MAGTEK®
SECURITY FROM THE INSIDE

IPAD, DynaPro, DynaPro Go, and DynaPro Mini

PIN Encryption Devices

Programmer's Reference (C++)

September 2018

Manual Part Number:
D99875656-100

REGISTERED TO ISO 9001:2015

Table 0.1 – Revisions

Rev Number	Date	Notes
1.01	March 12, 2014	Initial Release
2.01	April 21,2014	Add Ethernet and Bluetooth low energy support.
3.01	July 18, 2014	Add referenced documents.
40	January 20, 2015	Add Ethernet and Bluetooth low energy support. Add referenced documents. Add appendix C for cryptography. Add helper functions for card data and pin data. Add function GetKeyInfo, GetAMKInfo and IsDeviceSRED. Add function applicable table. Update SendBitmap, bitmap size, and add reference document. Add appendix for Smart Card L1 Session. Add IPAD to supported device list. Update cardtype to include Contactless. Deprecate functions containing preposition “With” in the prototype name.
50	April 21, 2015	Update the function UpdateFirmware to load firmware from a data buffer.
60	March 1, 2016	Replace references of EMVAcqResponseCompleteEventHandler with OnEMVDataComplete
70	April 25, 2016	Update RequestGetEMVTagsEx and RequestSetEMVTagsEx.

80	March 6, 2017	Add support of DynaPro Go. Add note to getKSN function of its usage for token reversal.
90	August, 14, 2017	Add LoadClientCertificate.
100	September 7, 2018	Updated RequestSmartCard to increase reserved bytes size from 29 bytes to 45 bytes. Add RequestTipOrCashback, GetSelectedMenuItem, DeviceMessageTipOrCashback, and DeviceMessageSelectedMenuItem.

SOFTWARE LICENSE AGREEMENT

IMPORTANT: YOU SHOULD CAREFULLY READ ALL THE TERMS, CONDITIONS AND RESTRICTIONS OF THIS LICENSE AGREEMENT BEFORE INSTALLING THE SOFTWARE PACKAGE. YOUR INSTALLATION OF THE SOFTWARE PACKAGE PRESUMES YOUR ACCEPTANCE OF THE TERMS, CONDITIONS, AND RESTRICTIONS CONTAINED IN THIS AGREEMENT. IF YOU DO NOT AGREE WITH THESE TERMS, CONDITIONS, AND RESTRICTIONS, PROMPTLY RETURN THE SOFTWARE PACKAGE AND ASSOCIATED DOCUMENTATION TO THE ADDRESS ON THE FRONT PAGE OF THIS DOCUMENT, ATTENTION: CUSTOMER SUPPORT.

TERMS, CONDITIONS, AND RESTRICTIONS

MagTek, Incorporated (the "Licensor") owns and has the right to distribute the described software and documentation, collectively referred to as the "Software."

LICENSE: Licensor grants you (the "Licensee") the right to use the Software in conjunction with MagTek products. LICENSEE MAY NOT COPY, MODIFY, OR TRANSFER THE SOFTWARE IN WHOLE OR IN PART EXCEPT AS EXPRESSLY PROVIDED IN THIS AGREEMENT. Licensee may not decompile, disassemble, or in any other manner attempt to reverse engineer the Software. Licensee shall not tamper with, bypass, or alter any security features of the software or attempt to do so.

TRANSFER: Licensee may not transfer the Software or license to the Software to another party without the prior written authorization of the Licensor. If Licensee transfers the Software without authorization, all rights granted under this Agreement are automatically terminated.

COPYRIGHT: The Software is copyrighted. Licensee may not copy the Software except for archival purposes or to load for execution purposes. All other copies of the Software are in violation of this Agreement.

TERM: This Agreement is in effect as long as Licensee continues the use of the Software. The Licensor also reserves the right to terminate this Agreement if Licensee fails to comply with any of the terms, conditions, or restrictions contained herein. Should Licensor terminate this Agreement due to Licensee's failure to comply, Licensee agrees to return the Software to Licensor. Receipt of returned Software by the Licensor shall mark the termination.

LIMITED WARRANTY: Licensor warrants to the Licensee that the disk(s) or other media on which the Software is recorded are free from defects in material or workmanship under normal use.

THE SOFTWARE IS PROVIDED AS IS. LICENSOR MAKES NO OTHER WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

Because of the diversity of conditions and PC hardware under which the Software may be used, Licensor does not warrant that the Software will meet Licensee specifications or that the operation of the Software will be uninterrupted or free of errors.

IN NO EVENT WILL LICENSOR BE LIABLE FOR ANY DAMAGES, INCLUDING ANY LOST PROFITS, LOST SAVINGS, OR OTHER INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE, OR INABILITY TO USE, THE SOFTWARE. Licensee's sole remedy in the event of a defect in material or workmanship is expressly limited to replacement of the Software disk(s) if applicable.

GOVERNING LAW: If any provision of this Agreement is found to be unlawful, void, or unenforceable, that provision shall be removed from consideration under this Agreement and will not affect the enforceability of any of the remaining provisions. This Agreement shall be governed by the laws of the State of California and shall inure to the benefit of MagTek, Incorporated, its successors or assigns.

ACKNOWLEDGMENT: LICENSEE ACKNOWLEDGES THAT HE HAS READ THIS AGREEMENT, UNDERSTANDS ALL OF ITS TERMS, CONDITIONS, AND RESTRICTIONS, AND AGREES TO BE BOUND BY THEM. LICENSEE ALSO AGREES THAT THIS AGREEMENT SUPERSEDES ANY AND ALL VERBAL AND WRITTEN COMMUNICATIONS BETWEEN LICENSOR AND LICENSEE OR THEIR ASSIGNS RELATING TO THE SUBJECT MATTER OF THIS AGREEMENT.

QUESTIONS REGARDING THIS AGREEMENT SHOULD BE ADDRESSED IN WRITING TO MAGTEK, INCORPORATED, ATTENTION: CUSTOMER SUPPORT, AT THE ADDRESS LISTED IN THIS DOCUMENT, OR E-MAILED TO SUPPORT@MAGTEK.COM.

Table of Contents

1 Introduction

This document provides instructions for software developers who want to create software solutions that include an IPAD, DynaPro, or DynaPro Mini connected to a Windows-based host via USB. It is part of a larger library of documents designed to assist IPAD, DynaPro, DynaPro Go, and DynaPro Mini implementers, which includes the following documents available from MagTek:

- ***D99875586 DynaPro Installation and Operation Manual***
- ***D99875642 DynaPro Mini Installation and Operation Manual***
- ***D99875622 Dynapro Image Installation Guide***
- ***D99875585 DynaPro Programmer's Reference (Commands)***
- ***D99875629 DynaPro Mini Programmer's Reference (Commands)***
- ***D998200136 DynaPro Go Programmer's Manual (Commands)***
- ***D99875656 IPAD, DynaPro, DynaPro Go, DynaPro Mini Programmer's Reference (C++)***
- ***D99875668 IPAD, DynaPro, DynaPro Go, DynaPro Mini Programmer's Reference (Android)***
- ***D99875633 IPAD, DynaPro, DynaPro Go, DynaPro Mini Programmer's Reference (Java / Java Applet)***
- ***D99875654 IPAD, DynaPro, DynaPro Go, DynaPro Mini Programmer's Reference (iOS)***

1.1 About the MagTek PIN Pad SCRA C++ Demo

The MTPPSCRA C++ Demo, available from MagTek, provides demonstration source code and reusable MTPPSCRA DLLs that provide developers of custom software solutions with an easy-to-use interface for IPAD, DynaPro, DynaPro Go, and DynaPro Mini. Developers can include the MTPPSCRA DLLs in custom branded software which can be distributed to customers or distributed internally as part of an enterprise solution.

1.2 Nomenclature

The general terms “device” and “host” are used in different, often incompatible ways in a multitude of specifications and contexts. For example “host” may have different meanings in the context of USB communication than it does in the context of networked financial transaction processing. In this document, “device” and “host” are used strictly as follows:

- **Device** refers to the PIN Entry Device (PED or PIN pad) that receives and responds to the command set specified in this document; in this case, IPAD, DynaPro, DynaPro Go, or DynaPro Mini.
- **Host** refers to the piece of general-purpose electronic equipment the device is connected or paired to, which can send data to and receive data from the device. Host types include PC and Mac computers/laptops, tablets, smartphones, teletype terminals, and even test harnesses. In many cases the host may have custom software installed on it that communicates with the PED. When “host” must be used differently, it is qualified as something specific, such as “USB host.”

Similarly, the word “user” is used in different ways in different contexts. In this document, **user** generally refers to the **cardholder**.

1.3 SDK Contents

File	Description
include\mtppscra.h	Header file of library

lib\mtppscra.lib	Lib file of library
bin\mtppscra.dll	Dynamic link library of windows
bin\mtppble.dll	Bluetooth Low Energy DLL, support windows 8 and above.
bin\MTIPADLibTest.exe	Demonstration software that shows how to use library and how to work with MagTek devices.
Doc\	Include document for this library
Samples\Demo\	Include a visual studio project and full source code to build MTIPADLibTest.exe.

