Manual For Kiosk III SDK

ID-TECH(ShangHai)

11/04/2014

Target Device:

Kiosk III.

Describe:

Support Kiosk III device.

Platform:

Microsoft Window Serial.

DLL Usage (Microsoft Visual C++ 6.0)

Add KioskDLL.lib to Project->Settings->Link->Object/library modules and include the head file KioskDLL.h , then call the DLL function directly.

Command Summary

All commands supported are listed below.

SysInitDevice
SysCloseDevice
Com_OpenUSBHID
Com_OpenPort

GT_Ping
GT_SetPollMode
GT_ControlUI
GT_GetSource
GT_GetConfiguration
GT_SetConfiguration
GT_ReadDownloadModeWithoutRes
GT_ReadDownloadModeWithRes
GT_GetVersion
GT_GetBootLoaderVersion
GT_SetBaudRate
GT_SetSerialNumber
GT_GetSerialNumber

CC_SetConfigurableGroup
CC_GetConfigurableGroup
CC_SetConfigurableAID
CC_GetConfigurableAID
CC_GetAllGroup
CC_GetAllAID

TC_ActiveTransaction
TC_UpdateBalance
TC_GetTransaction
TC_CancelTransaction

MC_StopTransaction
MC_ResetTornTransactionLog
MC_ClearTornTransactionLog

VT_SetCashTransactionRiskParam
VT_GetCashTransactionRiskParam
VT_SetCashbackTransactionRiskParam
VT_GetCashbackTransactionRiskParam
VT_SetDRLReaderRiskParam
VT_GetDRLReaderRiskParam

PK_GetCAPublicKey
PK_SetCAPublicKey
PK_GetAllCAPublicRID
PK_GetAllCAPublicKeyID
PK_GetCAPublicKeyHash
PK_DeleteCAPublicKey
PK_DeleteAllCAPublicKey

MV_GetProcessorType
MV_GetHarewareInfo
MV_GetModuleVersionInfo

EMV_GetRevocationLogStatus
EMV_AddEntryToRevocation
EMV_GetRevocationList
EMV_DeleteRevocationListEntry
EMV_DeleteAllRevocationListEntry
EMV_DeleteOneRevocationListEntry

PT_ModeStartStop
PT_GetPCDPICCParam
PT_PollForToken
PT_EnhancePollForToken
PT_GetATR
PT_AntennaControl
PT_LEDControl
PT_BuzzerControl

PT_ExchangeContactlessData
PT_ExchangeSingleCommand
PT_HighLevelHalt
PT_EnhancePassThroughControl

HL_MifareAuthenticateBlock
HL_MifareReadBlock
HL_ExchangeAPDUData
HL_MifareWriteBlock
HL_PurseValueBlock

SC_InitiateSecureComm
SC_InstallKey

TR_FlushTrackData
TR_GetFullTrackData
TR_GetFirmwareVersion
TR_SetRFErrorReport
TR_SetTime
TR_GetTime
TR_SetDate
TR_GetDate

Function description

DLL Version

Function:	SysInitDevice
Description:	Automatic identify USB or RS232 Port, If open device successfully, will return TRUE, else will return FALSE.
Format:	BOOL WINAPI SysInitDevice(void).
Parameter:	None
Return:	BOOL
Example	SysInitDevice( )

Function:	SysCloseDevice	
Description:	Close port for RS232 or HID communication.	
Format:	VOID WINAPI SysCloseDevice( )	
Parameter:	None	
Return:	VOID	
Example	SysCloseDevice( )	
Function:	Com_OpenUSBHID	
Description:	Open USBHID port to communicate	
Format:	BOOL WINAPI Com_OpenUSBHID()	
Parameter:	None	
Return:	BOOL	
Example	Com_OpenUSBHID( )	
Function:	Com_OpenPort	
Description:	Open port for RS232 communication.	
Format:	BOOL WINAPI Com_OpenPort(UINT32 nComport,LONG32 nBaud, CHAR cParity, UINT32 nStop, UINT32 nData)	
Parameter:	nComport	Port for RS232 communication
	nBaud	Baud rate
	cParity	Parity bit
	nStop	Stop bit
	nData	Data bit
Return:	BOOL	
Example	Com_OpenPort (1,9600, 'E', 1, 7);	
Function:	GT_Ping	
Description:	Check if the device reader is connected to the terminal	
Format:	BYTE WINAPI GT_Ping( )	
Parameter:	None	
Return:	Protocol II Response.	
Example	GT_Ping( )	
Function:	GT_SetPollMode	
Description:	This Command allow the teminal to set the device Reader polling mode	
Format:	BYTE WINAPI GT_SetPollMode(BOOL bDemand)	
Parameter:	bDemand	FALSE:Auto Poll TRUE:Poll on Demand
Return:	Protocol II Response.	
Example	GT_SetPollMode(TRUE)	
Function:	GT_ControlUI	
Description:	Instruct the reader to change LED behavior, and beep. Each action is	

	controlled independently by value in the parameters.	
Format:	BYTE WINAPI GT_ControlUI(BYTE nLCDMsg,BYTE nBeepIndicator, BYTE nLEDNum,BYTE nLEDStatus)	
Parameter:	nLCDMsg	Messages 00-07 are normally controlled by the reader. 00: Idle Message (Welcome) 01: Present card (Please Present Card) 02: Time Out or Transaction cancel (No Card) 03: Transaction between reader and card is in the middle (Processing…) 04: Transaction Pass (Thank You) 05: Transaction Fail (Fail) 06: Amount (Amount $ 0.00 Tap Card) 07: Balance or Offline Available funds (Balance $ 0.00) Messages 08-0B are controlled by the terminal 08: Insert or Swipe card (Use Chip & PIN) 09: Try Again(Tap Again) 0A: Tells the customer to present only one card (Present 1 card only) 0B: Tells the customer to wait for authentication/authorization (Wait) FF indicates the command is setting the LED/Buzzer only.
	nBeepIndicator	00h: No beep 01h: Single beep 02h: Double beep 03h: Three short beeps 04h: Four short beeps 05h: One long beep of 200 ms 06h: One long beep of 400 ms 07h: One long beep of 600 ms 08h: One long beep of 800 ms
	nLEDNum	00h: LED 0 (Power LED) 01h: LED 1 02h: LED 2 03h: LED 3

				FFh: All LEDs Where the LEDs are numbered 0, 1, 2, 3 counting from the left. Note: You can attempt to control the Power LED (LED 0) but other UI behavior takes control, so attempting to manipulate the Power LED in non-pass-through mode has no effect.
	nLEDStatus			00h: LED Off 01h: LED On
Return:	Protocol II Response			
Example	BYTE nLCDMsg = 0xFF; BYTE nBeepIndicator = 0x08; BYTE nLEDNum = 0xFF; BYTE nLEDStatus = 0x01; GT_ControlUI(nLCDMsg, nBeepIndicator, nLEDNum, nLEDStatus);			
Function:	GT_GetSource			
Description:	Get the source for RTC/LCD/Buzzer/LED on the device reader.			
Format:	BYTE WINAPI GT_GetSource(BYTE & nBuzzer, BYTE & nLED)			
Parameter:	nBuzzer	Bit1 Bit0	00	Don't User Buzzer
			01	User Buzzer from device reader
			10	User Buzzer from external source
			11	User Buzzer from both reader and external source
		Bit3 Bit2	00	Don't user LCD
			01	Use LCD from device reader
			10	Use LCD from external source
			11	Use LCD from both reader and external source
		Bit5 Bit4	00	Don't use RTC
			01	Use RTC from device reader
			10	Not allowed
			11	Not allowed
		Bit7 Bit6	Reserved	
	nLED	Bit1 Bit0	00	Don't use transaction LED
			01	Use transaction LED from device readr
			10	Use transaction LED from external source
			11	Use transaction LED from

				both reader and external source
		Bit3 Bit2	00	Don't use power LED
			01	Use power LED from device reader
			10	Use power LED from external source
			11	Use power LED from both reader and external source
		Bit5 Bit4	Reserved	
		Bit7 Bit6	Reserved	
Return:	Protocol II Response			
Example	BYTE nBuzzer = 0; BYTE nLED = 0; GT_GetSource(nBuzzer, nLED);			
Function:	GT_GetConfiguration			
Description:	Use this function to return the values of the TLV global data object and default group data objects(TLV Group 0) in the reader from the nonvolatile memory.			
Format:	BYTE WINAPI GT_GetConfiguration(BYTE * pTLVData, UINT32 & nTLVDataLen)			
Parameter:	pTLVData	The TLV data address.		
	nTLVDataLen	The TLV data length.		
Return:	Protocol II Response			
Example	BYTE arrTLVData[1024] = {0}; UINT32 nTLVDataLen = 0; GT_GetConfiguration(arrTLVData, nTLVDataLen);			
Function:	GT_SetConfiguration			
Description:	Use this function to set or change the value of the special TLV data objects in the reader.			
Format:	BYTE WINAPI GT_SetConfiguration(BYTE * pTLVData, UINT32 & nTLVDataLen)			
Parameter:	pTLVData	Global Configuration TLVs		
	nTLVDataLen	The length of TLV data objects.		
Return:	Protocol II Response			

Example	BYTE arrSend[512] = {0};
	UINT nSendLen = 0;
	arrSend[nSendLen++] = 0x9F;
	arrSend[nSendLen++] = 0x1A;
	arrSend[nSendLen++] = 0x02;
	arrSend[nSendLen++] = 0x08;			

	arrSend[nSendLen++] = 0x40;
	arrSend[nSendLen++] = 0x5F;
	arrSend[nSendLen++] = 0x2A;
	arrSend[nSendLen++] = 0x02;
	arrSend[nSendLen++] = 0x08;
	arrSend[nSendLen++] = 0x40;
	GT_SetConfiguration(arrSend,nSendLen);	

