CAPK list.

14.5.1 STRUCTURE

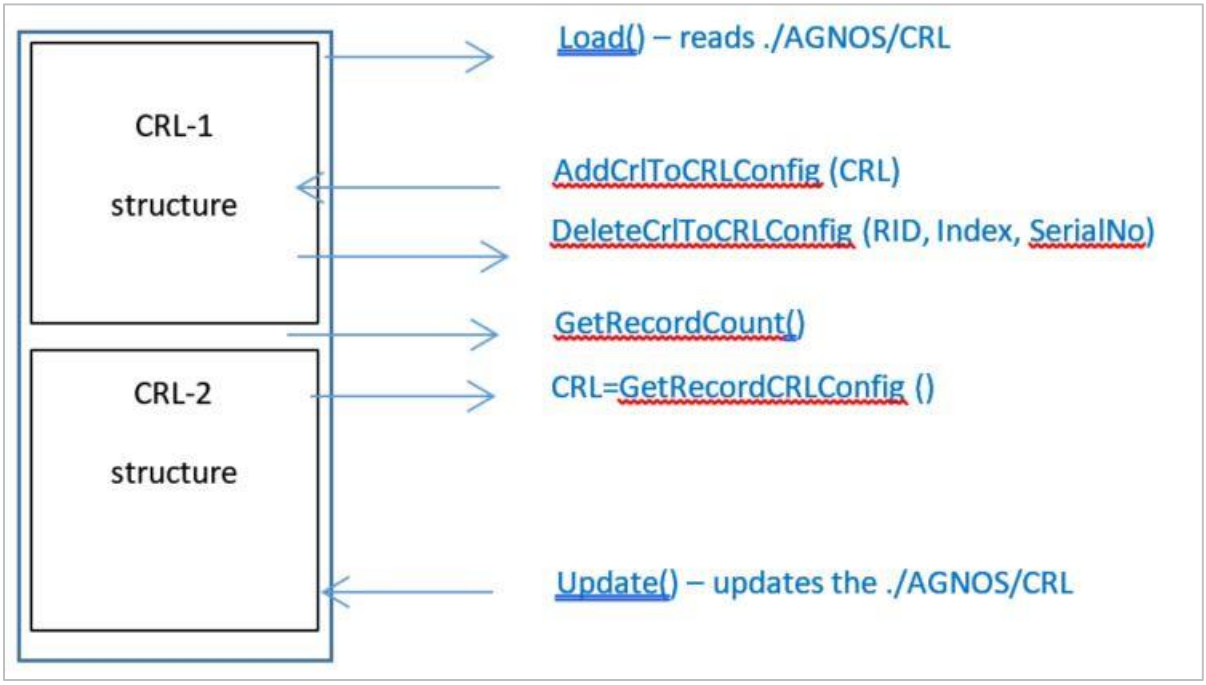

Figure 12 - CRL File Structure.

Field	Length	Notes
RID	unsigned char [5]	
Index	unsigned char	
Certificate Serial Number	unsigned char [3]	

14.5.2 API USAGE

- LoadAgnosConfig(CRL_CFG);
- AddCrlToCRLConfig(&Crl);
- GetRecordCountAgnosConfig(CRL_CFG);
- GetRecordAgnosConfig(CRL_CFG, iIndex, Data, &DataLen);
- UpdateAgnosConfig(CRL_CFG);

14.5.3 CODE SAMPLE

```
int DoAgnosCRLConfig(void)
{
	int iRetVal = AMP_SUCCESS;
	cAMPAgnosConfig *pConfig = cAMPAgnosConfig::get_Instance();
	tCRL Crl;
```

```
        int iIndex=0;
        int iRecCount=0;
        BYTE Data[MAX_PARAM_DATA_SIZE]={0};
        unsigned long DataLen=0;

        //Reads CRL config file
        pConfig->LoadAgnosConfig(CRL_CFG);

        //3 Add routines for add/delete of CRL record
        //
        //setup CRL

        //Add to CRL
        //pConfig->AddCrlToCRLConfig(&Crl);

        //Get CRL Record Count
        iRecCount = pConfig->GetRecordCountAgnosConfig(CRL_CFG);

        //Traverse Record
        for(iIndex=1; iIndex<=iRecCount; iIndex++)
        {
              pConfig->GetRecordAgnosConfig(
                    CRL_CFG, iIndex, Data, &DataLen);

        }

        //Writes the loaded data to CRL config file (./AGNOS/CRL), it
        also clears data in memory
        pConfig->UpdateAgnosConfig(CRL_CFG);

        return iRetVal;
}
```