1.4 System Requirements

Tested operating systems:
Windows 7
Windows 8
Windows 8.1
Windows 10

Tested development environments:
Windows 10 with Microsoft Visual Studio 2015

2 How to Set Up the MagTek PIN Pad SCRA Libraries

To set up the MTPPSCRA Libraries, download the ***IPAD/DynaPro/DynaPro Go/DynaPro Mini .NET API*** install, available from MagTek.com (**Support > PIN Pads > DynaPro > Software > IPAD/DynaPro/DynaPro Go/DynaPro Mini Windows API**) and run **99510127.exe**.

To set up the MTPPSCRA Libraries, follow these steps:

1) Copy **bin\MTPPSCRA.dll** and paste it to one of the following locations:
   a) If you are running a 32-bit operating system, paste to **C:\Windows\SysWOW64**.
   b) If you are running a 64-bit operating system, paste to **C:\Windows\System32**.

To build the MTPPSCRA Demo software, follow these steps:

1) Launch Visual Studio 2013 and open **samples\Demo\MTIPADLibTest.vcxproj**.
2) In the **Solution Explorer**, select **MTIPADLibTest**.
3) Select **Build** > **Build MTIPADLibTest**, or press **Shift-F6**.

To Run/Debug the MTPPSCRA Demo software, follow these steps:

1) Copy all DLL files in the **bin** folder and paste to one of the following locations:
   a) If you are running a 32-bit operating system, paste to **C:\Windows\SysWOW64**.
   b) If you are running a 64-bit operating system, paste to **C:\Windows\System32**.
2) In VisualStudio 2013, select **Debug**->**Start Without Debugging** to run the MTPPSCRA Demo, or select **Debug**->**Start Debugging** to run it in debug mode.

3 MTPPSCRA Library Functions

After creating an instance of the MTTPSCRA object in your custom software project, use the functions described in this section to communicate with IPAD, DynaPro, DynaPro Go, or DynaPro Mini.

3.1 GetSDKVersion

This function retrieves the MTPPSCRA library version information.

```
const char* _stdcall GetSDKVersion();
```

Return Value:
Returns a string containing the SDK version.

3.2 OpenDevice

This function opens a connection to the device.

```
int _stdcall OpenDevice();
```

Return Value:
Returns an `int` value (0: Success, Non-Zero: Error)

3.3 CloseDevice

This function closes the connection to the device.

```
int _stdcall CloseDevice();
```

Return Value:
Returns an `int` value (0: Success, Non-Zero: Error)

3.4 GetDeviceList

This function enumerate all IPAD devices.

```
const char* __stdcall GetDeviceList();
```

Return Value:
Returns a string, can contain Zero or more device paths, those paths are separated by ‘,’

3.5 SetDeviceTypeParameter

This function sets the device path to be used by `OpenDevice()`.

```
void _stdcall SetDeviceTypeParameter(const char* schemePath);
```

Parameter	Description
schemePath	A path to the device. Examples: • `TLSTRUST://DEVICESERIAL` will open a device TLS1.2 Wireless connection. • `IP://10.57.10.87:26` will open a device at Ethernet IP address 10.57.10.87, port 26. • `USB://98AF14220612070C` will open a device connected a USB port, using the device's serial number. • BLE://DynaProMini10D0C will open a device conneted to Bluetooth Low Energy host, using the device friend name. • If `schemePath` is empty or NULL, the function will open the first USB device.

3.6 IsDeviceOpened

This function retrieves the device's open status.

```
bool _stdcall IsDeviceOpened();
```

Return Value:
`True` if the host is connected to the device, otherwise `False`.

3.7 DeviceReset

This function sends a reset command to the device.

```
int _stdcall DeviceReset(int* opStatus);
```

Parameter	Description
opStatus	Integer pointer to receive the command response or operation status. Zero value means OK. For more values, see **Appendix B**.

Return Value:
Returns an `int` value (0: Success, Non-Zero: Error)

3.8 SetTimeOut

This function set the defaut time out value for operation.

```
void __stdcall SetTimeOut(int timeOutValue);
```

Parameter	Description
timeOutValue	Integer value of time out, in milliseconds. Default time out value is 1000 milli-second

Return Value:
None

3.9 GetStatusCode

This function retrieves the current status of the issued report.

```
int _stdcall GetStatusCode();
```

Return Value:
Returns the current status of the device after issuing a command. See **Appendix B** for details.

3.10 CancelOperation

This function directs the device to abort the previously issued command.

```
int _stdcall CancelOperation();
```

Return Value:
Returns an `int` value (0: Success, Non-Zero: Error)

3.11 RequestBypassPINCommand

This function sends the Bypass PIN command to the device. This affects the behavior of **RequestSmartCard**.

```
int _stdcall RequestBypassPINCommand(int* opStatus);
```

Parameter	Description
opStatus	Integer pointer to receive the command response or operation status. Zero value means OK. For more values, see **Appendix B**.

Return Value:
Returns an `int` value (0: Success, Non-Zero: Error)

3.12 SetPAN

This function wraps device command `0x0D`. It sends card PAN data to the device in cases where the PAN is coming from a source other than the card being processed.

```
int _stdcall SetPAN(char* accountNumber, int* opStatus);
```

Parameter	Description
accountNumber	value for PAN data (8 - 19 ASCII digits) in a null terminated string.
opStatus	Integer pointer to receive the command response or operation status. Zero value means OK. For more values, see **Appendix B**.

Return Value:
Returns an `int` value (0: Success, Non-Zero: Error)

3.13 SetAmount

This function sets the transaction amount before beginning a transaction.

```
int _stdcall SetAmount(
      int amountType,
      char* amount,
      int* opStatus)
```

Parameter	Description
amountType	Type of amount use: 0x67 = Credit 0x68 = Debit
amount	Amount to be used for the transaction, should be a null terminated string. For example "20.56"
opStatus	Integer pointer to receive the command response or operation status. Zero value means OK. For more values, see **Appendix B**.

Return Value:
Returns an `int` value (0: Success, Non-Zero: Error)

3.14 EndSession

This synchronous function wraps device command `0x02`. It directs the device to clear all existing session data including PIN, PAN, and amount. The device returns to the idle state and sets the display to the specified Welcome screen. Use of message IDs 1-4 require that the associated bitmaps have been previously loaded during configuration; otherwise, use `0` for `displayMessageID` and the device will display its default "Welcome" screen (shown below).

```
int _stdcall EndSession(int displayMessageID = 0);
```

Figure 3-1 - DynaPro Welcome Screen

Figure 3-2 - DynaPro Mini Welcome Screen

Parameter	Description
displayMessageID	value between 0 - 4 indicating the message to display

Return Value:
Returns an `int` value (0: Success, Non-Zero: Error)

3.15 RequestChallengeAndSessionForInformation (EMV L1 only)

This function retrieves the challenge key and session key for a smart card transaction.

```
int _stdcall RequestChallengeAndSessionForInformation(
     unsigned char* buffer,
     int* bufferLen,
     int* opStatus);
```

Parameter	Description
buffer	Pointer to a buffer to receive the challenge and session key data. See report `0xA9` in ***99875585 DynaPro Programmer's Reference (Commands)*** and/or ***99875629 DynaPro Mini Programmer's Reference (Commands)*** for details.
bufferLen	Size of the buffer.
opStatus	Integer pointer to receive the command response or operation status. Zero value means OK. For more values, see **Appendix B**.

Return Value:
Returns an `int` value (0: Success, Non-Zero: Error)

3.16 RequestConfirmSession (EMV L1 only)

This function sends a CMAC message to the device to confirm the session key.

```
int _stdcall RequestConfirmSession(
     int mode,
     unsigned char* encryptedRandomNumber,
     unsigned char* encryptedSerialNumber,
     unsigned char* cmac,
     int* opStatus);
```

Parameter	Description
mode	Mode: 0 - End Session 1 - Confirm Session
encryptedRandomNumber	32-bit encrypted random number
encryptedSerialNumber	32-bit encrypted partial serial number
cmac	64-bit CMAC
opStatus	Integer pointer to receive the command response or operation status. Zero value means OK. For more values, see **Appendix B**.

Return Value:
Returns an `int` value (0: Success, Non-Zero: Error)

3.17 RequestConfirmSessionWithMode (EMV L1 only) [Deprecated: V5.01]

This function sends a CMAC message to the device to confirm the session key.

```
int _stdcall RequestConfirmSessionWithMode(
      int mode,
      unsigned char* encryptedRandomNumber,
      unsigned char* encryptedSerialNumber,
      unsigned char* cmac,
      int* opStatus);
```

Parameter	Description
mode	Mode: 0 - End Session 1 - Confirm Session
encryptedRandomNumber	32-bit encrypted random number
encryptedSerialNumber	32-bit encrypted partial serial number
cmac	64-bit CMAC
opStatus	Integer pointer to receive the command response or operation status. Zero value means OK. For more values, see **Appendix B**.

Return Value:
Returns an `int` value (0: Success, Non-Zero: Error)

3.18 EndL1Session (EMV L1 only)

This function ends the session with the device.

```
int _stdcall EndL1Session(
      int* opStatus);
```

Parameter	Description
opStatus	Integer pointer to receive the command response or operation status. Zero value means OK. For more values, see **Appendix B**.