Function:	GT_ReadDownloadModeWithoutRes	
Description:	This function instructs the reader to switch to In System Programming mode, also known as Download Mode.	
Format:	VOID WINAPI GT_ReadDownloadModeWithoutRes(BYTE nTime)	
Parameter:	nTime	The Max Download Time Code is the maximum time in multiples of 10 seconds allowed to download the firmware.
Return:	Protocol II Response	
Example	GT_ReadDownloadModeWithoutRes(0x01)	
Function:	GT_ReadDownloadModeWithRes	
Description:	This command instructs the reader to switch to In System Programming (ISP) mode, also known as Download Mode. Once the reader is in ISP Mode, the terminal can download firmware.	
Format:	BYTE WINAPI GT_ReadDownloadModeWithRes(BYTE nTime)	
Parameter:	nTime	The Max Download Time Code is the maximum time in multiples of 10 seconds allowed to download the firmware.
Return:	Protocol II Response.	
Example	GT_ReadDownloadModeWithRes(nTime)	
Function:	GT_GetVersion	
Description:	Get the Firmware Version Number from the reader.	
Format:	BYTE WINAPI GT_GetVersion(BYTE * pVersion, UINT32 & nVersionLen);	
Parameter:	pVersion	
	nVersionLen	
Return:	Protocol II Response	
Example	BYTE arrVersion[512] = {0}; UINT32 nVersionLen = 0; GT_GetVersion(arrVersion, nVersionLen);	
Function:	GT_GetBootLoaderVersion	
Description:	Get the version of the USB Boot Loader.	
Format:	BYTE WINAPI GT_GetBootLoaderVersion(BYTE * pVersion, UINT32 & nVersionLen);	
Parameter:	pVersion	

	nVersionLen		
Return:	Protocol II Response.		
Example	BYTE arr arrVersion[512] = {0}; UINT32 nVersionLen = 0; GT_GetBootLoaderVersion		
Function:	GT_SetBaudRate		
Description:	This function instructs the reader to change its baud rate to the special Value.		
Format:	BYTE WINAPI GT_SetBaudRate(BYTE nBaudRateCode);		
Parameter:	nBaudRateCode	01h	9600baud
		02h	19200baud
		03h	38400baud
		04h	57600baud
		05h	115200baud
Return:	Protocol II Response		
Example	GT_SetBaudRate (0x01)		
Function:	GT_SetSerialNumber		
Description:	This function instructs the device to store the 15-digit serial number in its non-volatile momery.		
Format:	BYTE WINAPI GT_SetSerialNumber(BYTE arrSerialNumber[15])		
Parameter:	arrSerialNumber		
Return:	Protocol II Response		
Example	BYTE arrSerialNumber[15] = {0x00,0x00,0x00,0x00,0x00 0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00}; GT_SetSerialNumber(arrSerialNumber);		
Function:	GT_GetSerialNumber		
Description:	This function instructs the device to return 15-digit serial number stored in its non-volatile memery.		
Format:	BYTE WINAPI GT_GetSerialNumber(BYTE * pSerialNumber, UINT & nSerialNumberLen)		
Parameter:	pSerialNumber: The address of serial number string. nSerialNumberLen: The length of serial number string.		
Return:	Protocol II Response		
Example	BYTE arrSerialNumber[512] = {0}; UINT nSerialNumberLen = 0; GT_GetSerialNumber(arrSerialNumber, nSerialNumberLen);		
Function:	CC_SetConfigurableGroup		
Description:	This command creates or modifies a TLV Group.		
Format:	BYTE WINAPI CC_SetConfigurableGroup(BYTE * pTLVData,		

	UINT32 & nTLVDataLen)	
Parameter:	pTLVData	Group Configuration Tags
	nTLVDataLen	Group configuration Tags Length.
Return:	Protocol II Response	

Example	BYTE arrSend[512] = {0};
	UINT nSendLen = 0;
	arrSend[nSendLen++] = 0xFF;
	arrSend[nSendLen++] = 0xE4;
	arrSend[nSendLen++] = 0x01;
	arrSend[nSendLen++] = 0x01;
	arrSend[nSendLen++] = 0x9F;
	arrSend[nSendLen++] = 0x02;
	arrSend[nSendLen++] = 0x06;
	arrSend[nSendLen++] = 0x00;
	arrSend[nSendLen++] = 0x00;
	arrSend[nSendLen++] = 0x00;
	arrSend[nSendLen++] = 0x00;
	arrSend[nSendLen++] = 0x15;
	arrSend[nSendLen++] = 0x00;
	CC_SetConfigurableGroup(arrSend, nSendLen);	

Function:	CC_GetConfigurableGroup	
Description:	Use this Command to return all TLVs associated the specified Configurable Group.	
Format:	BYTE WINAPI CC_GetConfigurableGroup(BYTE * pInputTLVData, UINT32 & nInputTLVDataLen,BYTE * pOutputTLVData, UINT32 & nOutputTLVDataLen)	
Parameter:	pInputTLVData	The Input TLV data address
	nInputTLVDataLen	The Input TLV data length
	pOutputTLVData	The Output TLV data address
	nOutputTLVDataLen	The Output TLV data length
Return:	Protocol II Response	

Example	BYTE arrInput[512] = {0};
	UINT nInputLen = 0;
	BYTE arrOutput[512] = {0};
	UINT32 nOutputLen = 0;
	arrInput[nInputLen++] = 0xFF;
	arrInput[nInputLen++] = 0xE4;
	arrInput[nInputLen++] = 0x01;
	arrInput[nInputLen++] = 0x00;
	
	CC_GetConfigurableGroup(arrInput,nInputLen,arrOutput,nOutputLen);	

Function:	CC_SetConfigurableAID	

Description:	This function creates or selects an AID for configuration or deletion. There are eight TLVs that can be included in this function.	
Format:	BYTE WINAPI CC_SetConfigurableAID(BYTE * pTLVData, UINT32 & nTLVDataLen);	
Parameter:	pTLVData	AID configuration Tags
	nTLVDataLen	AID configuration Tags Length
Return:	Protocol II Response	

Example	BYTE arrSend[512] = {0};
	UINT32 nSendLen = 0;
	arrSend[nSendLen++] = 0xFF;
	arrSend[nSendLen++] = 0xE4;
	arrSend[nSendLen++] = 0x01;
	arrSend[nSendLen++] = 0x00;
	
	arrSend[nSendLen++] = 0x9F;
	arrSend[nSendLen++] = 0x06;
	arrSend[nSendLen++] = 0x07;
	arrSend[nSendLen++] = 0xA0;
	arrSend[nSendLen++] = 0x00;
	arrSend[nSendLen++] = 0x00;
	arrSend[nSendLen++] = 0x00;
	arrSend[nSendLen++] = 0x04;
	arrSend[nSendLen++] = 0x10;
	arrSend[nSendLen++] = 0x10;
	
	arrSend[nSendLen++] = 0xFF;
	arrSend[nSendLen++] = 0xE1;
	arrSend[nSendLen++] = 0x01;
	arrSend[nSendLen++] = 0x01;
	
	arrSend[nSendLen++] = 0xFF;
	arrSend[nSendLen++] = 0xE5;
	arrSend[nSendLen++] = 0x01;
	arrSend[nSendLen++] = 0x10;
	
	arrSend[nSendLen++] = 0xFF;
	arrSend[nSendLen++] = 0xEA;
	arrSend[nSendLen++] = 0x01;
	arrSend[nSendLen++] = 0x02;
	
	arrSend[nSendLen++] = 0xFF;
	arrSend[nSendLen++] = 0xE3;
	arrSend[nSendLen++] = 0x01;
	arrSend[nSendLen++] = 0x74;	

	arrSend[nSendLen++] = 0xFF;
	arrSend[nSendLen++] = 0xE9;
	arrSend[nSendLen++] = 0x0C;
	arrSend[nSendLen++] = 0x02;
	arrSend[nSendLen++] = 0x00;
	arrSend[nSendLen++] = 0x01;
	arrSend[nSendLen++] = 0x02;
	arrSend[nSendLen++] = 0x20;
	arrSend[nSendLen++] = 0x01;
	arrSend[nSendLen++] = 0x02;
	arrSend[nSendLen++] = 0x01;
	arrSend[nSendLen++] = 0x01;
	arrSend[nSendLen++] = 0x02;
	arrSend[nSendLen++] = 0x09;
	arrSend[nSendLen++] = 0x01;
	CC_SetConfigurableAID(arrSend,nSendLen);

Function:	CC_GetConfigurableAID
Description:	This function returns the configurable AID parameters. The user must send an AID TLV in the function, as the first TLV in the function.
Format:	BYTE WINAPI CC_GetConfigurableAID(BYTE * pInputTLVData, UINT32 & nInputTLVDataLen, BYTE * pOutputTLVData, UINT32 & nOutputTLVDataLen)
Parameter:	pInputTLVData — The Input TLV data address nInputTLVDataLen — The Input TLV data length pOutputTLVData — The Output TLV data address nOutputTLVDataLen — The Output TLV data length.
Return:	Protocol II Response

Example	BYTE arrSend[512] = {0};
	UINT32 nSendLen = 0;
	BYTE arrRecv[512] = {0};
	UINT32 nRecvLen = 0;
	arrSend[nSendLen++] = 0x9F;
	arrSend[nSendLen++] = 0x06;
	arrSend[nSendLen++] = 0x07;
	arrSend[nSendLen++] = 0xA0;
	arrSend[nSendLen++] = 0x00;
	arrSend[nSendLen++] = 0x00;
	arrSend[nSendLen++] = 0x00;
	arrSend[nSendLen++] = 0x04;
	arrSend[nSendLen++] = 0x10;
	arrSend[nSendLen++] = 0x10;