15 INI AND APPS FILES

15.1 AGNOS.INI

This is an ini file that contains parameters or settings for the payment application. This file is exclusively used by developers. It should not be modified by merchants.

```
[Agnos]
Path=./AGNOS/
ReaderK=OMNIKEY          CardMan          (076B:5421)          5421
(OKCM00706101415205652984335512 66) 00 00
ReaderCL=OMNIKEY          CardMan          (076B:5421)          5421
(OKCM00706101415205652984335512 66) 00 01
```

```
Trace=0
Language=lang.ini
; MODE
; 1: Automation
; 2: SDK (ACE)
Mode=2
FileIntegrity=No
CL=Yes
DefaultLanguage=1
AccountSelection=No
CurrencyCode=0840
USDebitAppSelect=Yes
QuickChip=Yes
[Contactless]
KeyIn=Yes
FallbackAID=No
EntryPoint=No
Default9C=2
AlwaysReceipt=No
TransactionLooping=No
Signal=Yes
ErrorLEDDisplay=Yes
EnableMifareDetect=No
[COM]
Address=192.168.1.163
Port=5560
;Address=localhost
;Port=5561
[Callback]
DE=No
EMVSelection=No
KernelSelection=No
LanguageSelection=No
AmountSelection=No
AmountSelection=No
```

15.2 LANG.INI

This ini file contains a list of standard messages defined in EMVCo Book-4 with different languages.

```
[Language]
; Language at position 1 acts as Default
1=en
2=fr
[en]
Label=ENGLISH
; Standard Messages EMVCo Book IV, section 11.2
```

```
01=AMOUNT
02=AMOUNT OK?
03=APPROVED
04=CALL YOUR BANK
05=CANCEL OR ENTER
06=CARD ERROR
07=DECLINED
08=ENTER AMOUNT
09=ENTER PIN
0A=INCORRECT PIN
0B=INSERT CARD
0C=NOT ACCEPTED
0D=PIN OK
0E=PLEASE WAIT...
0F=PROCESSING ERROR
10=REMOVE CARD
11=USE CHIP READER
12=USE MAGSTRIPE
13=TRY AGAIN
; Contactless Specific Messages EMVCo Book A, section 9.4
14=WELCOME
15=PRESENT CARD
16=PROCESSING
17=CARD READ OK - REMOVE CARD
18=INSERT OR SWIPE CARD
19=PRESENT ONE CARD ONLY
1A=APPROVED - SIGN
1B=AUTHORIZING WAIT
1C=ERROR - USE OTHER CARD
1D=PRESENT CARD AGAIN
1E=CLEAR DISPLAY
20=SEE PHONE
21=TRY AGAIN
22=INSERT/SWIPE/TRY OTHER CARD
; Acquirer Specific Messages
81=SELECT ACCOUNT
82=MAG PROCESSING
83=SELECT LANGUAGE
84=CANCELLED
85=SWIPE CARD
86=FAILED
87=CASHBACK
88=CASHBACK OK?
89=REFERRAL?
8A=THANK YOU
8B=TRANSACTION NOT PERMITTED
check=325829E28E2CDB90
[fr]
Label=FRANCAIS
```

```
; Standard Messages EMVCo Book IV, section 11.2
01=MONTANT
02=MONTANT OK?
03=APPROUVE
04=APPELER BANQUE
05=ANNULER OU VALIDER
06=ERREUR CARTE
07=REFUSE
08=ENTRER MONTANT
09=ENTRER CODE
0A=MAUVAIS CODE
0B=INSERER CARTE
0C=NON ACCEPTE
0D=CODE OK
0E=PATIENTEZ...
0F=ERREUR DE TRAITEMENT
10=ENLEVER CARTE
11=UTILISER PUCE
12=UTILISER PISTE
13=REESSAYEZ
; Contactless Specific Messages EMVCo Book A, section 9.4
14=BIENVENUE
15=PRESENTER CARTE
16=TRAITEMENT
17=CARTE LUE OK - RETIRER CARTE
18=INSERER/GLISSER CARTE
19=PRESENTER UNE SEULE CARTE
1A=APPROUVE - SIGNER
1B=AUTHORISATION PATIENTEZ
1C=ERREUR - UTILISER AUTRE CARTE
1D=INSERER CARTE
1E=RETIRER CARTE
20=VOIR TELEPHONE
21=REESSAYEZ
22=UTILISER AUTRE CARTE
; Acquirer Specific Messages
81=CHOISIR COMPTE
82=TRAITEMENT PISTE
83=CHOISIR LANGUE
84=ANNULE
85=GLISSER CARTE
86=ECHOUE
87=RETRAIT?
88=RETRAIT OK?
89=RENVOI?
8A=MERCI
8B=TRANSACTION NON PERMISE
check=CE0A913A40DA7140
```

15.2.1 ADDCHECK

A tool that allows string sets generation for each different language supported by the system. Moreover, it adds a checksum value for each language in the lang.ini file.