Return Value:
Returns an `int` value (0: Success, Non-Zero: Error)

3.19 RequestPowerUpResetICC (EMV L1 only)

This function will prompt the user to insert a smart card, then power it up when inserted. The event associated with this command is **OnPowerUpICC**.

```
int _stdcall RequestPowerUpResetICC(
      int waitTime,
      int operation);
```

Parameter	Description
waitTime	Wait time
Operation	Operation ID 0 = Power down 1 = Power up or warm reset

Return Value:
Returns an `int` value (0: Success, Non-Zero: Error)

3.20 RequestPowerUpResetICCWithWaitTime (EMV L1 only) [Deprecated: V5.01]

This function will prompt the user to insert a smart card, then power it up when inserted. The event associated with this command is **OnPowerUpICC**.

```
int _stdcall RequestPowerUpResetICCWithWaitTime(
      int waitTime,
      int operation);
```

Parameter	Description
waitTime	Wait time
Operation	Operation ID 0 = Power down 1 = Power up or warm reset

Return Value:
Returns an `int` value (0: Success, Non-Zero: Error)

3.21 RequestPowerDownICC (EMV L1 only)

This function requests that the device power down an inserted smart card.

```
int _stdcall RequestPowerDownICC(int waitTime);
```

Parameter	Description
waitTime	Not used.

Return Value:
Returns an `int` value (0: Success, Non-Zero: Error)

3.22 RequestPowerDownICCWithWaitTime (EMV L1 only) [Deprecated: V5.01]

This function requests that the device power down an inserted smart card.

```
int _stdcall RequestPowerDownICCWithWaitTime(int waitTime);
```

Parameter	Description
waitTime	Not used.

Return Value:
Returns an `int` value (0: Success, Non-Zero: Error)

3.23 RequestICCAPDUForInformation (EMV L1 only)

This function sends the ICC APDU report to the device. It will send a secure APDU to an inserted smart card, and will receive a secure response from the card. The event associated with this command is **OnAPDUArrived**.

```
int _stdcall RequestICCAPDUForInformation(
     unsigned char* apdu,
     int apduLen);
```

Parameter	Description
apdu	An array of APDU bytes to send out
apduLen	Size of the APDU byte array

Return Value:
Returns an `int` value (0: Success, Non-Zero: Error)

3.24 SendSpecialCommand

This function sends a direct "SET" byte command to the device. For information about direct commands, see ***99875585 DynaPro Programmer's send (Commands)*** and/or ***99875629 DynaPro Mini Programmer's Reference (Commands)***. The event associated with this command is **OnDataReady**.

```
int _stdcall SendSpecialCommand(
     unsigned char* specialCommand,
     int commandLength);
```

Parameter	Description
specialCommand	An array of command bytes to send to the device.
commandLength	Length of the specialCommand array.

Return Value:
Returns an `int` value (0: Success, Non-Zero: Error)

3.25 SendSpecialCommandWithCommand [Deprecated: V5.01]

This function sends a direct "SET" byte command to the device. For information about direct commands, see ***99875585 DynaPro Programmer's send (Commands)*** and/or ***99875629 DynaPro Mini Programmer's Reference (Commands)***. The event associated with this command is **OnDataReady**.

```
int _stdcall SendSpecialCommandWithCommand(
     unsigned char* specialCommand,
     int commandLength);
```

Parameter	Description
specialCommand	An array of command bytes to send to the device.
commandLength	Length of the specialCommand array.

Return Value:
Returns an `int` value (0: Success, Non-Zero: Error)

3.26 GetSpecialCommand

This function sends a direct "GET" byte command to the device. For information about direct commands, see ***99875585 DynaPro Programmer's Reference (Commands)*** and/or ***99875629 DynaPro Mini Programmer's Reference (Commands)***.

```
int _stdcall GetSpecialCommand(
     unsigned char* specialCommand,
     int commandLength,
     unsigned char* pSpecialCommandData,
     int* specialDataLen);
```

Parameter	Description
specialCommand	An array of command bytes to send to the device.
commandLength	Length of the specialCommand array.
pSpecialCommandData	A buffer pointer to receive returned data.
specialDataLen	Buffer size to allocate for returned data.

Return Value:
Returns an `int` value (0: Success, Non-Zero: Error)

3.27 GetSpecialCommandWithCommand [Deprecated: V5.01]

This function sends a direct "GET" byte command to the device. For information about direct commands, see ***99875585 DynaPro Programmer's Reference (Commands)*** and/or ***99875629 DynaPro Mini Programmer's Reference (Commands)***.

```
int _stdcall GetSpecialCommandWithCommand(
     unsigned char* specialCommand,
     int commandLength,
     unsigned char* pSpecialCommandData,
     int* specialDataLen);
```

Parameter	Description
specialCommand	An array of command bytes to send to the device.
commandLength	Length of the specialCommand array.
pSpecialCommandData	A buffer pointer to receive returned data.
specialDataLen	Buffer size to allocate for returned data.

Return Value:
Returns an `int` value (0: Success, Non-Zero: Error)

3.28 RequestGetEMVTags

This function sends the EMV Tag report to the device to read or write EMV Tags. The event associated with this command is **OnEMVTagsComplete**.

```
int _stdcall RequestGetEMVTags(
     int tagType,
     int tagOperation,
     unsigned char* inputTLVData,
     int inputDataLength);
```

Parameter	Description
tagType	EMV Tag to set or get: 0x00 = Reader Tags 0x80 = Application Tags Lower 7 bits indicate which application slot of operation
tagOperation	Type of operation to be performed: 0x00 = Read Reader tag 0x01 = Read All EMV Reader tags 0x02 = Read EMV Application tags 0x03 = Read All EMV Application tags 0x0F = Read All PIN-PAD or Application tags
inputTLVData	TLV Data Block to be sent to the device
inputDataLength	Length of the TLV Data Block

Return Value:
Returns an `int` value (0: Success, Non-Zero: Error)

3.29 RequestGetEMVTagsWithTagType [Deprecated: V5.01]

This function sends the EMV Tag report to the device to read or write EMV Tags. The event associated with this command is **OnEMVTagsComplete**.

```
int _stdcall RequestGetEMVTagsWithTagType(
      int tagType,
      int tagOperation,
      unsigned char* inputTLVData,
      int inputDataLength);
```

Parameter	Description
tagType	EMV Tag to set or get: 0x00 – Reader Tags 0x80 – Application Tags Lower 7 bits indicate which application slot of operation
tagOperation	Type of operation to be performed: 0 = Read Operation 0x0F = Read All PIN-PAD or Application Tags
inputTLVData	TLV Data Block to be sent to the device
inputDataLength	Length of the TLV Data Block

Return Value:
Returns an `int` value (0: Success, Non-Zero: Error)

3.30 RequestGetEMVTagsEx

This function sends the EMV Tag report to the device to read or write EMV Tags. The event associated with this command is **OnEMVTagsComplete**.

```
int _stdcall RequestGetEMVTagsEx(
      int tagType,
      int tagOperation,
      unsigned char* inputTLVData,
      int inputDataLength,
      int database,
      int option,
      unsigned char* reserved);
```

Parameter	Description
tagType	EMV Tag to set or get: 0x00 = Reader Tags 0x80 = Application Tags Lower 7 bits indicate which application slot of operation

tagOperation	Type of operation to be performed: 0x00 = Read Reader tag 0x01 = Read All EMV Reader tags 0x02 = Read EMV Application tags 0x03 = Read All EMV Application tags 0x0F = Read All PIN-PAD or Application tags
inputTLVData	TLV Data Block to be sent to the device
inputDataLength	Length of the TLV Data Block
database	Database Selector: 00 = Contact L2 EMV Tags 01 = PayPass-MasterCard 02 = PayWave-VISA 03 = ExpressPay-AMEX 04 = Discover
option	Response type: 0x00 = Normal 0x01 = Delay Response
reserved	Reserved Bytes

Return Value:
Returns an `int` value (0: Success, Non-Zero: Error)

3.31 RequestSetEMVTags

This function sends the EMV Tag report to the device to read or write EMV Tags.

```
int _stdcall RequestSetEMVTags(
     int tagType,
     int tagOperation,
     unsigned char* inputTLVData,
     int inputDataLength);
```

Parameter	Description
tagType	value for the EMV Tag to set or get: 0x00 – Reader Tags 0x80 – Application Tags Lower 7 bits indicate which application slot of operation
tagOperation	Type of operation to be performed: 0x04 = Write EMV Reader tags 0x05 = Write EMV Application tags 0xFF = Set to factory defaults
inputTLVData	TLV Data Block to send to the device
inputDataLength	Length of the TLV Data Block

Return Value:
Returns an `int` value (0: Success, Non-Zero: Error)

3.32 RequestSetEMVTagsWithTagType [Deprecated: V5.01]

This function sends the EMV Tag report to the device to read or write EMV Tags.

```
int _stdcall RequestSetEMVTagsWithTagType(
     int tagType,
     int tagOperation,
     unsigned char* inputTLVData,
     int inputDataLength);
```