	CC_GetConfigurableAID(arrSend,nSendLen,arrRecv,nRecvLen);	
Function:	CC_GetAllGroup	
Description:	This function returns all Groups in the reader. This command may be used to verify all configured Groups in the reader.	
Format:	BYTE WINAPI CC_GetAllGroup(BYTE * pOutputTLVData, UINT32 & nOutputTLVDataLen);	
Parameter:	pOutputTLVData	The Output TLV data address
	nOutputTLVDataLen	The Output TLV data length
Return:	Protocol II Response	
Example	BYTE arrRecv[512] = {0}; UINT32 nRecvLen = 0; CC_GetAllGroup(arrRecv,nRecvLen);	
Function:	CC_GetAllAID	
Description:	Use this function to return all AIDs in the reader.This funtion may be used to verify configured AIDs or to determine what System AIDs are in the reader.	
Format:	BYTE WINAPI CC_GetAllAID(BYTE * pOutputTLVData, UINT32 & nOutputTLVDataLen)	
Parameter:	pOutputTLVData	The output TLV data address.
	nOutputTLVDataLen	The output TLV data length.
Return:	Protocol II Response	
Example	BYTE arrRecv[512] = {0}; UINT32 nRecvLen = 0; CC_GetAllAID(arrRecv,nRecvLen);	
Function:	CC_DeleteConfigurableAID	
Description:	Close port for RS232 or HID communication.	
Format:	BYTE WINAPI CC_DeleteConfigurableAID(BYTE * pTLVData, UINT32 nTLVDataLen)	
Parameter:	pTLVData	The Input TLV Data address.
	nTLVDataLen	The Input TLV Data lenght.
Return:	Protocol II Response	

Example	BYTE arrSend[512] = {0};
	UINT32 nSendLen = 0;
	arrSend[nSendLen++] = 0x9F;
	arrSend[nSendLen++] = 0x06;
	arrSend[nSendLen++] = 0x07;
	arrSend[nSendLen++] = 0xA0;
	arrSend[nSendLen++] = 0x00;
	arrSend[nSendLen++] = 0x00;
	arrSend[nSendLen++] = 0x00;	

	arrSend[nSendLen++] = 0x04;
	arrSend[nSendLen++] = 0x10;
	arrSend[nSendLen++] = 0x10;
	CC_DeleteConfigurableAID (arrSend, nSendLen);		

Function:	TC_ActiveTransaction		
Description:	Use this function when the is in "Poll on Demand" to begin an EMV or contactless MagStripe card transaction.		
Format:	BYTE WINAPI TC_ActiveTransaction(BYTE * pInputTLVData, UINT32 & nInputTLVDataLen, BYTE * pOutputTLVData, UINT32 & nOutputTLVDataLen);		
Parameter:	pInputTLVData	The Input TLV data address.	
	nInputTLVDataLen	The Input TLV data length.	
	pOutputTLVData	The Output TLV data address.	
	nOutputTLVDataLen	The Output TLV data length.	
Return:	Protocol II Response		
Example	BYTE arrSend[512] = {0}; UINT32 nSendLen = 0; BYTE arrRecv[512] = {0}; UINT32 nRecvLen = 0; arrSend[nSendLen++] = 0x0A; TC_ActiveTransaction(arrSend,nSendLen,arrRecv,nRecvLen);		
Function:	TC_GetTransaction		
Description:	Use this function when the reader is in "Auto Poll" mode.		
Format:	BYTE WINAPI TC_GetTransaction(BYTE * pOutputData, UINT32 & nOutputDataLen);		
Parameter:	pOutputData	Track 1 Length	If Track 1 is available, then this field gives the length of the Track 1 data that follows. If Track 1 is not available, then a Length of 00h is returned. Format: Binary
		Track 1 Data (MagStripe card)	Track 1 Data (if available). Format: ASCII (no null terminator)
		Track 2 Length	If Track 2 is available, then this field gives the length of the Track 2 data that follows. If Track 2 is not available, then a Length of 00h is returned.

			Format: Binary
		Track 2 Data (MagStripe card)	Track 2 Data (if available). Format: ASCII (no null terminator)
		DE055 (Clearing Record) Present	If a Clearing Record (DE 055) field is available, then this field is 01h. If there is no Clearing Record (DE 055) field, then this field is 00h.
		TLV DE 055 (Clearing Record) (see Clearing Record Format)	DE 055 data (if available) as a TLV data object encoded with Tag 'E1'. The DE 055 data is the same data as is included in the Activate Transaction Clearing Record. Refer to the Activate Transaction Clearing Record table. Tag: E1 Format: b1...126 variable.
		TLV Data	Refer to Activate Response TLVs.
	nOutputDataLen		
Return:	Protocol II Response		
Example	BYTE arrRecv[512] = {0}; UINT32 nRecvLen = 0; TC_GetTransaction(arrRecv, nRecvLen);		
Function:	TC_UpdateBalance		
Description:	Use this function when the reader has been put in "Poll On Demand" mode and after the reader sends an online request to the issuer.		
Format:	BYTE WINAPI TC_UpdateBalance(BYTE nStatuCode, BYTE * pInputTLVData, UINT32 & nInputTLVDataLen, BYTE * pOutputTLVData, UINT32 & nOutputTLVDataLen)		
Parameter:	nStatusCode	0x00	OK
		0x01	NOT OK
	pInputTLVData	Authorization Code as a TLV object Tag:E300 Format:b8	
	nInputTLVDataLen	EMV data element "Transaction Date" as a TLV	

		data object. Local date that the transaction was authorized. If this TLV is not provided, the transaction uses the reader's current date. Tag: 9A Format: n6 (YYMMDD) Note: The reader does not perform range checking on this value. The POS application should perform range checking on this value to ensure it is within acceptable limits.
	pOutputTLVData	EMV data element "Transaction Time" as a TLV data object. Local time that the transaction was authorized. If this TLV is not provided, the transaction uses the reader's current time. Tag: 9F21 Format: n6 (HHMMSS) Note: The reader does not perform range checking on this value. The POS application should perform range checking on this value to ensure it is within acceptable limits.
	nOutputTLVDataLen	The length of TLV Datas.
Return:	Protocol II Response	

Example:	BYTE arrInputTLVData[512] = {0};
	UINT nInputTLVDataLen = 0;
	BYTE arrOutputTLVData[1024] = {0};
	UINT nOutputTLVDataLen = 0;
	BYTE nStatusCode = 0x00;
	arrTLVData[nInputTLVDataLen++] = 0xE3;
	arrTLVData[nInputTLVDataLen++] = 0x00;
	arrTLVData[nInputTLVDataLen++] = 0x06;
	arrTLVData[nInputTLVDataLen++] = 0x31;
	arrTLVData[nInputTLVDataLen++] = 0x32;
	arrTLVData[nInputTLVDataLen++] = 0x33;
	arrTLVData[nInputTLVDataLen++] = 0x34;
	arrTLVData[nInputTLVDataLen++] = 0x35;
	arrTLVData[nInputTLVDataLen++] = 0x36;
	arrTLVData[nInputTLVDataLen++] = 0x9A;
	arrTLVData[nInputTLVDataLen++] = 0x03;
	arrTLVData[nInputTLVDataLen++] = 0x14;
	arrTLVData[nInputTLVDataLen++] = 0x07;
	arrTLVData[nInputTLVDataLen++] = 0x28;
	arrTLVData[nInputTLVDataLen++] = 0x9F;
	arrTLVData[nInputTLVDataLen++] = 0x21;
	arrTLVData[nInputTLVDataLen++] = 0x03;	

	arrTLVData[nInputTLVDataLen++] = 0x09;
	arrTLVData[nInputTLVDataLen++] = 0x56;
	arrTLVData[nInputTLVDataLen++] = 0x10;
	TC_UpdateBalance(nStatusCode, arrTLVData, nInputTLVDataLen,
	arrOutputTLVData, nOutputTLVDataLen);	

Function:	TC_CancelTransaction	
Description:	Use this function to stop reader/card communication after the Active Transaction command or Update Balance command has been sent to the reader.	
Format:	BYTE WINAPI TC_CancelTransaction()	
Parameter	None	
Return:	Protocol II Response	
Example	TC_CancelTransaction()	
Function:	MC_StopTransaction	
Description:	The stop transaction function is similar to the cancel command. However, the transaction will exit at whatever phase it was currently in.	
Format:	BYTE WINAPI MC_StopTransaction(BYTE * pOutputData, UINT32 & nOutputDataLen)	
Parameter	pOutputData	
	nOutputDataLen	
Return:	Protocol II Response	
Example	BYTE arrRecv[512] = {0}; UINT32 nRecvLen = 0; MC_StopTransaction(arrRecv, nRecvLen);	
Function:	MC_ResetTornTransactionLog	
Description:	The Reset Torn Transaction Log effectively erases the content of The torn transaction log and sets it back to an "empty" state.	
Format:	BYTE WINAPI MC_ResetTornTransactionLog()	
Parameter	None	
Return:	Protocol II Response	
Example	MC_ResetTornTransactionLog( )	
Function:	MC_ClearTornTransactionLog	
Description:	This function is used to remove Torn Transaction Log entries that have exceeded the allowed lifetime defined in tag DF811C.	
Format:	BYTE WINAPI MC_ClearTornTransactionLog(BYTE * pOutputTLVData, UINT32 & nOutputTLVDataLen)	
Parameter	pOutputTLVData	The Output TLV data address.