To execute the program:

Run ./AddCheck file

15.3 APPS

This file is used as an input during AMPEMVL2 library initialization (see Initialize()) – where it will load the listed contactless application kernels that were previously defined in the apps file.

```
7
libPayPass3x.so
libPayWave2x.so
libXPressPay3x.so
libDPAS10.so
libJSpeedy.so
libFlash.so
libEMVCo.so
```

APPENDIX A

TAG REFERENCE

Tag	Description
42	Issuer Identification Number (IIN)
4F	Application Identifier (AID)
50	Application Label
56	Track 1 Data
57	Track 2 Equivalent Data
5A	Application Primary Account Number (PAN)
61	Application Template
6F	File Control Information (FCI) Template
70	READ RECORD Response Message Template
73	Directory Discretionary Template
77	Response Message Template Format 2
80	Response Message Template Format 1
81	Amount, Authorized (Binary)
82	Application Interchange Profile
84	Dedicated File (DF) Name
87	Application Priority Indicator
88	Short File Identifier (SFI)
8A	Authorization Response Code
8C	Card Risk Management Data Object List 1 (CDOL1)
8D	Card Risk Management Data Object List 2 (CDOL2)
8E	Cardholder Verification Method (CVM) List
8F	Certification Authority Public Key Index
90	Issuer Public Key Certificate

91	Issuer Authentication Data
92	Issuer Public Key Remainder
93	Signed Static Application Data
94	Application File Locator (AFL)
95	Transaction Verification Results (TVR)
97	Transaction Certificate Data Object List (TDOL)
98	Transaction Certificate (TC) Hash Value
99	Transaction Personal Identification Number (PIN) Data
9A	Transaction Date
9B	Transaction Status Information
9C	Transaction Type
9D	Directory Definition File (DDF) Name
A5	File Control Information (FCI) Proprietary Template Agnos Framework
C5	CRM Currency Code
C7	Terminal Transaction Processing Information (TTPI)
CD	Contactless Cryptogram Information Data (CCID)
D1	Offline Balance Limit (Discover)
D2	VIUDS Scheme Directory
5F20	Cardholder Name
5F24	Application Expiration Date
5F25	Application Effective Date
5F28	Issuer Country Code
5F2A	Transaction Currency Code
5F2D	Language Preference
5F30	Service Code
5F34	Application Primary Account (PAN) Sequence Number

5F36	Transaction Currency Exponent
5F50	Issuer URL
5F53	International Bank Account Number (IBAN)
5F54	Bank Identifier Code (BIC)
5F55	Issuer Country Code (alpha2 format)
5F56	Issuer Country Code (alpha3 format)
5F57	Account Type
9F01	Acquirer Identifier
9F02	Amount, Authorized (Numeric)
9F03	Amount, Other (Numeric)
9F04	Amount, Other (Binary)
9F05	Application Discretionary Data
9F06	Application Identifier
9F07	Application Usage Control
9F08	Application Version Number
9F09	Application Version Number
9F0B	Cardholder Name Extended
9F0D	Issuer Action Code – Default
9F0E	Issuer Action Code – Denial
9F0F	Issuer Action Code – Online
9F10	Issuer Application Data
9F11	Issuer Code Table Index
9F12	Application Preferred Name
9F13	Last Online Application Transaction Counter (ATC) Register
9F14	Lower Consecutive Offline Limit
9F15	Merchant Category Code