Parameter	Description
tagType	value for the EMV Tag to set or get: 0x00 – Reader Tags 0x80 – Application Tags Lower 7 bits indicate which application slot of operation
tagOperation	Type of operation to be performed: 1 = Write Operation 0xFF = Set to factory defaults
inputTLVData	TLV Data Block to send to the device
inputDataLength	Length of the TLV Data Block

Return Value:
Returns an `int` value (0: Success, Non-Zero: Error)

3.33 RequestSetEMVTagsEx

This function sends the EMV Tag report to the device to read or write EMV Tags.

```
int _stdcall RequestSetEMVTagsWithTagType(
     int tagType,
     int tagOperation,
     unsigned char* inputTLVData,
     int inputDataLength int database,
     int option,
     unsigned char* reserved);
```

Parameter	Description
tagType	Value for the EMV Tag to set or get: 0x00 = Reader Tags 0x80 = Application Tags Lower 7 bits indicate which application slot of operation
tagOperation	Type of operation to be performed: 0x04 = Write EMV Reader tags 0x05 = Write EMV Application tags 0xFF = Set to factory defaults
inputTLVData	TLV Data Block to send to the device

inputDataLength	Length of the TLV Data Block
database	Database Selector: 00 = Contact L2 EMV Tags 01 = PayPass - MasterCard 02 = PayWave - VISA 03 = ExpressPay - AMEX 04 = Discover
option	Response type: 0x00 = Normal 0x01 = Delay Response
reserved	Reserved Bytes

Return Value:
Returns an `int` value (0: Success, Non-Zero: Error)

3.34 SetCAPublicKey

This function sets / deletes the corresponding CA Public Key, depending on the operation specified. When using the `0x0F` or `0x04` `operation` parameters, the event associated with this command is **OnGetCAPublicKey**.

```
int _stdcall SetCAPublicKey(
      int operation,
      unsigned char* keyBlock,
      int keyBlockLength);
```

Parameter	Description
operation	Type of operation to be performed: 0x00 = Erase all CA Public Keys 0x01 = Erase all CA Public Keys for a given RID 0x02 = Erase a single CA Public Key 0x03 = Add a single CA Public Key 0x04 = Read Single Public Key 0x0F = Read all CA Public Keys
keyBlock	CA Public Key to be sent
keyBlockLength	Length of the CA Public Key

Return Value:
Returns an `int` value (0: Success, Non-Zero: Error)

3.35 SetCAPublicKeyWithOperation [Deprecated: V5.01]

This function sets / deletes the corresponding CA Public Key, depending on the operation specified. When using the `0x0F` or `0x04` `operation` parameters, the event associated with this command is **OnGetCAPublicKey**.

```
int _stdcall SetCAPublicKeyWithOperation(
      int operation,
```

```
    unsigned char* keyBlock,
    int keyBlockLength);
```

Parameter	Description
operation	Type of operation to be performed: 0x00 = Erase all CA Public Keys 0x01 = Erase all CA Public Keys for a given RID 0x02 = Erase a single CA Public Key 0x03 = Add a single CA Public Key 0x04 = Read Single Public Key 0x0F = Read all CA Public Keys
keyBlock	CA Public Key to be sent
keyBlockLength	Length of the CA Public Key

Return Value:
Returns an `int` value (0: Success, Non-Zero: Error)

3.36 RequestDeviceConfigurationForInformation

This function retrieves the device configuration.

```
int _stdcall RequestDeviceConfigurationForInformation(
    unsigned char* buffer,
    int* bufferLen,
    int* opStatus);
```

Parameter	Description
buffer	Pointer to a buffer to receive configuration data
bufferLen	Buffer size
opStatus	Integer pointer to receive the command response or operation status. Zero value means OK. For more values, see **Appendix B**.

Return Value:
Returns an `int` value (0: Success, Non-Zero: Error)

3.37 SetDisplayMessage

This function shows a predefined message or bitmap on the device's LCD display. The device will display the specified message until `waitTime` has elapsed, then reset the display to a blank screen. The event associated with this function is **OnDisplayRequestComplete**.

```
int _stdcall SetDisplayMessage(
    int waitTime,
    int messageID,
    int* opStatus);
```

Parameter	Description
waitTime	Length of time the message will be displayed
messageID	Predefined message to be displayed
opStatus	Integer pointer to receive the command response or operation status. Zero value means OK. For more values, see **Appendix B**.

Return Value:
Returns an `int` value (0: Success, Non-Zero: Error)

3.38 SendBigBlockData

This function wraps device command `0x10`. It sends a packet of big block data to the device. For details on using big block data, see ***99875585 DynaPro Programmer's Reference (Commands)*** and/or ***99875629 DynaPro Mini Programmer's Reference (Commands)***.

```
int _stdcall SendBigBlockData(
     int dataTypeID,
     void* data,
     unsigned long dataLength,
     int* opStatus);
```

Parameter	Description
dataTypeID	Data type ID for this data. The device will use this type to process data with the next command.
data	Pointer to the data
dataLength	Data length in bytes
opStatus	Integer pointer to receive the command response or operation status. Zero value means OK. For more values, see **Appendix B**.

Return Value:
Returns an `int` value (0: Success, Non-Zero: Error)

3.39 SendBitmap

This function wraps device command `0x0C`. It directs the device to save new bitmap image data in the specified memory slot. The device can hold up to four bitmaps. Recommend bitmap size for monochrome image is 128x64 (1024 bytes), and Recommend bitmap size for color image is 320x240. For information about creating the bitmap image see ***99875622 Dynapro Image Installation Guide.***

If the `flag` parameter is `0` (“clear”), the current image will be cleared from the specified slot. Otherwise, if the command is successful, the new bitmap image data will be stored in the specified slot with the selected format, and will display (b/w or inverted) when the **EndSession** function is invoked.

```
int _stdcall SendBitmap(
     int slot,
     int option,
```

```
      unsigned char* bitmapData,
      int bitmapLen,
      int* opStatus);
```

Parameter	Description
slot	Device bitmap slot, 1 - 4.
option	Options flag: 0 = Clear 1 = Save 2 = Invert and Save
bitmapData	Pointer to the data
bitmapLen	Data length in bytes
opStatus	Integer pointer to receive the command response or operation status. Zero value means OK. For more values, see **Appendix B**.

Return Value:
Returns an `int` value (0: Success, Non-Zero: Error)

3.40 IsDeviceSRED

This function checks device is SRED (Secure Reading and Exchange of Data).
The SRED version of the firmware enables Secure Reading and Exchange of Data. In this mode, the device will not allow complete unmasking of card data, such as the PAN.
In some cases, the solution may require further options for unmasking and encrypting card data before the device transmits it to the host. In those cases, the device can be loaded with the Non-SRED version of the firmware.

```
bool __stdcall IsDeviceSRED();
```

Return Value:
Returns an `bool` value (true: SRED Firmware, false: Non-SRED firmware)

3.41 UpdateFirmware

This function wraps device command `0x17`. It directs the device to update the firmware from data buffer.

```
int _stdcall UpdateFirmware(
      void* firmwareData,
      unsigned int firmwareDataLength,
      int* opStatus);
```

Parameter	Description
firmwareData	A pointer to firmware data buffer.
firmwareDataLength	Data length in bytes.

opStatus	Integer pointer to receive the command response or operation status. Zero value means OK. For more values, see **Appendix B**.

Return Value:
Returns an `int` value (0: Success, Non-Zero: Error)

3.42 GetIPADInfoData

This function returns information about the device in an `IPADDevInfo` class, defined below.

```
int _stdcall GetIPADInfoData(IPADDevInfo* pIPADDevInfo);
```

Parameter	Description
pIPADDevInfo	An `IPADDevInfo` structure to hold IPAD information. See type definition below.

```
typedef struct _IPADDevInfo
{
     char * Model;
     char * DevicePath;
     char * Serial;
     char * FWVersion;
     int  Version;
     int  PID;
     int  VID;
} IPADDevInfo, *pIPADDevInfo;
```

Return Value:
Returns an `int` value (0: Success, Non-Zero: Error)

3.43 RequestDeviceInformation

This synchronous function wraps device command `0x1A`. It returns the device information specified by the `mode` parameter.

```
int _stdcall RequestDeviceInformation(
     int mode,
     char* infoBuffer,
     int len,
     int* opStatus);
```

Parameter	Description
mode	Information requested: 0 – Product_ID 1 – Maximum Application Message Size 2 – Capability String 3 – Manufacturer 4 – Product Name 5 – Serial Number 6 – Firmware Number 7 – Build Info 8 – MAC address for ethernet versions only A – Boot1 Firmware Version B – Boot2 Firmware Version
infoBuffer	A pointer to a buffer to receive the device information. After success, the buffer contains a null terminated string.
len	Size of buffer
opStatus	Integer pointer to receive the command response or operation status. Zero value means OK. For more values, see **Appendix B**.