	nOutputTLVDataLen	The Output TLV data length.
Return:	Protocol II Response.	
Example	BYTE arrRecv[512] = {0}; UINT32 nRecvLen = 0; MC_ClearTornTransactionLog(arrRecv, nRecvLen);	
Function:	VT_SetCashTransactionRiskParam	
Description:	This command creates or modifies the TTQ and reader risk parameters Associated with VCPS 2.1.1 cash transactions.	
Format:	BYTE WINAPI VT_SetCashTransactionRiskParam(BYTE * pInputTLVData, UINT32 & nInputTLVDataLen)	
Parameter	pInputTLVData	Cash Transaction TLVs.
	nInputTLVDataLen	Cash Transaction TLVs length.
Return:	Protocol II Response	

Example	BYTE arrInputTLVData[512] = {0};
	UINT nInputTLVDataLen = 0;
	arrInputTLVData[nInputTLVDataLen++] = 0x9f;
	arrInputTLVData[nInputTLVDataLen++] = 0x66;
	arrInputTLVData[nInputTLVDataLen++] = 0x04;
	arrInputTLVData[nInputTLVDataLen++] = 0xFF;
	arrInputTLVData[nInputTLVDataLen++] = 0xFF;
	arrInputTLVData[nInputTLVDataLen++] = 0xFF;
	arrInputTLVData[nInputTLVDataLen++] = 0xFF;
	VT_SetCashTransactionRiskParam(arrInputTLVData,
	nInputTLVDataLen);	

Function:	VT_GetCashTransactionRiskParam	
Description:	This command returns the TTQ and reader risk parameters that will be used for cash transactions, if enabled.	
Format:	BYTE WINAPI VT_GetCashTransactionRiskParam(BYTE * pOutputTLVData, UINT32 & nOutputTLVDataLen)	
Parameter	pOutputTLVData	The Output TLV Data address.
	nOutputTLVDataLen	The Output TLV Data length.
Return:	Protocol II Response	
Example	BYTE arrRecv[512] = {0}; UINT32 nRecvLen = 0; VT_GetCashTransactionRiskParam(arrRecv, nRecvLen);	
Function:	VT_SetCashbackTransactionRiskParam	
Description:	This function creates or modifies the TTQ and reader risk parameters associated with a VCPS 2.1.1 cashback transaction.	
Format:	BYTE WINAPI VT_SetCashbackTransactionRiskParam(BYTE * pInputTLVData, UINT32 & nInputTLVDataLen);	

Parameter	pInputTLVData	The Input TLV Data address.
	nInputTLVDataLen	The Input TLV Data length.
Return:	Protocol II Response	

Example	BYTE arrInputTLVData[512] = {0};
	UINT nInputTLVDataLen = 0;
	arrInputTLVData[nInputTLVDataLen++] = 0x9F;
	arrInputTLVData[nInputTLVDataLen++] = 0x66;
	arrInputTLVData[nInputTLVDataLen++] = 0x04;
	arrInputTLVData[nInputTLVDataLen++] = 0xFF;
	arrInputTLVData[nInputTLVDataLen++] = 0xFF;
	arrInputTLVData[nInputTLVDataLen++] = 0xFF;
	arrInputTLVData[nInputTLVDataLen++] = 0xFF;
	VT_SetCashbackTransactionRiskParam(arrInputTLVData,
	nInputTLVDataLen);	

Function:	VT_GetCashbackTransactionRiskParam	
Description:	This command returns the TTQ and reader risk parameters for all cashback transactions.	
Format:	BYTE WINAPI VT_GetCashbackTransactionRiskParam(BYTE * pOutputTLVData, UINT32 & nOutputTLVDataLen)	
Parameter	pOutputTLVData	The Output TLV Data address.
	nOutputTLVDataLen	The Output TLV data length.
Return:	Protocol II Response.	
Example	BYTE arrRecv[512] = {0}; UINT32 nRecvLen = 0; VT_GetCashbackTransactionRiskParam(arrRecv,nRecvLen);	
Function:	VT_SetDRLReaderRiskParam	
Description:	This command creates or modifies the Application Program ID and reader risk parameters for four Dynamic Reader Limit sets.	
Format:	BYTE WINAPI VT_SetDRLReaderRiskParam(BYTE * pInputTLVData, UINT32 & nInputTLVDataLen)	
Parameter	pInputTLVData	The buffer for the input TLV data.
	nInputTLVDataLen	The length of the input TLV data.
Return:	Protocol II Response.	

Example	BYTE arrTLVData[512] = {0};
	UINT nTLVDataLen = 0;
	arrTLVData[nTLVDataLen++] = 0x01;
	arrTLVData[nTLVDataLen++] = 0x9F;
	arrTLVData[nTLVDataLen++] = 0x5A;
	arrTLVData[nTLVDataLen++] = 0x00;
	arrTLVData[nTLVDataLen++] = 0xFF;
	arrTLVData[nTLVDataLen++] = 0xF4;
	arrTLVData[nTLVDataLen++] = 0x03;
	arrTLVData[nTLVDataLen++] = 0x00;
	arrTLVData[nTLVDataLen++] = 0x00;
	arrTLVData[nTLVDataLen++] = 0x00;
	arrTLVData[nTLVDataLen++] = 0xFF;
	arrTLVData[nTLVDataLen++] = 0xF1;
	arrTLVData[nTLVDataLen++] = 0x06;
	arrTLVData[nTLVDataLen++] = 0x00;
	arrTLVData[nTLVDataLen++] = 0x00;
	arrTLVData[nTLVDataLen++] = 0x00;
	arrTLVData[nTLVDataLen++] = 0x00;
	arrTLVData[nTLVDataLen++] = 0x00;
	arrTLVData[nTLVDataLen++] = 0x00;
	arrTLVData[nTLVDataLen++] = 0x9F;
	arrTLVData[nTLVDataLen++] = 0x1B;
	arrTLVData[nTLVDataLen++] = 0x04;
	arrTLVData[nTLVDataLen++] = 0x00;
	arrTLVData[nTLVDataLen++] = 0x00;
	arrTLVData[nTLVDataLen++] = 0x00;
	arrTLVData[nTLVDataLen++] = 0x00;
	arrTLVData[nTLVDataLen++] = 0xFF;
	arrTLVData[nTLVDataLen++] = 0xF5;
	arrTLVData[nTLVDataLen++] = 0x06;
	arrTLVData[nTLVDataLen++] = 0x00;
	arrTLVData[nTLVDataLen++] = 0x00;
	arrTLVData[nTLVDataLen++] = 0x00;
	arrTLVData[nTLVDataLen++] = 0x00;
	arrTLVData[nTLVDataLen++] = 0x00;
	arrTLVData[nTLVDataLen++] = 0x00;
	VT_SetDRLReaderRiskParam(arrTLVData, nTLVDataLen);

Function:	VT_GetDRLReaderRiskParam
Description:	This command returns the Index, Application Program ID, and reader risk parameters for the DRL settings.
Format:	BYTE WINAPI VT_GetDRLReaderRiskParam(

	UINT32 nDRLIndex, BYTE * pOutputTLVData, UINT32 & nOutputTLVDataLen)	
Parameter	nDRLIndex	The index of DRL
	pOutputTLVData	The Output TLV Data address.
	nOutputTLVDataLen	The output TLV data length.
Return:	Protocol II Response	
Example	UINT nIndex = 0x01; BYTE arrTLVData[512] = {0}; UINT nTLVDataLen = 0; VT_GetDRLReaderRiskParam(nIndex, arrTLVData, nTLVDataLen);	
Function:	PK_GetCAPublicKey	
Description:	This function retrieve all of the information related to a speciafic key. It includes the key hash, the algorithms, and so forth.	
Format:	BYTE WINAPI PK_GetCAPublicKey(BYTE arrRID[5], BYTE nKeyIndex, BYTE * pOutputData, UINT32 & nOutputDataLen)	
Parameter	arrRID	The RID
	nKeyIndex	The index of Key
	pOutputData	The Output TLV data.
	nOutputDataLen	The length of output TLV data.
Return:	Protocol II Response.	
Example	BYTE arrRIDData[512] = {0}; UINT32 nRIDDataLen = 0; PK_GetAllCAPublicRID(arrRIDData, nRIDDataLen); BYTE arrRIDIndex[512] = {0}; UINT32 nRIDIndexLen = 0; BYTE arrSend[5] = {0xA0,0x00,0x00,0x00,0x04}; PK_GetAllCAPublicKeyID(arrSend, arrRIDIndex, nRIDIndexLen); BYTE arrRID[5] = {0xA0,0x00,0x00,0x00,0x40}; BYTE arrRecv[512] = {0}; UINT32 nRecvLen = 0; PK_GetCAPublicKey(arrRID,0xf1,arrRecv,nRecvLen);	
Function:	PK_SetCAPublicKey	
Description:	This function adds a new key to the SAM in the reader.	
Format:	BYTE WINAPI PK_SetCAPublicKey(BYTE arrRID[5], BYTE nKeyIndex, BYTE * pKeyData,	