9F16	Merchant Identifier
9F17	Personal Identification Number (PIN) Try Counter
9F1A	Terminal Country Code
9F1B	Terminal Floor Limit
9F1C	Terminal Identification
9F1D	Terminal Risk Management Data
9F1E	IFD Serial Number
9F1F	Track 1 Discretionary Data
9F20	Track 2 Discretionary Data
9F21	Transaction Time
9F22	CA PKI fDDA Agnos Framework
9F23	Upper Consecutive Offline Limit
9F26	Application Cryptogram
9F27	Cryptogram Information Data
9F29	Extended Selection
9F2A	Kernel ID
9F2D	Integrated Circuit Card (ICC) PIN Encipherment Public Key Certificate
9F2E	Integrated Circuit Card (ICC) PIN Encipherment Public Key Exponent
9F2F	Integrated Circuit Card (ICC) PIN Encipherment Public Key Remainder
9F32	Issuer Public Key Exponent
9F33	Terminal Capabilities
9F34	Cardholder Verification Method (CVM) Results
9F35	Terminal Type
9F36	Application Transaction Counter
9F37	Unpredictable Number
9F38	Processing Option Data Object List (PDOL)

9F39	POS Entry Mode
9F3A	Amount Reference Currency
9F3B	Application Reference Currency
9F3C	Transaction Reference Currency Code
9F3D	Transaction Reference Currency Exponent
9F40	Additional Terminal Capabilities
9F41	Transaction Sequence Counter
9F42	Application Currency Code
9F43	Application Reference Currency Exponent
9F44	Application Currency Exponent
9F45	Data Authentication Code
9F46	Issuer Circuit Card (ICC) Public Key Certificate
9F47	Issuer Circuit Card (ICC) Public Key Exponent
9F48	Issuer Circuit Card (ICC) Public Key Remainder
9F49	Dynamic Data Authentication Data Object List (DDOL)
9F4A	Static Data Authentication Tag List
9F4B	Signed Dynamic Application Data
9F4C	ICC Dynamic Number
9F4D	Log Entry
9F4E	Merchant Name and Location
9F4F	Log Format
9F50	Offline Accumulator Balance
9F51	DR DOL (Mastercard) or Application Currency Code (Visa)
9F53	Transaction Category Code (Mastercard) or Consecutive Transaction Counter International Limit (CTTIL) (Visa)
9F54	DS ODS Card (Mastercard) or Cumulative Total Transaction Amount Limit (CTTAL) (Visa)

9F55	RFU
9F5A	Application Program Identifier
9F5B	DSDOL (Mastercard) or Issuer Script Results (Visa)
9F5C	DS Requested Operator ID (Mastercard) or Cumulative Total Counter Upper Limit (CTCUL) (Visa)
9F5D	Application Capabilities Information (Mastercard) or Available Offline Spending Amount (AOSA) (Visa)
9F5E	DS ID (Mastercard) or Consecutive Transaction International Upper Limit (CTIUL) (Visa)
9F60	CVC3 (Track1)
9F61	CVC3 (Track2)
9F62	PCVC3 (Track1)
9F63	PUNATC (Track1) (Mastercard) or Offline Counter Initial Value (Visa)
9F64	NATC (Track1)
9F65	PCVC3 (Track2)
9F66	PUNATC (Track2) (Mastercard) or Terminal Transaction Qualifiers (TTQ) (Visa)
9F67	NATC (Track2) (Mastercard) or MSD Offset (Visa)
9F68	Card Additional Processes (CAP)
9F69	UDOL (Mastercard) or Card Authentication Related Data (Visa)
9F6A	Unpredictable Number (Numeric)
9F6B	Track 2 Data (Mastercard and AMEX) or Card CVM Limit (Visa)
9F6C	Card Transaction Qualifiers (CTQ) or Application Version Number (Mastercard)
9F6D	Magstripe Application Version Number (Reader) or Modified Terminal Capabilities (AMEX)
9F6E	Third Party Data (Mastercard) or Form Factor Indication (FFI) (Visa)
9F6F	DS Slot Management Control
9F70	Protected Data Envelope 1 (Mastercard) or Card Interface Capabilities (AMEX)

9F71	Protected Data Envelope 2 (Mastercard)
9F72	Protected Data Envelope 3 (Mastercard)
9F73	Protected Data Envelope 4 (Mastercard) or Currency Conversion Parameters (Visa)
9F74	Protected Data Envelope 5 (Mastercard)
9F75	Unprotected Data Envelope 1 (Mastercard)
9F76	Unprotected Data Envelope 2 (Mastercard)
9F77	Unprotected Data Envelope 3 (Mastercard) or VLP Funds Limit (Visa)
9F78	Unprotected Data Envelope 4 (Mastercard) or VLP Single Transaction Limit (Visa)
9F79	Unprotected Data Envelope 5 (Mastercard) or VLP Available Funds (Visa)
9F7C	Merchant Custom Data (Mastercard) or Customer Exclusive Data (CED) (Visa)
9F7D	DS Summary 1 (Mastercard) or Application Version Number (Discover)
9F7E	DCVV Discover version 1 or Mobile Support Indicator (Mastercard)
9F7F	DS Unpredictable
9F80	DCVV Discover version 2
BF0C	File Control Information (FCI) Issuer Discretionary Data
BF5F	Scheme Data Template
BF70	Memory Slot Update Template (PURE)
DF01	RFU
DF02	RFU
DF03	Security Capabilities a.k.a 9F33's Byte3
DF04	Balance Read Before GAC
DF05	Balance Read After GAC
DF06	RFU
DF07	RFU
DF08	RFU