Return Value:
Returns an `int` value (0: Success, Non-Zero: Error)

3.44 RequestDeviceInformationWithMode [Deprecated: V5.01]

This synchronous function wraps device command `0x1A`. It returns the device information specified by the `mode` parameter.

```
int _stdcall RequestDeviceInformationWithMode(
     int mode,
     char* infoBuffer,
     int len,
     int* opStatus);
```

Parameter	Description
mode	Information requested: 0 – Product_ID 1 – Maximum Application Message Size 2 – Capability String 3 – Manufacturer 4 – Product Name 5 – Serial Number 6 – Firmware Number 7 – Build Info 8 – MAC address for ethernet versions only A – Boot1 Firmware Version B – Boot2 Firmware Version

infoBuffer	A pointer to a buffer to receive the device information. After success, the buffer contains a null terminated string.
len	Size of buffer
opStatus	Integer pointer to receive the command response or operation status. Zero value means OK. For more values, see **Appendix B**.

Return Value:
Returns an `int` value (0: Success, Non-Zero: Error)

3.45 RequestDeviceStatusForInformation

This function retrieves the device's status information in a `DEV_STATE_STAT` class, defined below. The event associated with this function is **OnDeviceStateUpdated**.

```
int _stdcall RequestDeviceStatusForInformation(
      DEV_STATE_STAT* status,
      int* opStatus);
```

Parameter	Description
status	`DEV_STATE_STAT` structure containing the following device status information: • DeviceState • SessionState • DeviceStatus • DevCertStatus • HWStatus See below for type definition.
opStatus	Integer pointer to receive the command response or operation status. Zero value means OK. For more values, see **Appendix B**.

```
typedef struct _DEV_STATE_STAT
{
      BYTE nDeviceState;
      BYTE nSessionState;
      BYTE nDeviceStatus;
      BYTE nDevCertStatus;
      BYTE nHWStatus;
      BYTE nICCMasterSessKeyStatus;
} DEV_STATE_STAT;
```

Return Value:
Returns an `int` value (0: Success, Non-Zero: Error)

3.46 RequestKernelInformation

This function retrieves the device's kernel information.

```
int _stdcall RequestKernelInformation(
     int kernelInfoID,
     unsigned char* kernelInfoBuffer,
     int* kernelInfoBufferLen,
     int* opStatus);
```

Parameter	Description
kernelInfoID	value containing the Key information ID: 0x00 – Version L1 Kernel 0x01 – Version L2 Kernel 0x02 – Checksum/Signature L1 Kernel 0x03 – Checksum/Signature L2 Kernel 0x03 – Checksum/Signature L2 Kernel + Configuration
kernelInfoBuffer	A pointer to receive the kernel information.
kernelInfoBufferLen	Pointer to hold the size of the kernelInfoBuffer after a successful call.
opStatus	Integer pointer to receive the command response or operation status. Zero value means OK. For more values, see **Appendix B**.

Return Value:
Returns an `int` value (0: Success, Non-Zero: Error)

3.47 GetBINTableData

This function retrieves the BIN table data, see report `0x32`.

```
int _stdcall GetBINTableData(
     unsigned char* buffer,
     int* bufferLen,
     int* opStatus);
```

Parameter	Description
buffer	Pointer to a buffer to receive BIN table data.
bufferLen	Contains the size of buffer after a successful call.
opStatus	Integer pointer to receive the command response or operation status. Zero value means OK. For more values, see **Appendix B**.

Return Value:
Returns an `int` value (0: Success, Non-Zero: Error)

3.48 SetBINTableData

This function retrieves the BIN table data, see report `0x32`.

```
int _stdcall SetBINTableData(
     int reserved
     unsigned char* binTable,
     int tableLen,
```

```
      int* opStatus);
```

Parameter	Description
Reserved	Put 0 here, reserved for future use.
binTable	Buffer pointer to the BIN table byte array
tableLen	Size of binTable in bytes
opStatus	Integer pointer to receive the command response or operation status. Zero value means OK. For more values, see **Appendix B**.

Return Value:
Returns an `int` value (0: Success, Non-Zero: Error)

3.49 GetKSN

This function retrieves the device KSN value. It requires that the software first call **RequestPIN** or **RequestCard** for valid KSN data. This feature is used for the Token Reversal Function and not supported on DynaPro Go.

```
int _stdcall GetKSN(
      unsigned char* ksn,
      int* ksnLen,
      int* opStatus);
```

Parameter	Description
ksn	Pointer to a buffer to receive the KSN
ksnLen	Contains the size of ksn after a successful call.
opStatus	Integer pointer to receive the command response or operation status. Zero value means OK. For more values, see **Appendix B**.

Return Value:
Returns an `int` value (0: Success, Non-Zero: Error)

3.50 SetKSNEncryptedData

This function will display the KSN encrypted data on the device.

```
int _stdcall SetKSNEncryptedData(
      int waitTime,
      unsigned char* data,
      int dataLen,
      int* opStatus);
```

Parameter	Description
waitTime	Seconds to display the KSN on the device.
data	Pointer to encrypted data.

dataLen	Size of encrypted data in bytes.
opStatus	Integer pointer to receive the command response or operation status. Zero value means OK. For more values, see **Appendix B**.

Return Value:
Returns an `int` value (0: Success, Non-Zero: Error)

3.51 RequestCard

This function wraps device command `0x03`. It directs the device to prompt the user to swipe a card by displaying one of four predefined messages and playing a specified sound. Example request screens look like the figures below. The event associated with this function is **OnCardRequestComplete**.

```
int _stdcall RequestCard(
      int waitTime,
      int displayMessage,
      int beepTones);
```

Figure 3-3 - DynaPro Swipe Prompts

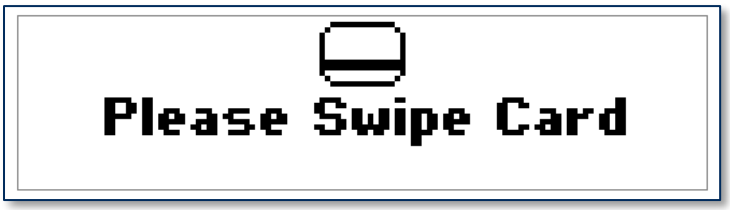

Figure 3-4 - DynaPro Mini Initial Swipe Prompt

Parameter	Description
waitTime	Time the device will wait for the user to complete a card swipe
messageID	Message to prompt the user with: 0x00 - CardMsgSwipeCardIdle 0x01 - CardMsgSwipeCard 0x02 - CardMsgPleaseSwipeCard 0x03 - CardMsgPleaseSwipeAgain
tone	Tone to use: 0x00 – No Sound 0x01 – Single Beep 0x02 – Double Beeps

Return Value:

Returns an `int` value (0: Success, Non-Zero: Error)

3.52 RequestManualCardData

This function triggers the device to begin a manual card data entry transaction. The event associated with this function is **OnCardRequestComplete**.

```
int _stdcall RequestManualCardData(
      int waitTime,
      int beepTones,
      int options);
```

Parameter	Description
waitTime	Time the device will wait for user to begin manual data entry
beepTones	Tone to use: 0 - None 1 - Single Beep 2 - Double Beep
option	This is an ORed combination of flags that changes the device's data entry request behavior as follows: Bits 0 and 1 0 = Acct,Date,CVC 1 = Acct,Date 2 = Acct,CVC 3 = Acct Bit 2 1=Use Qwick Codes entry Bit 3 1=Use PAN in PIN block creation Bit 4 0=Use PAN min 9, max 19 1=Use PAN min 14, max 21 Bits 5-7 are reserved and should be set to 0.

Return Value:
Returns an `int` value (0: Success, Non-Zero: Error)

3.53 RequestManualCardDataWithWaitTime [Deprecated: V5.01]

This function triggers the device to begin a manual card data entry transaction. The event associated with this function is **OnCardRequestComplete**.

```
int _stdcall RequestManualCardDataWithWaitTime(
      int waitTime,
      int beepTones,
```

```
    int options);
```

Parameter	Description
waitTime	Time the device will wait for user to begin manual data entry
beepTones	Tone to use: 0 - None 1 - Single Beep 2 - Double Beep
option	This is an ORed combination of flags that changes the device's data entry request behavior as follows: Bits 0 and 1 0 = Acct,Date,CVC 1 = Acct,Date 2 = Acct,CVC 3 = Acct Bit 2 1=Use Qwick Codes entry Bit 3 1=Use PAN in PIN block creation Bit 4 0=Use PAN min 9, max 19 1=Use PAN min 14, max 21 Bits 5-7 are reserved and should be set to 0.