	UINT32 & nKeyDataLen)	
Parameter	arrRID	The RID
	nKeyIndex	The Index of Key
	pKeyData	The address of Key Data.
	nKeyDataLen	The length of key data.
Return:	Protocol II Response	
Example	BYTE arrRID[6] = { 0xA0,0x00,0x00,0x00,0x04,0x00}; BYTE nKeyIndex = 0x01; BYTE arrKeyData[] = {0x01,0xEC,0x0A ,0x59,0xD3,0x5D,0x19, 0xF0,0x31,0xE9,0xE8,0xCB,0xEC,0x56,0xDB,0x80,0xE2,0x2B, 0x1D,0xE1,0x30,0x00,0x00,0x00,0x03,0x00,0xA0,0x9C,0x6B,0xE5 ,0xAD,0xB1,0x0B,0x4B,0xE3,0xDC,0xE2,0x09,0x9B,0x4B,0x21, 0x06,0x72,0xB8,0x96,0x56,0xEB,0xA0,0x91,0x20,0x4F,0x61 ,0x3E,0xCC,0x62,0x3B,0xED,0xC9,0xC6,0xD7,0x7B,0x66,0x0E ,0x8B,0xAE,0xEA,0x7F,0x7C,0xE3,0x0F,0x1B,0x15,0x38,0x79,0xA4 ,0xE3,0x64,0x59,0x34,0x3D,0x1F,0xE4,0x7A,0xCD,0xBD,0x41,0xFC ,0xD7,0x10,0x03,0x0C,0x2B,0xA1,0xD9,0x46,0x15,0x97,0x98,0x2C ,0x6E,0x1B,0xDD,0x08,0x55,0x4B,0x72,0x6F,0x5E,0xFF,0x79,0x13 ,0xCE,0x59,0xE7,0x9E,0x35,0x72,0x95,0xC3,0x21,0xE2,0x6D,0x0B ,0x8B,0xE2,0x70,0xA9,0x44,0x23,0x45,0xC7,0x53,0xE2,0xAA,0x2A ,0xCF,0xC9,0xD3,0x08,0x50,0x60,0x2F,0xE6,0xCA,0xC0,0x0C,0x6D ,0xDF,0x6B,0x8D,0x9D,0x9B,0x48,0x79,0xB2,0x82,0x6B,0x04,0x2A ,0x07,0xF0,0xE5,0xAE,0x52,0x6A,0x3D,0x3C,0x4D,0x22,0xC7,0x2B ,0x9E,0xAA,0x52,0xEE,0xD8,0x89,0x38,0x66,0xF8,0x66,0x38,0x7A ,0xC0,0x5A,0x13,0x99}; UINT nKeyDataLen = 187; PK_SetCAPublicKey(arrRID, nKeyIndex,arrKeyData, nKeyDataLen);	
Function:	PK_GetAllCAPublicRID	
Description:	This function tells the reader to retrieve a list of all the RIDs form the SAM.	
Format:	BYTE WINAPI PK_GetAllCAPublicRID(BYTE * pRIDData, UINT32 & nRIDDataLen)	
Parameter	pRIDData	
	nRIDDataLen	
Return:	Protocol II Response	
Example	BYTE arrRIDData[512] = {0}; UINT32 nRIDDataLen = 0; PK_GetAllCAPublicRID(arrRIDData, nRIDDataLen);	
Function:	PK_GetAllCAPublicKeyID	
Description:	This function retrieve a list of key indices that are installed for this RID.	
Format:	BYTE WINAPI PK_GetAllCAPublicKeyID(BYTE arrRID[5], BYTE * pRIDIndex,	

	UINT32 & nRIDIndexLen)	
Parameter	BYTE arrRIDIndex[512] = {0}; UINT32 nRIDIndexLen = 0; BYTE arrSend[5] = {0xA0,0x00,0x00,0x00,0x04}; PK_GetAllCAPublicKeyID(arrSend, arrRIDIndex, nRIDIndexLen);	
Return:	Protocol II Response.	
Example	BYTE arrRIDData[512] = {0}; UINT32 nRIDDataLen = 0; PK_GetAllCAPublicRID(arrRIDData, nRIDDataLen); BYTE arrRIDIndex[512] = {0}; UINT32 nRIDIndexLen = 0; BYTE arrSend[5] = {0xA0,0x00,0x00,0x00,0x04}; PK_GetAllCAPublicKeyID(arrSend, arrRIDIndex, nRIDIndexLen);	
Function:	PK_GetCAPublicKeyHash	
Description:	This function returns only the "Checksum" portion of the key.	
Format:	BYTE WINAPI PK_GetCAPublicKeyHash(BYTE arrRID[5], BYTE nKey, BYTE * pKeyHash, UINT32 & nKeyHashLen)	
Parameter	arrRID	The RID
	nKey	The Key Index
	pKeyHash	The address of Key Hash
	nKeyHanshLen	The length of Key Hash
Return:	Protocol II Response.	
Example	BYTE arrRID[5] = {0x9F,0x06,0x06,0xA0,0x00}; BYTE nKey = 0x03; BYTE arrKeyHash[1024] = {0}; UINT32 nKeyHashLen = 0; PK_GetCAPublicKeyHash(arrRID, nKey, arrKeyHash, nKeyHashLen);	
Function:	PK_DeleteCAPublicKey	
Description:	This function allows the POS to delete a specific key.	
Format:	BYTE WINAPI PK_DeleteCAPublicKey(BYTE arrRID[5], BYTE nKeyIndex);	
Parameter	arrRID	The RID
	nKeyIndex	The Index of Key.
Return:	Protocol II Response	
Example	BYTE arrRID[5] = { 0xA0,0x00,0x00,0x00,0x04}; BYTE nKeyIndex = 0x00; PK_DeleteCAPublicKey(arrRID, nKeyIndex);	

Function:	PK_DeleteAllCAPublicKey	
Description:	This function deletes all of the CA public Keys.	
Format:	BYTE WINAPI PK_DeleteAllCAPublicKey();	
Parameter	sNumber	The buffer for Serial Number
	Length	The length of Serial Number
Return:	Protocol II Response.	
Example	PK_DeleteAllCAPublicKey( )	
Function:	MV_GetProcessorType	
Description:	This function returns a processor type TLV.	
Format:	BYTE WINAPI MV_GetProcessorType(BYTE * pOutputTLVData, UINT32 & nOutputTLVDataLen)	
Parameter	pOutputTLVData	The Get Processor Type sub-command returns a TLV string as follows: **Tag:** 0xDF61 **Length:** 0x01 **Value:** a 1-byte field representing the processor type.
	nOutputTLVDataLen	
Return:	Protocol II Response	
Example	BYTE arrTLVData[512] = {0}; UINT32 nTLVDataLen = 0; MV_GetProcessorType(arrTLVData, nTLVDataLen);	
Function:	MV_GetHardwareInfo	
Description:	This function returns the hardware version information.	
Format:	BYTE WINAPI MV_GetHardwareInfo(BYTE * pOutputTLVData, UINT32 & nOutputTLVDataLen);	
Parameter	pOutputTLVData	The Output TLV Data address.
	nOutputTLVDataLen	The Output TLV Data Length.
Return:	Protocol II Response	
Example	BYTE arrTLVData[1024] = {0}; UINT nTLVDataLen = 0; MV_GetHardwareInfo(arrTLVData, nTLVDataLen);	
Function:	MV_GetModuleVersionInfo	
Description:	The format for module version information returned is “human readable”, consisting of fields that are separated by commas, and lines separated by carriage return and line feed characters: <module type>,<module name and spec. version>,[<implementation version>],<CRLF>	
Format:	BYTE WINAPI MV_GetModuleVersionInfo(BYTE * pOutputTLVData, UINT32 & nOutputTLVDataLen)	
Parameter	pOutputTLVData	The Output TLV Data address.

	nOutputTLVDataLen	The Output TLV Data Length.
Return:	Protocol II Response.	
Example	BYTE arrTLVData[512] = {0}; nTLVDataLen = 0; MV_GetModuleVersionInfo(arrTLVData, nTLVDataLen);	
Function:	EMV_GetRevocationLogStatus	
Description:	This function returns information about the EMV revocation log.	
Format:	BYTE WINAPI EMV_GetRevocationLogStatus(BYTE * pOutputTLVData, UINT32 & nOutputTLVDataLen)	
Parameter	pOutputTLVData	The output TLV data address.
	nOutputTLVDataLen	The length of Output TLV data.
Return:	Protocol II Response	
Example	BYTE arrTLVData[1024] = {0}; UINT nTLVDataLen = 0; EMV_GetRevocationLogStatus(arrTLVData, nTLVDataLen);	
Function:	EMV_AddEntryToRevocation	
Description:	This function adds a new entry to revocation list.	
Format:	BYTE WINAPI EMV_AddEntryToRevocation(BYTE arrRID[5], BYTE nKeyIndex, BYTE arrInputSerialNum[3])	
Parameter:	arrRID	RID (packed hex format)
	nKeyIndex	Key Index (packed hex format)
	arrInputSerialNum	Certificate serial number (packed hex format)
Return:	Protocol II Response	

Example:	BYTE arrRID[5] = {
	0xA0,0x00,0x00,0x00,0x04
	};
	BYTE nKeyIndex = 0xF9;
	BYTE arrSerialNum[3] = {
	0x00,0x10,0x11
	};
	EMV_AddEntryToRevocation(arrRID, nKeyIndex, arrSerialNum);	

Function:	EMV_GetRevocationList	
Description:	This function retrieves a sequence of consecutive records from the EMV revocation list.	
Format:	BYTE WINAPI EMV_GetRevocationList(WORD nSize, WORD nStartRecord, BYTE * pOutputData, UINT & nOutputDataLen)	
Parameter:	nSize	maximum number of bytes to be retrieved

	nStartRecord	Starting record
	pOutputData	The address of output data.
	nOutputDataLen	The length of output data.
Return:	Protocol II Response	
Example:	EMV_GetRevocationList( );	
Function:	EMV_DeleteRevocationListEntry	
Description:	This command deletes all entries that match a key index and RID from the EMV revocation list.	
Format:	BYTE WINAPI EMV_DeleteRevocationListEntry(BYTE arrRID[5], BYTE nKeyIndex)	
Parameter:	arrRID	RID (packed hex format)
	nKeyIndex	Key Index (packed hex format)
Return:	Protocol II Response	
Example:	BYTE arrRID = {0xA0, 0x00,0x00,0x00,0x04 }; BYTE nKeyIndex = 0xF8; EMV_DeleteRevocationListEntry(arrRID, nKeyIndex);	
Function:	EMV_DeleteAllRevocationListEntry	
Description:	This command deletes all entries from the EMV revocation list.	
Format:	EMV_DeleteAllRevocationListEntry( )	
Parameter:	None	
Return:	Protocol II Response	
Example:	EMV_DeleteAllRevocationListEntry( );	
Function:	EMV_DeleteOneRevocationListEntry	
Description:	This command deletes a specific entry from the EMV revocation list. Unlike the commands described previously, this command deletes the specific entry that matches the RID, the key index, and the certificate serial number.	
Format:	EMV_DeleteOneRevocationListEntry(BYTE arrRID[5], BYTE nKeyIndex, BYTE arrSerialNumber[3])	
Parameter:	arrRID	RID (packed hex format)
	nKeyIndex	Key Index (packed hex format)
	arrSerialNumber	Certificate Serial Number (packed hex format)
Return:	Protocol II Response	