DF09	RFU
DF0A	RFU
DF0B	RFU
DF0C	Kernel ID
DF0D	RFU
DF0E	RFU
DF0F	RFU Agnos Framework
DF10	RFU
DF11	RFU
DF12	RFU
DF13	RFU
DF14	RFU
DF15	RFU
DF16	RFU
DF17	Card Data Input Capability a.k.a. 9F33's Byte1
DF18	CVM Capability - CVM Required a.k.a. 9F33's Byte2
DF19	CVM Capability - No CVM Required a.k.a. 9F33's Byte2
DF1A	Default UDOL
DF1B	Kernel Configuration
DF1C	Max Lifetime Torn Transaction
DF1D	Max Number Torn Transaction
DF1E	Magstripe CVM Capabilities - CVM Required
DF20	TAC – Default
DF21	TAC – Denial
DF22	TAC – Online
DF23	RCFL (Reader Contactless Floor Limit)

DF24	RCTL (No On-device CVM) Reader Contactless Transacion Limit
DF25	RCTL (On-device CVM)
DF26	RCRL (Reader CVM Required Limit)
DF27	Timeout Value
DF2C	Magstripe CVM Capabilities - No CVM Required
DF2D	Message Hold time
DF30	Bitmap Entry Point
DF31	Online Tag List
DF32	Status Zero Amount Allowed Flag
DF33	Extended Selection Flag
DF34	Mandatory Tag Object List (MTOL)
DF35	Authentication Data Tag Object (ATDTOL)
DF41	RFU
DF4B	POS Cardholder Interaction Information
DF51	RFU
DF60	DS Input (Card)
DF61	DS Digest H (Mastercard)
DF62	DS ODS Info
DF63	DS ODS Term
DF71	RFU