Return Value:
Returns an `int` value (0: Success, Non-Zero: Error)

3.54 RequestUserDataEntry

This function sends the User Data Entry report to the device. The device will prompt the user to enter SSN, zip code, or birth date by displaying one of four predefined messages. The event associated with this command is **OnUserDataEntry**.

```
int _stdcall RequestUserDataEntry(
     int waitTime,
     int displayMessageID,
     int beepTones);
```

Parameter	Description
waitTime	Time the device will wait for the user to begin data entry

displayMessageID	Message to prompt the user with: 0 – SSN 1 – Zip code 2 – Birth (four-digit year) 3 – Birth (two-digit year)
beepTones	Tone to use: 0 - None 1 - Single Beep 2 - Double Beep

Return Value:
Returns an `int` value (0: Success, Non-Zero: Error)

3.55 RequestUserDataEntryWithWaitTime [Deprecated: V5.01]

This function sends the User Data Entry report to the device. The device will prompt the user to enter SSN, zip code, or birth date by displaying one of four predefined messages. The event associated with this command is **OnUserDataEntry**.

```
int _stdcall RequestUserDataEntryWithWaitTime(
     int waitTime,
     int displayMessageID,
     int beepTones);
```

Parameter	Description
waitTime	Time the device will wait for the user to begin data entry
displayMessageID	Message to prompt the user with: 0 – SSN 1 – Zip code 2 – Birth (four-digit year) 3 – Birth (two-digit year)
beepTones	Tone to use: 0 - None 1 - Single Beep 2 - Double Beep

Return Value:
Returns an `int` value (0: Success, Non-Zero: Error)

3.56 RequestClearTextUserDataEntry

This function sends the Clear Text User Data Entry report to the device that supports this feature. The device will prompt the user to enter SSN, zip code, or birth date by displaying one of four preset messages. Similar to **RequestUserDataEntry**, but data will displayed and returned in clear text. The event associated with this command is **OnClearTextUserDataEntry**.

```
int _stdcall RequestClearTextUserDataEntry(
     int waitTime,
     int displayMessageID,
```

```
    int beepTones);
```

Parameter	Description
waitTime	Time the device will wait for the user to begin data entry
displayMessageID	Message to prompt the user with: 0 – SSN 1 – Zip code 2 – Birth (four-digit year) 3 – Birth (two-digit year)
beepTones	Tone to use: 0 - None 1 - Single Beep 2 - Double Beep

Return Value:
Returns an `int` value (0: Success, Non-Zero: Error)

3.57 RequestClearTextUserDataEntryWithWaitTime [Deprecated: V5.01]

This function sends the Clear Text User Data Entry report to the device that supports this feature. The device will prompt the user to enter SSN, zip code, or birth date by displaying one of four preset messages. Similar to **RequestUserDataEntryWithWaitTime**, but data will displayed and returned in clear text. The event associated with this command is **OnClearTextUserDataEntry**.

```
int _stdcall RequestClearTextUserDataEntryWithWaitTime(
     int waitTime,
     int displayMessageID,
     int beepTones);
```

Parameter	Description
waitTime	Time the device will wait for the user to begin data entry
displayMessageID	Message to prompt the user with: 0 – SSN 1 – Zip code 2 – Birth (four-digit year) 3 – Birth (two-digit year)
beepTones	Tone to use: 0 - None 1 - Single Beep 2 - Double Beep

Return Value:
Returns an `int` value (0: Success, Non-Zero: Error)

3.58 RequestResponse

This function sends the Response report to the device. The device will prompt the user to select a transaction type or user defined message. The event associated with this function is **OnKeyInput**.

```
int _stdcall RequestResponse(
      int waitTime,
      int selectMsg,
      int keyMask,
      int beepTones);
```

Parameter	Description
waitTime	Time the device will wait for the user to select the option
selectMsg	Message to prompt the user with: 0 – Transaction Type (Credit/Debit) 1 – Verify Transaction Amount 2 – Credit Other Debit 3 – Credit EBT Debit 4 – Credit Gift Debit 5 – EBT Gift Other 255 – User Defined Message
keyMask	Key codes to mask (combine using OR): 1 – Left 2 – Middle 4 – Right 8 – Enter
beepTones	Tone to be used: 0 - None 1 - Single Beep 2 - Double Beep

Return Value:
Returns an `int` value (0: Success, Non-Zero: Error)

3.59 ConfirmAmount

This function prompts the user to confirm the transaction amount. The amount should be set by using **SetAmount**. The event associated with this command is **OnKeyInput**.

```
int _stdcall ConfirmAmount(
      int waitTime,
      int beepTones);
```

Parameter	Description
waitTime	Time the device will wait for the user to confirm the amount

beepTones	Tone to use: 0 - None 1 - Single Beep 2 - Double Beep

Return Value:
Returns an `int` value (0: Success, Non-Zero: Error)

3.60 SelectCreditDebit

This function will prompt user to confirm card type. The event associated with this command is **OnKeyInput**.

```
int _stdcall SelectCreditDebit(
      int waitTime,
      int beepTones);
```

Parameter	Description
waitTime	Time the device will wait for the user to select the option
beepTones	Tone to use: 0 - None 1 - Single Beep 2 - Double Beep

Return Value:
Returns an `int` value (0: Success, Non-Zero: Error)

3.61 RequestPIN

This function wraps device command `0x04`. It directs the device to prompt the user to enter a PIN by displaying one of five predetermined messages and playing a specified sound. The messages on the device's screen look like the figures below. The event associated with this function is **OnPINRequestComplete**.

```
int _stdcall RequestPIN(
      int waitTime,
      int pinMode,
      int minPINLength,
      int maxPINLength,
      int beepTones,
      int option);
```

Figure 3-5 - DynaPro PIN Prompts

Figure 3-6 - DynaPro Mini Initial PIN Prompt

Parameter	Description
waitTime	Time the device should wait for the user to begin PIN entry
pinMode	Message to display as a user prompt: 0 = PINsgEnterPIN 1 = PINMsgEnterPINAmt 2 = PINMsgReenterPINAmt 3 = PINMsgReenterPIN 4 = PINMsgVerifyPIN
minPINLength	Minimum PIN length. Must be greater than 3.
maxPINLength	Maximum PIN length. Must be less than 13.
beepTones	Tone to use: 0 = No sound 1 = Single beep 2 = Double beep
option	PIN verification and format: 0 = ISO0 Format, No verify PIN 1 = ISO3 Format, No verify PIN 2 = ISO0 Format, Verify PIN 3 = ISO3 Format, Verify PIN

Return Value:
Returns an `int` value (0: Success, Non-Zero: Error)

3.62 RequestSignature

This function sends the Request Signature report to the device. The device will prompt the user to sign. The event associated with this command is **OnSignatureArrived**.

```
int _stdcall RequestSignature(
      int waitTime,
```

```
    int beepTones,
    int option);
```

Parameter	Description
waitTime	Time the device should wait for the user to begin signing
beepTones	Tone to use: 0 - None 1 - Single Beep 2 - Double Beep
option	Option to verify or not to verify the PIN: 0 – Timeout to clean data 1 – Timeout with available data, signature can be retrieved if exists

Return Value:
Returns an `int` value (0: Success, Non-Zero: Error)

3.63 RequestSmartCard

This function wraps device command `0xA2`. It directs the device to prompt the user to confirm the transaction amount, and to arm the MSR and / or contact ICC reader to wait for a card to be swiped or presented into the contact ICC connector. If armed to read a contact ICC, the device will turn on the LED near the smart card connector after the cardholder confirms the transaction amount. The host should abort the transaction if the user presses the CANCEL button.

If there are no errors, the device will prompt the user to approve an amount and swipe or insert card by displaying pre-determined EMV messages.

The LCD display will cycle showing "(AMOUNT)," "(AMOUNT) OK?" and "CANCEL OR ENTER," and will wait for the cardholder to push either the confirmation or cancellation button.

If the cardholder presses the confirmation button, then depending on the card type requested to be read, the LCD display will show either SWIPE or INSERT CARD. If the user presses the cancellation button or the transaction times out, the device will perform the command completion action.

If the cardholder has inserted an ICC card, and if the Acquirer has set the device's payment brand account type setting for ICC to **Debit or Credit**, the device will prompt the cardholder to select debit or credit.

Per EMV 4.x requirements, if the cardholder uses the MSR input, the device will check the service code from the magnetic stripe data to see if it begins with a 2 or a 6 to determine if the card also includes an ICC, and will advise the cardholder that ICC is preferred by displaying USE CHIP READER. If the ICC fails or the service code does not begin with a 2 or a 6, the device will prompt the cardholder for an MSR swipe. After a successful swipe, the device will prompt the user to select debit or credit. If this is a debit account type, the device will request a PIN.

If the user presents an ICC card, the LCD display will show ICC applications that are mutually supported and ask the cardholder to choose the preferred application. If a PIN entry is needed per ***EMV 4.x*** requirements, the LCD will show ENTER PIN and start the PIN entry timer. If the user presses the cancelation button or the transaction times out, cancelled or timed out, the device will perform the command completion action.

After PIN entry, the device will display either PIN OK or will cycle through INCORRECT PIN and TRY AGAIN up to the PIN retry limit. If the number of attempts reaches PIN try limit-1, the device will display LAST TRY. If the user exceeds the PIN entry retry limit, the device will perform the command completion action, otherwise the transaction proceeds to the approval stage.

The device can be directed to allow PIN bypass using **RequestBypassPINCommand**. The PIN requirement can also be bypassed by the cardholder.

The transaction approval method will be determined per EMV 4.x requirements.

For OFFLINE, the device gets the TC or AAC from the ICC for later transmission to the host. Depending on the transaction outcome, the LCD will show APPROVED, DECLINED, or ERROR, and the device will perform the command completion action.

For ONLINE, the device sends the ARQC tags to the host using **OnEMVDataComplete** for approval, starts a HOST response timer, and waits for SendAcquirerResponse from the host, processes the Host Response, gets TC or AAC from the ICC, depending on the transaction outcome, the LCD will show “APPROVED”, “DECLINED” or "ERROR," and perform the command completion action.