Example:	BYTE arrRID [] = {
	0xA0,0x00,0x00,0x00,0x04
	};
	BYTE nKeyIndex = 0xF8;
	BYTE arrSerialNumber[3] = {
	0x00,0x10,0x01	

	}; EMV_DeleteOneRevocationListEntry (arrRID, nKeyIndex, arrSerialNumber);		
Function:	PT_ModeStartStop		
Description:	The Pass-Through Mode Start/Stop command is used to enter and exit Pass-Through Mode.		
Format:	BYTE WINAPI PT_ModeStartStop(BYTE nMode)		
Parameter:	nMode	0 = Stop Pass-Through 1 = Start Pass-Through	
Return:	Protocol II Response.		
Example:	PT_ModeStartStop(0x00);		
Function:	PT_GetPCDPICCParam		
Description:	This command allows the terminal to retrieve PCD and PICC related parameters from the device reader.		
Format:	BYTE WINAPI PT_GetPCDPICCParam(BYTE * pOutputData, UINT32 & nOutputDataLen)		
Parameter:	pOutputData	Reader Buffer Size	Reader RF Buffer Size stored as a big-endian number.
		Max PICC Frame Size	Maximum PICC Frame Size stored as a big-endian number.
		CID	CID
		Block	Block Number
		CID Supported	CID Supported
		FWT	Frame Waiting Time in ETUs. It is stored as a big-endian number.
		D-FWT	Delta FWT in ETUs. It is stored as a big-endian number.
	nOutputDataLen		
Return	Protocol II Response		
Example:	BYTE arrOutputData[512] = {0}; UINT32 nOutputDataLen = 0; PT_GetPCDPICCParam(arrOutputData, nOutputDataLen);		
Function:	PT_PollForToken		
Description:	Once Pass-Through Mode is started, ViVOpay will not poll for any cards until the “Poll for Token” command is received. This command tells ViVOpay to start polling for a Type A or Type B PICC until a PICC is detected or a timeout occurs.		
Format:	BYTE WINAPI PT_PollForToken(BYTE nTimeSecs,		

	BYTE nTimeMilSecs, BYTE nCardType, BYTE * pSerialNumber, UINT32 & nSerialNumberLen)	
Parameter:	nTimeSecs	Time in Seconds Timeout1 cannot be zero seconds if Timeout2 is Zero.
	nTimeMilSecs	Multiplier for Time in multiples of 10 milliseconds.
	nCardType	Type of Card Found (or No Card Found). 00h None (Card Not Detected or Could not Activate) 01h ISO 14443 Type A (Supports ISO 14443-4 Protocol) 02h ISO 14443 Type B (Supports ISO 14443-4 Protocol) 03h Mifare Type A (Standard) 04h Mifare Type A (Ultralight) 05h ISO 14443 Type A (Does not support ISO 14443-4 Protocol) 06h ISO 14443 Type B (Does not support ISO 14443-4 Protocol) 07h ISO 14443 Type A and Mifare (NFC phone)
	pSerialNumber	Serial Number (or the UID) of the PICC. Length depends on the Card Detected. If no card was detected, then a Serial Number is not returned.
	nSerialNumberLen	
Return:	Protocol II Response	
Example:	BYTE nTimeSec = 0x02; BYTE nTimeMilSec = 0x00; BYTE nCardType = 0; BYTE arrSerialCard[128] = {0}; UINT32 nSerialCardLen = 0 ; PT_PollForToken(nTimeSec, nTimeMilSec, nCardType, arrSerialCard, nSerialCardLen);	
Function:	PT_EnhancePollForToken	
Description:	This command tells ViVOpay to start polling for a Type A or Type B PICC until a PICC is detected or a timeout occurs.	
Format:	BYTE WINAPI PT_EnhancePollForToken(BYTE nTimeSecs, BYTE nTimeMilSecs, BYTE arrTransactType[2], BYTE nCardType, BYTE * pSerialNumber, UINT32 & nSerialNumberLen)	
Parameter:	nTimeSecs	Time in Seconds Timeout1 cannot be zero seconds if Timeout2 is Zero.
	nTimeMilSecs	Multiplier for Time in multiples of 10 milliseconds.

	arrTransactType	Initiate a transaction based upon the following masks (more than 1 can be active):
	nCardType	Type of Card Found (or No Card Found). 00h None (Card not detected or could not activate) 01h ISO 14443 Type A (Supports ISO 14443-4 protocol) 02h ISO 14443 Type B (Supports ISO 14443-4 protocol) 03h Mifare Type A (Standard) 04h Mifare Type A (Ultralight) 05h ISO 14443 Type A (Does not support ISO 14443-4 protocol) 06h ISO 14443 Type B (Does not support ISO 14443-4 protocol) 07h ISO 14443 Type A and Mifare (NFC phone)
	pSerialNumber	Serial number (or the UID) of the PICC. Length depends on the card detected. If no card was detected, then a serial number is not returned.
	nSerialNumberLen	
Return:	Protocol II Response	
Example:	BYTE nTimeSecs = 0x00; BYTE nTimeMilSecs = 0x14; BYTE arrTransactType = 0x00; BYTE nCardType = 0x01; BYTE arrSerialNumber[512] = {0}; UINT nSerialNumberLen = 0; PT_EnhancePollForToken(nTimeSecs, nTimeMilSecs, arrTransactType, nCardType, nSerialNumber, nSerialNumberLen);	
Function:	PT_GetATR	
Description:	This pass-through command can be used to get the ATR received by the reader from the SAM when a Level 1 session was established. This command applies to the SAM interfaces (SAM1/SAM2).	
Format:	BYTE WINAPI PT_GetATR(BYTE nInterface,BYTE * pATR, UINT32 & nATRLen)	
Parameter:	pATR	Allowed interfaces for which to get the ATR. 00h = retrieve last ATR received from PICC 21h = SAM 1 22h = SAM 2
	nATRLen	The Length of ATR.
Return	Protocol II Response	
Example:	BYTE arrATR[1024] = {0}; UINT nATRLen = 0; PT_GetATR(0x21, arrATR, nATRLen);	
Function:	PT_AntennaControl	

Description:	This command turns the RF Antenna ON or OFF.	
Format:	BYTE WINAPI PT_AntennaControl(BYTE nMode);	
Parameter:	nMode	0 = Disable RF Antenna 1 = Enable RF Antenna
Return:	Protocol II Response	
Example:	PT_AntennaControl(0x00)	
Function:	PT_LEDControl	
Description:	This command switches the specified ViVOpay LEDs off or on only when ViVOpay is in Pass-Through Mode.	
Format:	BYTE WINAPI PT_LEDControl(BYTE nLEDNum, BYTE nLEDStatus)	
Parameter:	nLEDNum	00h: LED 0 (Power LED) 01h: LED 1 02h: LED 2 03h: LED 3 FFh: All 4 LEDs Where the LEDs are numbered 0, 1, 2, 3 counting from the left. **Note:** If you are using pass-through mode to control the Power LED (LED 0), it is your responsibility to make sure that it behaves correctly.
	nLEDStatus	00h: LED Off 01h: LED On
Return:	Protocol II Response	
Example:	PT_LEDControl(0xFF,0x01);	
Function:	PT_BuzzerControl	
Description:	This command can be used to sound the ViVOpay Buzzer.	
Format:	BYTE WINAPI PT_BuzzerControl(BYTE nBuzzerMode, BYTE nBuzzerParam)	
Parameter:	nBuzzerMode	Sub-Command = 01h N Short Beeps = 02h Single Long Beep of Specified Duration
	nBuzzerParam	If Sub-Command is Short Beeps … Num Beeps = 01h One Short Beep = 02h Two Short Beeps = 03h Three Short Beeps = 04h Four Short Beeps If Sub-Command is Long Beep … Duration = 00h 200 ms = 01h 400 ms = 02h 600 ms = 03h 800 ms
Return:	Protocol II Response	
Example:	PT_BuzzerControl(0x01, 0x01);	
Function:	PT_ExchangeContactlessData	