APPENDIX B

EMV CONTACT FLOW

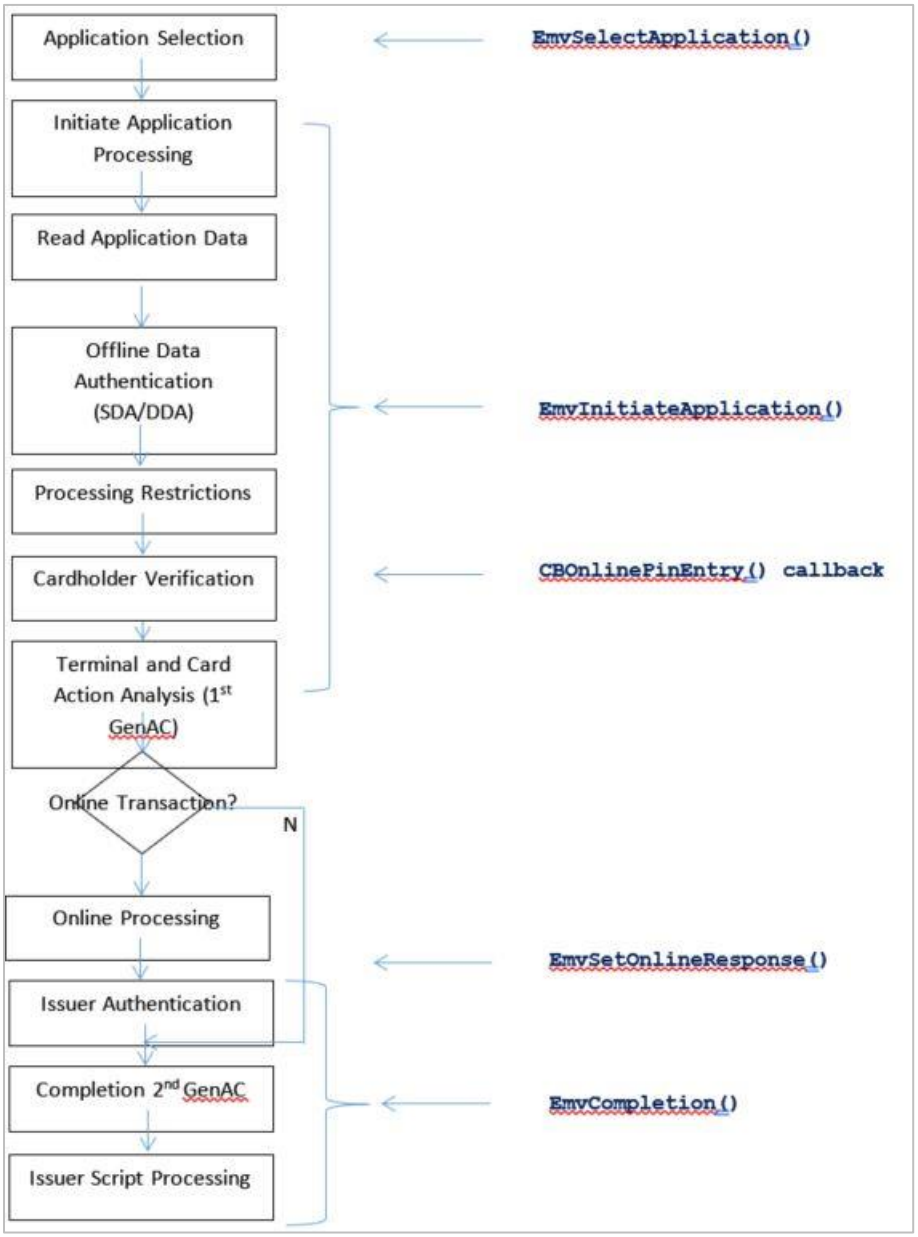

Figure 13 - EMV Contact Flow.

EMV CONTACTLESS

Reference: EMV Contactless Specification for Payments Systems Book A

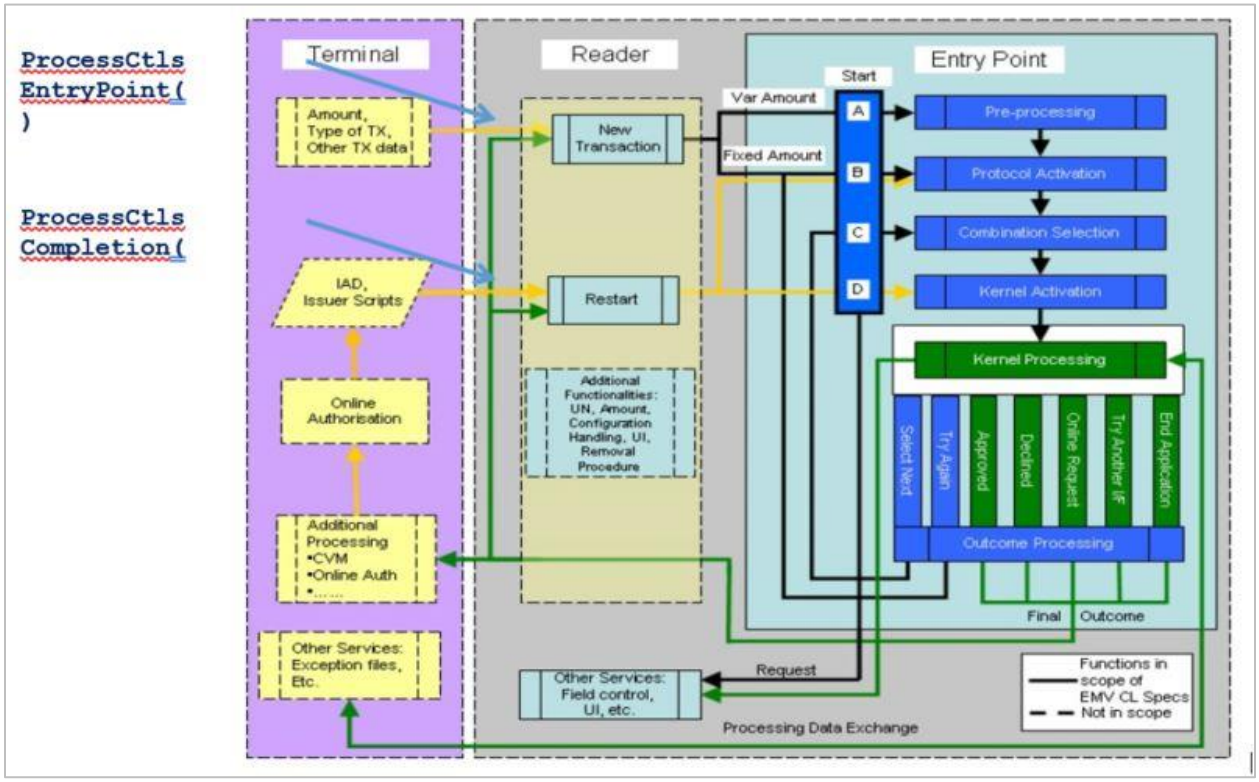

Figure 14 - EMV Contactless Flow.