A transaction can be forced ONLINE by the merchant by setting the `ForcedOnlineBypassPIN` parameter.

The events associated with this command are **OnCardHolderStateChanged** and **OnEMVDataComplete**.

```
int _stdcall RequestSmartCard(
     int cardType,
     int confirmationTime,
     int pinEnteringTime,
     int beepTones,
     int option,
     char* Amount,
     int transactionType,
     char* cashBack,
     char* rfu);
```

Parameter	Description
cardType	Card type that can be used for the transaction: 1 – Magnetic stripe 2 – Contact smart card 3 – Magnetic stripe or contact smart card 4 – Contactless smart card (not supported on DynaPro Mini) 5 – Contactless smart card + magnetic stripe (not supported on DynaPro Mini) 6 – Contactless smart card + contact smart card (not supported on DynaPro Mini) 7 – Contactless smart card + contact smart card + magnetic stripe (not supported on DynaPro Mini)
comfirmationTime	Time the device will wait for the user to begin the transaction
pinEnteringTime	Time the device will wait for the user to enter the PIN
beepTones	Tone to use: 0 - None 1 - Single Beep 2 - Double Beep
option	Transaction options: 0 – Normal 1 – Bypass PIN 2 – Force Online 4 – Acquirer not available
Amount	The amount to be used and authorized, EMV Tag 9F02, format n12. It should be a 6-byte array.
transactionType	Type of transaction to be used: 0x02 = Cash back 0x04 = Goods 0x08 = Services
cashBack	Amount of cash back to be used, EMV Tag 9F02, format n12. It should be a 6-byte array.
rfu	At least 45 bytes reserved for future use

Return Value:
Returns an `int` value (0: Success, Non-Zero: Error)

3.64 RequestSmartCardWithCardType [Deprecated: V5.01]

This function wraps device command `0xA2`. It directs the device to prompt the user to confirm the transaction amount, and to arm the MSR and / or contact ICC reader to wait for a card to be swiped or presented into the contact ICC connector. If armed to read a contact ICC, the device will turn on the LED near the smart card connector after the cardholder confirms the transaction amount. The host should abort the transaction if the user presses the CANCEL button.

If there are no errors, the device will prompt the user to approve an amount and swipe or insert card by displaying pre-determined EMV messages.

The LCD display will cycle showing "(AMOUNT)," "(AMOUNT) OK?" and "CANCEL OR ENTER," and will wait for the cardholder to push either the confirmation or cancellation button.

If the cardholder presses the confirmation button, then depending on the card type requested to be read, the LCD display will show either SWIPE or INSERT CARD. If the user presses the cancellation button or the transaction times out, the device will perform the command completion action.

If the cardholder has inserted an ICC card, and if the Acquirer has set the device's payment brand account type setting for ICC to **Debit or Credit**, the device will prompt the cardholder to select debit or credit.

Per EMV 4.x requirements, if the cardholder uses the MSR input, the device will check the service code from the magnetic stripe data to see if it begins with a 2 or a 6 to determine if the card also includes an ICC, and will advise the cardholder that ICC is preferred by displaying USE CHIP READER. If the ICC fails or the service code does not begin with a 2 or a 6, the device will prompt the cardholder for an MSR swipe. After a successful swipe, the device will prompt the user to select debit or credit. If this is a debit account type, the device will request a PIN.

If the user presents an ICC card, the LCD display will show ICC applications that are mutually supported and ask the cardholder to choose the preferred application. If a PIN entry is needed per ***EMV 4.x*** requirements, the LCD will show ENTER PIN and start the PIN entry timer. If the user presses the cancelation button or the transaction times out, cancelled or timed out, the device will perform the command completion action.

After PIN entry, the device will display either PIN OK or will cycle through INCORRECT PIN and TRY AGAIN up to the PIN retry limit. If the number of attempts reaches PIN try limit-1, the device will display LAST TRY. If the user exceeds the PIN entry retry limit, the device will perform the command completion action, otherwise the transaction proceeds to the approval stage.

The device can be directed to allow PIN bypass using **RequestBypassPINCommand**. The PIN requirement can also be bypassed by the cardholder.

The transaction approval method will be determined per EMV 4.x requirements.

For OFFLINE, the device gets the TC or AAC from the ICC for later transmission to the host. Depending on the transaction outcome, the LCD will show APPROVED, DECLINED, or ERROR, and the device will perform the command completion action.

For ONLINE, the device sends the ARQC tags to the host using **OnEMVDataComplete** for approval, starts a HOST response timer, and waits for SendAcquirerResponse from the host, processes the Host Response, gets TC or AAC from the ICC, depending on the transaction outcome, the LCD will show "APPROVED", "DECLINED" or "ERROR," and perform the command completion action.

A transaction can be forced ONLINE by the merchant by setting the `ForcedOnlineBypassPIN` parameter.

The events associated with this command are **OnCardHolderStateChanged** and **OnEMVDataComplete**.

```
int _stdcall RequestSmartCardWithCardType(
     int cardType,
     int confirmationTime,
     int pinEnteringTime,
     int beepTones,
     int option,
     char* Amount,
     int transactionType,
     char* cashBack,
     char* rfu);
```

Parameter	Description
cardType	Card type that can be used for the transaction: 1 – Magnetic stripe 2 – Contact smart card 3 – Magnetic stripe or contact smart card
comfirmationTime	Time the device will wait for the user to begin the transaction
pinEnteringTime	Time the device will wait for the user to enter the PIN
beepTones	Tone to use: 0 - None 1 - Single Beep 2 - Double Beep
option	Transaction options: 0 – Normal 1 – Bypass PIN 2 – Force Online 4 – Acquirer not available
Amount	The amount to be used and authorized, EMV Tag 9F02, format n12. It should be a 6-byte array.
transactionType	Type of transaction to be used: 0x02 = Cash back 0x04 = Goods 0x08 = Services
cashBack	Amount of cash back to be used, EMV Tag 9F02, format n12. It should be a 6-byte array.
rfu	At least 45 bytes reserved for future use

Return Value:
Returns an `int` value (0: Success, Non-Zero: Error)

3.65 SendAcquirerResponse

This function sends the Acquirer report to the device. The event associated with this command is **OnEMVTransactionComplete**.

```
int _stdcall SendAcquirerResponse(
```

```
    unsigned char* responseData,
    int responseDataLength);
```

Parameter	Description
responseData	Byte array containing Acquirer Response report data
responseDataLength	responseData length in bytes

Return Value:
Returns an `int` value (0: Success, Non-Zero: Error)

3.66 SendAcquirerResponseWithResponse [Deprecated: V5.01]

This function sends the Acquirer report to the device. The event associated with this command is **OnEMVTransactionComplete**.

```
int _stdcall SendAcquirerResponseWithResponse(
    unsigned char* responseData,
    int responseDataLength);
```

Parameter	Description
responseData	Byte array containing Acquirer Response report data
responseDataLength	responseData length in bytes

Return Value:
Returns an `int` value (0: Success, Non-Zero: Error)

3.67 GetKeyInfo

This function get the injected key information from device.

```
const char* __stdcall GetKeyInfo(int KeyInfoId);
```

Parameter	Description
KeyInfoId	Info Id. Reference to 99875585 DynaPro Programmer's Reference (Commands) – Report 0x0E Get Information.

Return Value:
Hex String. Represented the binary data of report 0x0E Get Information.

3.68 GetAMKInfo

This function get the Acquirer Master Key information.

```
AMKInfo __stdcall GetAMKInfo();
```

Return Value:
An AMKInfo structure.

```
typedef struct AMKInfo_tag
{
      // Key Status or KCV type
      int Status;
      // Hex String for a 24 bit value
      char KCV[17];
      // Acquirer Master Key Label
      char KeyLabel[64];
} AMKInfo;
```

3.69 RequestPowerUpResetICCSync

This is a synchronous version of the **RequestPowerUpResetICCWithWaitTime (EMV L1 only)** function. The function will return after the user inserts a smart card or until `waitTime` has elapsed.

```
int _stdcall RequestPowerUpICCSync(
      int waitTime,
      int operation,
      EMV_DATA* atr,
      int* opStatus);
```

Parameter	Description
waitTime	Time the device will wait for the user to insert a smart card
operation	Operation ID: 1 = Power up or warm reset 2 = Power down
atr	An `EMV_DATA` structure that includes IC card ATR. See below for type definition.
opStatus	Integer pointer to receive the command response or operation status. Zero value means OK. For more values, see **Appendix B**.

```
typedef struct _EMV_DATA
{
      BYTE OpStatus;
      CByte *Data;
      int Length;
}EMV_DATA;
```

Return Value:
Returns an `int` value (0: Success, Non-Zero: Error)

3.70 RequestPowerUpResetICCWithWaitTimeSync [Deprecated: V5.01]

This is a synchronous version of the **RequestPowerUpResetICCWithWaitTime (EMV L1 only)** function. The function will return after the user inserts a smart card or until `waitTime` has elapsed.

```
int _stdcall RequestPowerUpICCWithWaitTimeSync(
      int waitTime,
```

```
      int operation,
      EMV_DATA* atr,
      int* opStatus);
```

Parameter	Description
waitTime	Time the device will wait for the user to insert a smart card
operation	Operation ID: 1 = Power up or warm reset 2 = Power down
atr	An `EMV_DATA` structure that includes IC card ATR. See below for type definition.
opStatus	Integer pointer to receive the command response or operation status. Zero value means OK. For more values, see **Appendix B**.

```
typedef struct _EMV_DATA
{
      BYTE OpStatus;
      CByte *Data;
      int Length;
}EMV_DATA;
```

Return Value:
Returns an `int` value (0: Success, Non-Zero: Error)

3.71 RequestICCAPDUForInformationSync

This is a synchronous version of the **RequestICCAPDUForInformation (EMV L1 only)** function.

```
int _stdcall RequestICCAPDUForInformationSync(
      unsigned char* apdu,
      int apduLen,
      BYTE* pData,
      DWORD* pdwDataLen,
      int* opStatus);
```

Parameter	Description
apdu	Pointer to a byte array of APDU
apduLen	APDU length in bytes
pData	Pointer to a buffer to receive the smart card response
pdwDataLen	Pointer to receive the length of `pData` in bytes after a successful call.
opStatus	Integer pointer to receive the command response or operation status. Zero value means OK. For more values, see **Appendix B**.