Description:	This command allows the terminal to send, via device, application-level APDUs to a PICC that supports ISO 14443-4 Protocol. The PICC response is sent back by ViVOpay to the Terminal.	
Format:	BYTE WINAPI PT_ExchangeContactlessData(BYTE * pData, UINT32 nDataLen, BYTE * pOutputAPDU, UINT32 & nOutputAPUD);	
Parameter:	pData	APDU data that sent to the PICC.
	nDataLen	APDU data length.
	pOutputAPDU	The Output APDU data address.
	nOutputAPDU	The Output APDU data length.
Return:	Response	
Example:	BYTE arrAPDU[] = {0x00,0x0A4,0x04,0x00,0x0E,0x32,0x50,0x41,0x59 0x2E,0x53,0x59,0x53,0x2E,0x44,0x44,0x46,0x30,0x31}; BYTE arrOutputAPDU[1024] = {0}; UINT nOutputAPDULen = 0; PT_ExchangeContactlessData(arrAPDU, sizeof(arrAPDU), arrOutputAPDU, nOutputAPDULen);	
Function:	PT_ExchangeSingleCommand	
Description:	This command allows the terminal to send, via the ViVOpay reader, raw data to an ISO 14443 PICC that does not support ISO 14443-4 Protocol (such as Mifare Standard or Mifare Ultralight). The PICC response is sent back by the reader to the terminal.	
Format:	BYTE WINAPI PT_ExchangeSingleCommand(BYTE nPCDCmd, BYTE arrPCDTimeout[4], BYTE nPCDCmdFlag, BYTE nRedundancy, BYTE * pRawData, UINT32 nRawDataLen, BYTE * pOutputData, UINT32 & nOutputDataLen)	
Parameter:	nPCDCmd	This is the command that is sent to the PCD Reader IC on the ViVOpay board. It tells the PCD what to do with the data sent with the command. The PCD commands supported and their values are given in the “PCD Cmd” Table below
	arrPCDTimeout	This is the RF communication timeout in ETUs stored as a 4-byte big-endian number, where 1 ETU is 9.44 microseconds. The RF communication timeout guards the communication

		between the PCD reader IC and the PICC Card. The timeout is measured between the last bit sent to the PICC and the first bit received from the PICC.
	nPCDCmdFlag	These flags allow greater control over the way ViVOpay processes the command via the PCD Reader IC. Format of the PCD Command Flags byte is given in the "PCD Command Flags" Table below.
	nRedundancy	This value tells the PCD what data integrity checks to perform during communication with the PICC Card. The checks to perform at each stage are defined by the protocol (14443 Type A or B). The format of the PCD Command Flags byte is given in the "Channel Redundancy Register" Table below.
	pRawData	Raw data that is sent to the PICC or to the PCD.
	nRawDataLen	The length of Raw Data.
	pOutputData	The output data address.
	nOutputDataLen	The output data length.
Return:	Protocol II Response	

Example:	BYTE nPCDCmd = 0x19;
	arrPCDTimeout[] = {0x00,0x00,0x29,0x68};
	nPCDCmdFlag = 0x01;
	nRedundancy = 0x0F;
	BYTE arrRawData[] = {0x0F,0x0F,0x0F,0x0F,0x0F,0x0F,
	0x0F,0x0F,0x0F,0x0F,0x0F,0x0F};
	UINT nRawDataLen = 12;
	PT_ExchangeSingleCommand(nPCDCmd, arrPCDTimeout,
	nPCDCmdFlag, nRedundancy, arrRawData, nRawDataLen);	

Function:	PT_HighLevelHalt	
Description:	This command instructs the ViVOpay reader to send a HALT command to the card and can be used for any Type A or Type B card. This command can only be used once the reader has been put in Pass-Through mode and the "Poll for Token" command has indicated that a Card is present.	
Format:	BYTE WINAPI PT_HighLevelHalt(BYTE nCardType)	
Parameter:	nCardType	Card Type: 0x01 = Type A

		0x02 = Type B
Return:	Protocol II Response	
Example:	PT_HighLevelHalt(0x01)	
Function:	PT_EnhancePassThroughControl	
Description:	This command instructs the reader to carry out several tasks while in Pass-Through Mode. This command is ONLY enabled in Pass-Through Mode. If the reader is not in Pass-Through Mode, the reader ignores this command.	
Format:	BYTE WINAPI PT_EnhancePassThroughControl(BYTE nShotCmd, BYTE nLCDMsgIndex, BYTE nBeepIndicator, BYTE nLEDNumber, BYTE nLEDStatus, BYTE nFirTimeout, BYTE nSecTimeout)	
Parameter:	nShotCmd	This bitmask contains the following commands. If a bit is set (1), the command is issued by the reader in the proper order. If the command mask is cleared (0), the command is NOT executed. 0000 0001 : Activate Interface (mutually exclusive) 0000 0010 : Deactivate Interface (mutually exclusive) 0000 0100 : Issue Poll For Token 0000 1000 : Use Independent LED instead (mutually exclusive) 0001 0000 : Use Independent Buzzer instead (mutually exclusive) 0010 0000 : Use Specified Interface 0100 0000 : Reserved 1000 0000 : Reserved
	nLCDMsgIndex	00-07 is controlled by the reader and normally not set by this command 00: Idle Message (Welcome) 01: Present card (Please Present Card) 02: Time Out or Transaction Cancel (No Card) 03: Transaction between reader and card is in the middle (Processing…) 04: Transaction Pass (Thank You) 05: Transaction Fail (Fail) 06: Amount (Amount $ 0.00 Tap Card) 07: Balance or Offline Available funds (Balance $ 0.00) 08-0B is controlled by the terminal through this command 08: Insert or Swipe card (Use Chip & PIN)

		09: Try Again(Tap Again) 0A: Indicate the custom to present only one card (Present 1 card only) 0B: Indicate the custom to wait for authentication/authorization (Wait) 80 MASK – indicates the User has included a String1 character string to be displayed with the standard message. If 0x80 is present, then the message index (in the lower portion of the byte) is displayed and LCD String1 and LCD String2 are only used for the Amount and Balance messages. FE indicates that the user wants to use a custom message, which is contained in the Custom LCD Message field described below. This is mutually exclusive with the 80 MASK. If FE is present, then the Reserved field must be present and immediately followed by the Custom LCD Message string. FF indicates not to set the LCD message which allows terminal to set LED/Buzzer only. EXAMPLE 1: Index = 00h, reader displays standard "Welcome" EXAMPLE 2: Index = 86h, reader displays standard 06h "Amount" message but also displays String1 (in this case String1 = "$3.95"
	nBeepIndicator	00h: No beep 01h: Single beep 02h: Double beep 03h: Three short beeps 04h: Four short beeps 05h One long beep of 200 ms 06h One long beep of 400 ms 07h One long beep of 600 ms 08h One long beep of 800 ms
	nLEDNumber	00h: LED 0 (Power LED) 01h: LED 1 03h: LED 2 03h: LED 3 FFh: All LEDs Where the LEDs are numbered 0, 1, 2, 3 counting from the left. **Note:** If you are using past-through mode to control the Power LED (LED 0), it is your responsibility to make sure that it behaves correctly.
	nLEDStatus	00h: LED Off 01h: LED On

	nFirTimeout	Time in Seconds. Timeout1 cannot be zero seconds if Timeout2 is Zero.
	nSecTimeout	Multiplier for Time in multiples of 10 milliseconds.
Return:	Protocol II Response	
Example:	PT_EnhancePassThroughControl (0x19,0x00,0x01,0x00,0x00,0x00,0x00);	
Function:	HL_MifareAuthenticateBlock	
Description:	This command allows the terminal to instruct the ViVOpay reader to authenticate the Mifare Card sector containing the specified block of data.	
Format:	BYTE WINAPI HL_MifareAuthenticateBlock(BYTE nBlock, BYTE nKeyType,BYTE arrKey[6])	
Parameter:	nBlock	Block Number in the Mifare Card for which the relevant sector must be authenticated.
	nKeyType	Specifies which type of key to use for authentication. It can have the following values. 01h: Key A 02h: Key B
	arrKey	Value of the Key
Return:	Protocol II Response	
Example:	BYTE nBlock = 0x01; BYTE nKeyType = 0x01; BYTE arrKey[6] = {0xFF,0xFF,0xFF,0xFF,0xFF,0xFF}; HL_MifareAuthenticateBlock(nBlock, nKeyType, arrKey);	
Function:	HL_MifareReadBlock	
Description:	Use this command to instruct the ViVOpay reader to write data to one or more blocks on the Mifare Card.	
Format:	BYTE WINAPI HL_MifareReadBlock(BYTE nCardBlock, BYTE nStartBlock, BYTE * pOutputData, UINT32 & nOutputDataLen)	
Parameter	nCardBlock	Card Type: [Bit 7..4] This can only indicate the following cards Mifare Type A (Standard) Mifare Type A (Ultralight) The values for these card types are defined in the “Poll for Token” command (consider only the lower 4 bits). Block Count: [Bit 3..0] This is the number of blocks that are written. The Block Count cannot be greater than 15. This count does not include the skipped blocks if the card is a Mifare Standard card.
	nStartBlock	This is the card block number from which the reader starts writing.

	pOutputData	Data to write to the Card. The length of the data to be written to the card depends on the number of blocks to be written and the card type.
	nOutputDataLen	
Return:	Protocol II Response	
Example:	BYTE nCardBlock = 0x31; BYTE nStartBlock = 0x02; BYTE arrOutputData[1024] = {0}; UINT nOutputDataLen = 0; HL_MifareReadBlock(nCardBlock, nStartBlock, arrOutputData, nOutputDataLen);	
Function:	HL_MifareWriteBlock	
Description:	Use this command to instruct the ViVOpay reader to write data to one or more blocks on the Mifare Card. The terminal can instruct ViVOpay to write up to 15 blocks of data using this command. If more than one block is defined, then the reader automatically writes to the starting block and the blocks that follow.	
Format:	BYTE WINAPI HL_MifareWriteBlock(BYTE nCardBlock, BYTE nStartBlock, BYTE * pWriteData, UINT32 nWriteDataLen)	
Parameter	nCardBlock	Card Type: [Bit 7..4] This can only indicate the following cards Mifare Type A (Standard) Mifare Type A (Ultralight) The values for these card types are defined in the “Poll for Token” command (consider only the lower 4 bits). Block Count: [Bit 3..0] This is the number of blocks that are written. The Block Count cannot be greater than 15. This count does not include the skipped blocks if the card is a Mifare Standard card.
	nStartBlock	This is the card block number from which the reader starts writing.
	pWriteData	Data to write to the Card. The length of the data to be written to the card depends on the number of blocks to be written and the card type.
	nWriteDataLen	The Length of written Data.
Return:	Protocol II Response	
Example:	HL_MifareWriteBlock (0x34,0x00);	
Function:	HL_PurseValueBlock	
Description:	Use this command to instruct the ViVOpay reader to carry out Debit, Credit and Backup operations on value blocks in a Mifare card.	