APPENDIX C

TECHNICAL NOTES/LIMITATIONS

1) Adding AID records in the PROCESSING configuration

It is important to add Contactless AIDs first before Contact AIDs as depicted through the following figure.

For contactless AIDs, use the first byte of the Acquirer ID field to indicate the Kernel ID.

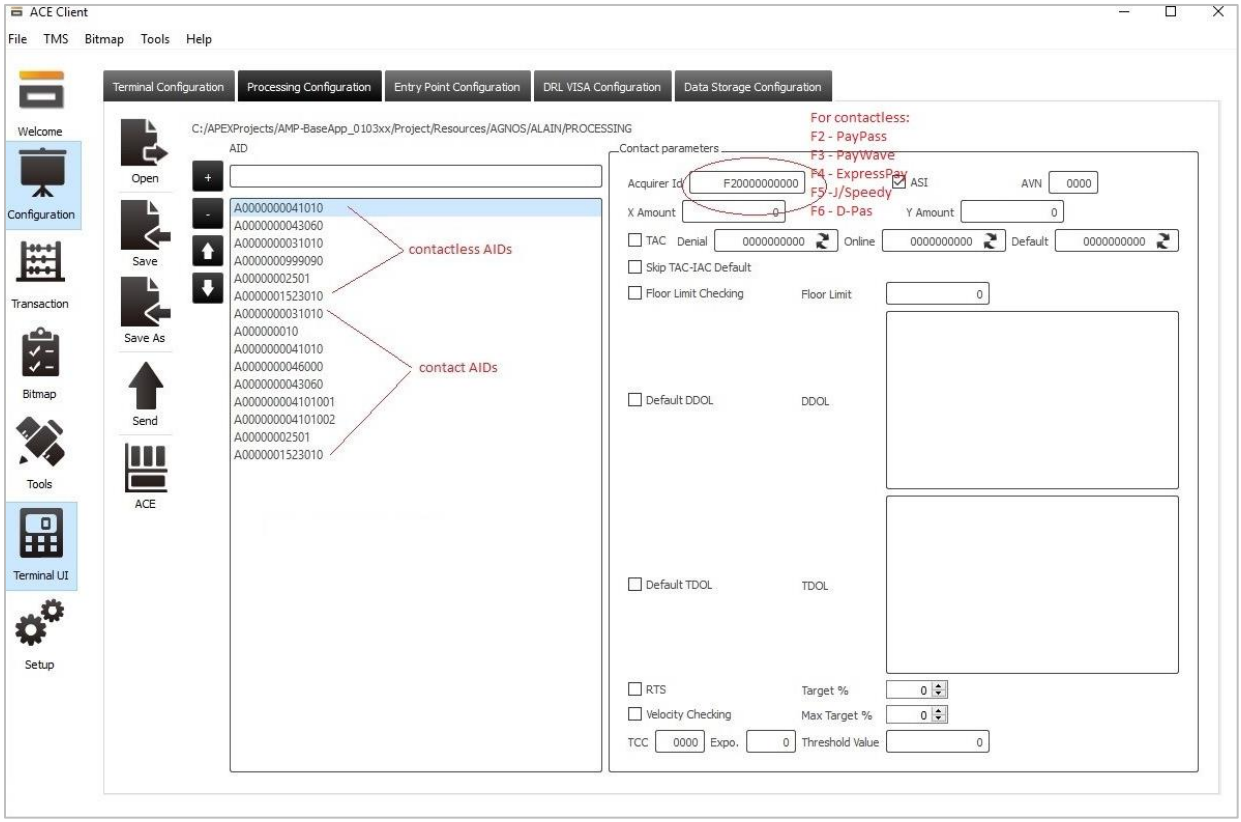

Figure 15 - Adding AID records in the processing configuration.

2) Use of DirectFB library of application layer

AMPEMVL2 library uses DirectFB for display and key press events, if the application layer is also utilizing the DirectFB, ensure that the application sets the window focus to avoid key press conflicts. See the sample code below.

```
int kbhit(void)
{
      DFBWindowEvent  windowEvent;
      DFBResult ret;

      global_window->RequestFocus(global_window);
```

```
      while(1){
      if (global_events->HasEvent(global_events) == DFB_OK){
      if (global_events->PeekEvent(global_events,
      DFB_EVENT(&windowEvent)) == DFB_OK) {
      if (/*windowEvent.type == DWET_KEYDOWN || */ windowEvent.type
                  == DWET_KEYUP)
      return windowEvent.type;

      ret = global_events->GetEvent(global_events,
                                    DFB_EVENT(&windowEvent));
      if(ret){
            DirectFBError("IDirectFBEventBuffer::GetEvent() failed",
            ret);
            }
         }
      } else {
            break;
      }
   }//end while

      return 0;
}

int kbGetKey(void)
{
   DFBWindowEvent windowEvent;
   DFBResult ret;

   global_window->RequestFocus(global_window);

   while (1){
      global_events->WaitForEvent(global_events);
      if ((ret = global_events->GetEvent(global_events,
DFB_EVENT(&windowEvent))) == DFB_OK)
{
            if (windowEvent.type == DWET_KEYUP)
                  return windowEvent.key_symbol;
      } else
{
            break;
      }
   }
      return 0;
}
```

3) Adding Terminal Risks Management Data (9F1D)

Terminal Risk Management Data is an EMV data object whose format is an application specific value used by the contactless card for risk management purposes. It can be added and configured through ENTRY_POINT as shown below.

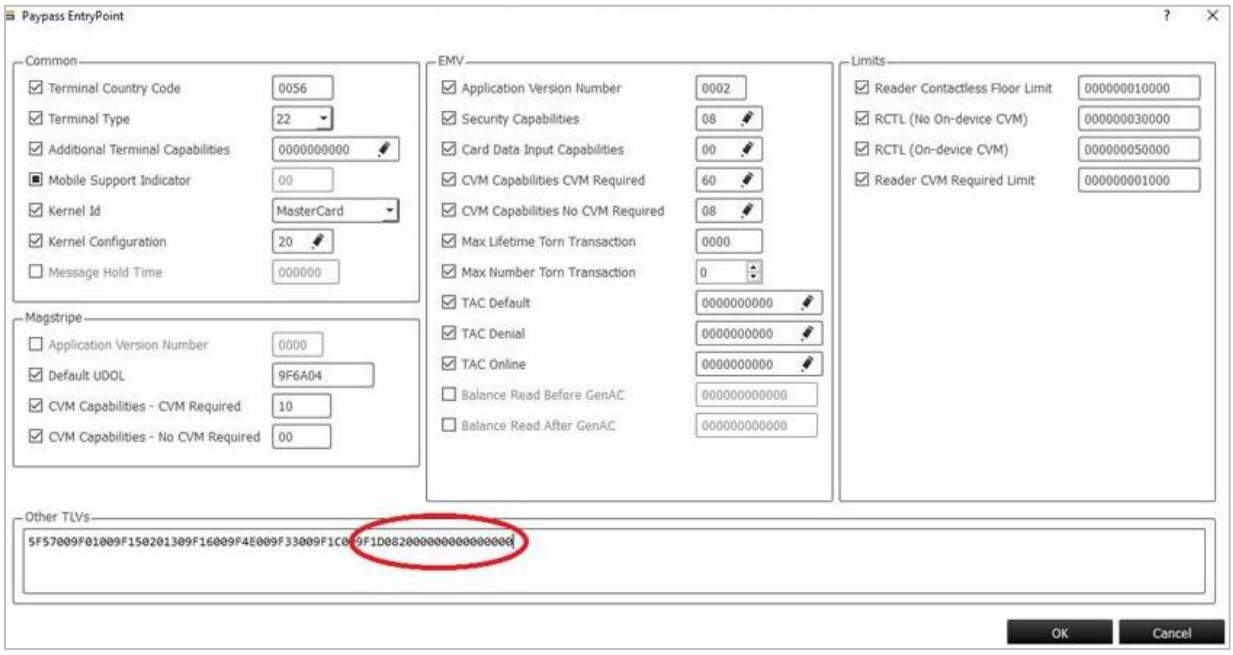

Figure 16 - Adding Terminal Risks Management Data through the ENTRY_POINT.