Format:	BYTE WINAPI HL_PurseValueBlock(BYTE nMode, BYTE * pPurseFuncBlock, UINT32 nPurseFuncBlockLen);	
Parameter	nMode	7 6 5 4 3 2 1 0 1 inc Card Type Operation Count 0 dec Increment / Decrement Flag: [Bit 7] Set to 1 instructs reader to Add to (Credit) amount. Set to 0 instructs reader to Subtract from (Debit) amount. Card Type: [Bit 6..4] This can only indicate Mifare Type A (Standard) card (3, as defined in the "Poll for Token" command). Operation Count: [Bit 3..0] This is the number of operation command blocks contained within the rest of the Purse Function data area.
	pPurseFuncBlock	Series of any combination of supported Purse Function blocks (Debit/Credit, Backup). Refer to the description of each individual Command Frame below.
	nPurseFuncBlock	The Length of PurseFuncBlock.
Return:	Protocol II Response	
Example:	BYTE nMode = 0x31; BYTE arrPurseFuncBlock[] = {0x20,0x04,0xD0,0x07,0x00,0x00}; HL_PurseValueBlock(nMode, arrPurseFuncBlock, sizeof(arrPurseFuncBlock));	
Function:	TR_FlushTrackData	
Description:	This command allows the POS application to instruct device to flush any Track Data that was read from a card previously but has not been transferred to the POS yet. On receiving this command device clears any pending card data.	
Format:	BYTE WINAPI TR_FlushTrackData()	
Parameter	None	
Return:	Protocol I Response	
Example:	TR_FlushTrackData();	
Function:	TR_GetFullTrackData	
Description:	Use this command to return full track data from the ViVOpay reader.	
Format:	TR_GetFullTrackData(BYTE * pReadData, UINT32 & nReadDataLen)	

Parameter	pReadData	Byte 12 is used for Tracks or Error Code, depending on the value of the Status in Byte 11 (see Status Code Protocol 1). When Status is OK, Byte 12 is used to store Tracks. When Status is Failed, Byte 12 is used to store the Error Codes from device.
	nReadDataLen	Number of Data Bytes in the Data Frame to Follow. This does not include the Frame Tag, Frame Type and Checksum bytes.
Return:	Protocol I Response	
Example:	BYTE arrReadData[1024] = {0}; UINT nReadDataLen = 0; TR_GetFullTrackData(arrReadData, nReadDataLen);	
Function:	TR_GetFirmwareVersion	
Description:	Use this command to return the device reader's Firmware Version Number.	
Format:	BYTE WINAPI TR_GetFirmwareVersion(BYTE * pFirmVersion, UINT32 & nFirmVersionLen)	
Parameter	pFirmVersion	Device Version
	nFirmVersionLen	The Length of Device Version
Return:	Protocol I Response	
Example:	BYTE arrFirmwareVersion[512] = {0}; UINT nFirmwareVersionLen = 0; TR_GetFirmwareVersion(arrFirmwareVersion, nFirmwareVersionLen);	
Function:	TR_DeleteCaPublicKey	
Description:	Use this command to instruct the ViVOpay reader to delete a previously set CA Public Key from within secure storage in the Crypto Chip.	
Format:	BYTE WINAPI TR_DeleteCaPublicKey(BYTE arrRID[5], BYTE & nKeyIndex)	
Parameter	arrRID	Registered Identifier (5 Bytes)
	nKeyIndex	Key Index (1 Byte)
Return:	Protocol I Response	
Example:	BYTE arrRID[5] = {0xA0,0x00,0x00,0x00,0x40}; TR_DeleteCaPublicKey(arrRID, 0xF1);	
Function:	TR_DeleteAllCaPublicKey	
Description:	Use this command to instruct the ViVOpay reader to delete all previously set CA Public Keys from within secure storage in the Crypto Chip.	
Format:	BYTE WINAPI TR_DeleteAllCaPublicKey( );	
Parameter	None	
Return:	Protocol I Response	

Example:	TR_DeleteAllCaPublicKey();	
Function:	TR_SetRFErrorReport	
Description:	This command allows the POS application to Enable/Disable RF Error Code Reporting for the Get Full Track Data command.	
Format:	BYTE WINAPI TR_SetRFErrorReport(BYTE nOperationCode)	
Parameter	nOperationCode	Operation Code: 00h: Disable RF Error Code Reporting 01h: Enable RF Error Code Reporting 02h or others: No change
Return:	Protocol I Response	
Example:	TR_SetRFErrorReport(0x00);	
Function:	TR_SetTime	
Description:	Use this command to instruct the ViVOpay reader to set a specific time in the Real Time Clock.	
Format:	BYTE WINAPI TR_SetTime(BYTE nHour, BYTE nMinute)	
Parameter	nHour	Hour (2-digit, BCD, Range 00-23)
	nMinute	Minutes (2-digit, BCD, Range 00-59)
Return:	Protocol I Response	
Example:	TR_SetTime(0x18, 0x00);	
Function:	TR_GetTime	
Description:	Use this command to instruct the ViVOpay reader to return the current time from the Real Time Clock.	
Format:	BYTE WINAPI TR_GetTime(BYTE & nHour, BYTE & nMinute);	
Parameter	nHour	Hour (2-digit, BCD, Range 00-23)
	nMinute	Minutes (2-digit, BCD, Range 00-59)
Return:	Protocol I Response.	
Example:	BYTE nHour = 0; BYTE nMinute = 0; TR_GetTime(nHour, nMinute);	
Function:	TR_SetDate	
Description:	Use this command to instruct the ViVOpay reader to set a specific Date in the Real Time Clock.	
Format:	BYTE WINAPI TR_SetDate(BYTE nMYear, BYTE nLYear, BYTE nMonth, BYTE nDay)	
Parameter	nMYear	Year (Higher Century Byte) (2-Digit, BCD, Range 00-99)
	nLYear	Year (Lower Byte) (2-Digit, BCD, Range 00-99)
	nMonth	(2-Digit, BCD, Range 01-12)

	nDay	(2-Digit, BCD, Range 01-31)
Return:	Protocol I Response	
Example:	TR_SetDate(0x20,0x04,0x12,0x25);	
Function:	TR_GetDate	
Description:	This command returns the reader's the current Date from the Real Time Clock.	
Format:	BYTE WINAPI TR_GetDate(BYTE & nMYear, BYTE & nLYear, BYTE & nMonth, BYTE & nDay);	
Parameter	nMYear	Year (Higher Century Byte) (2-Digit, BCD, Range 00-99)
	nLYear	Year (Lower Byte) (2-Digit, BCD, Range 00-99)
	nMonth	(2-Digit, BCD, Range 01-12)
	nDay	(2-Digit, BCD, Range 01-31)
Return:	Protocol I Response	
Example:	BYTE nMYear; BYTE nLYear; BYTE nMonth; BYTE nDay; TR_GetDate(nMYear, nLYear, nMonth, nDay);	

Protocol II Response

Status Code	Status
00h	OK
01h	Incorrect Header Tag
02h	Unknown Command
03h	Unknown Sub-Command
04h	CRC Error in Frame
05h	Incorrect Parameter
06h	Parameter Not Supported
07h	Mal-formatted Data
08h	Timeout
0Ah 0Bh	Failed / NACK Command not Allowed
0Ch	Sub-Command not Allowed
0Dh	Buffer Overflow (Data Length too large for reader buffer)
0Eh	User Interface Event
11h	Communication type not supported, VT-1, burst, etc.
12h	Secure interface is not functional or is in an intermediate state.
13h	Data field is not mod 8
14h	Pad 0x80 not found where expected

15h	Specified key type is invalid
16h	Could not retrieve key from the SAM (InitSecureComm)
17h	Hash code problem
18h	Could not store the key into the SAM (InstallKey)
19h	Frame is too large
1Ah	Unit powered up in authentication state but POS must resend the InitSecureComm command
1Bh	The EEPROM may not be initialized because SecCommInterface does not make sense
1CH	Problem encoding APDU
20h	Unsupported Index (ILM) SAM Transceiver error – problem communicating with the SAM (Key Mgr)
21h	Unexpected Sequence Counter in multiple frames for single bitmap (ILM) Length error in data returned from the SAM (Key Mgr)
22h	Improper bitmap (ILM)
23h	Request Online Authorization
24h	ViVOCard3 raw data read successful
25h	Message index not available (ILM) ViVOcomm activate transaction card type (ViVOcomm)
26h	Version Information Mismatch (ILM)
27h	Not sending commands in correct index message index (ILM)
28h	Time out or next expected message not received (ILM)
29h	ILM languages not available for viewing (ILM)
2Ah	Other language not supported (ILM)
41h – 4Fh	Module-specific errors for Key Manager
50h	Auto-Switch OK
51h	Auto-Switch failed
FFh	Happen a mistake in SDK running.

Protocol I Response

Status Code	Status
00h	OK
01h	Incorrect Frame Tag
02h	Incorrect Frame Type
03h	Unknown Frame Type
04h	Unknown Command
05h	Unknown Sub-Command
06h	CRC Error
07h	Failed
08h	Timeout
0Ah	Incorrect Parameter
0Bh	Command Not Supported
0Ch	Sub-Command Not Supported
0Dh	Parameter Not Supported / Status Abort Command
0Eh	Command not Allowed
0Fh	Sub-Command Not Allowed
FFh	Happen a mistake in running.