Return Value:

Returns an `int` value (0: Success, Non-Zero: Error)

3.72 RequestGetEMVTagsSync

This is a synchronous version of the **RequestGetEMVTags** function.

```
int _stdcall RequestGetEMVTagsSync(
     int tagType,
     int tagOperation,
     unsigned char* inputTLVData,
     int inputDataLength,
     EMV_DATA* emvTags,
     int* opStatus);
```

Parameter	Description
tagType	EMV Tag to set or get: 0x00 – Reader Tags 0x80 – Application Tags Lower 7 bits indicate which application slot of operation
operation	Type of operation to be performed: 0 – Read Operation 0x0F – Read All PIN-PAD or Application Tags
inputTLVData	TLV data block to send to the device
inputDataLength	Length of the TLV data block
pEMVTag_DATA	Pointer to an `EMV_DATA` structure to hold tags. See section **RequestPowerUpResetICCSync** for type definition.
opStatus	Integer pointer to receive the command response or operation status. Zero value means OK. For more values, see **Appendix B**.

Return Value:
Returns an `int` value (0: Success, Non-Zero: Error)

3.73 RequestGetEMVTagsWithTagTypeSync [Deprecated: V5.01]

This is a synchronous version of the **RequestGetEMVTagsWithTagType** function.

```
int _stdcall RequestGetEMVTagsWithTagTypeSync(
     int tagType,
     int tagOperation,
     unsigned char* inputTLVData,
     int inputDataLength,
     EMV_DATA* emvTags,
     int* opStatus);
```

Parameter	Description
tagType	EMV Tag to set or get: 0x00 – Reader Tags 0x80 – Application Tags Lower 7 bits indicate which application slot of operation
operation	Type of operation to be performed: 0 – Read Operation 0x0F – Read All PIN-PAD or Application Tags
inputTLVData	TLV data block to send to the device
inputDataLength	Length of the TLV data block
pEMVTag_DATA	Pointer to an `EMV_DATA` structure to hold tags. See section **3.70 RequestPowerUpResetICCWithWaitTimeSync** for type definition.
opStatus	Integer pointer to receive the command response or operation status. Zero value means OK. For more values, see **Appendix B**.

Return Value:
Returns an `int` value (0: Success, Non-Zero: Error)

3.74 RequestSetEMVTagsSync

This is a synchronous version of the **RequestSetEMVTags** function.

```
int _stdcall RequestSetEMVTagsSync(
     int tagType,
     int tagOperation,
     unsigned char* inputTLVData,
     int inputDataLength,
     EMV_DATA* emvTags,
     int* opStatus);
```

Parameter	Description
tagType	EMV tag to set or get: 0x00 – Reader tags 0x80 – Application tags Lower 7 bits indicate which application slot of operation
tagOperation	Type of operation to be performed: 1 – Write operation 0xFF – Set to factory defaults
inputTLVData	TLV data block to send to the device
inputDataLength	Length of the TLV data block
emvTags	Pointer to an `EMV_DATA` structure to hold tags. See section **RequestPowerUpResetICCSync** for type definition.
opStatus	Integer pointer to receive the command response or operation status. Zero value means OK. For more values, see **Appendix B**.

Return Value:
Returns an `int` value (0: Success, Non-Zero: Error)

3.75 RequestSetEMVTagsWithTagTypeSync [Deprecated: V5.01]

This is a synchronous version of the **RequestSetEMVTagsWithTagType** function.

```
int _stdcall RequestSetEMVTagsWithTagTypeSync(
      int tagType,
      int tagOperation,
      unsigned char* inputTLVData,
      int inputDataLength,
      EMV_DATA* emvTags,
      int* opStatus);
```

Parameter	Description
tagType	EMV tag to set or get: 0x00 – Reader tags 0x80 – Application tags Lower 7 bits indicate which application slot of operation
tagOperation	Type of operation to be performed: 1 – Write operation 0xFF – Set to factory defaults
inputTLVData	TLV data block to send to the device
inputDataLength	Length of the TLV data block
emvTags	Pointer to an `EMV_DATA` structure to hold tags. See section **3.70 RequestPowerUpResetICCWithWaitTimeSync** for type definition.
opStatus	Integer pointer to receive the command response or operation status. Zero value means OK. For more values, see **Appendix B**.

Return Value:
Returns an `int` value (0: Success, Non-Zero: Error)

3.76 SetCAPublicKeySync

This is a synchronous version of the **SetCAPublicKey** function.

```
int _stdcall SetCAPublicKeySync(
      int operation,
      unsigned char* keyBlock,
      int keyBlockLength,
      BYTE* key,
      DWORD* keyLen,
      int* opStatus);
```

Parameter	Description
operation	Type of operation to be performed: 0 – Erase all CA Public Keys 1 – Erase all CA Public Keys for a given RID 2 – Erase a single CA Public Key 3 – Add a single CA Public Key 0x0F – Read all CA Public Keys
keyBlock	Byte array for the CA Public Key to send
keyBlockLength	Length of the CA Public Key
pData	Pointer to receive the key data
pdwDataLen	Buffer length. After successfully reading keys, the function will set it to the key length.
opStatus	Integer pointer to receive the command response or operation status. Zero value means OK. For more values, see **Appendix B**.

Return Value:
Returns an `int` value (0: Success, Non-Zero: Error)

3.77 SetCAPublicKeyWithOperationSync [Deprecated: V5.01]

This is a synchronous version of the **SetCAPublicKeyWithOperation** [Deprecated: V5.01] function.

```
int _stdcall SetCAPublicKeyWithOperationSync(
     int operation,
     unsigned char* keyBlock,
     int keyBlockLength,
     BYTE* key,
     DWORD* keyLen,
     int* opStatus);
```

Parameter	Description
operation	Type of operation to be performed: 0 – Erase all CA Public Keys 1 – Erase all CA Public Keys for a given RID 2 – Erase a single CA Public Key 3 – Add a single CA Public Key 0x0F – Read all CA Public Keys
keyBlock	Byte array for the CA Public Key to send
keyBlockLength	Length of the CA Public Key
pData	Pointer to receive the key data
pdwDataLen	Buffer length. After successfully reading keys, the function will set it to the key length.

opStatus	Integer pointer to receive the command response or operation status. Zero value means OK. For more values, see **Appendix B**.

Return Value:
Returns an `int` value (0: Success, Non-Zero: Error)

3.78 SetDisplayMessageSync

This is a synchronous version of the **SetDisplayMessage** function. The function will return after `waitTime` has elapsed, and the display will be reset to a blank screen.

```
int _stdcall SetDisplayMessageSync(
      int waitTime,
      int messageID,
      int* opStatus);
```

Parameter	Description
waitTime	Length of time the message will be displayed
messageID	Predefined message to be displayed
opStatus	Integer pointer to receive the command response or operation status. Zero value means OK. For more values, see **Appendix B**.

Return Value:
Returns an `int` value (0: Success, Non-Zero: Error)

3.79 RequestCardSync

This is a synchronous version of the **RequestCard** function. The function will return after the user swipes a card or until `waitTime` has elapsed.

```
int _stdcall RequestCardSync(
      int waitTime,
      int displayMessage,
      int beepTones,
      CARD_DATA* card,
      int* opStatus);
```

Parameter	Description
waitTime	Time the device will wait for the user to complete a card swipe
messageID	Message to prompt the user with: 0x00 - CardMsgSwipeCardIdle 0x01 - CardMsgSwipeCard 0x02 - CardMsgPleaseSwipeCard 0x03 - CardMsgPleaseSwipeAgain
beepTones	Tone to use: 0x00 – No Sound 0x01 – Single Beep 0x02 – Double Beeps
card	Pointer to a `CARD_DATA` structure to hold card data. See definition below.
opStatus	Integer pointer to receive the command response or operation status. Zero value means OK. For more values, see **Appendix B**.

```
typedef struct _CARD_DATA
{
      BYTE DataType;
      BYTE CardOperationStatus;
      BYTE CardStatus;
      BYTE CardType;

      BYTE Track1Length;
      BYTE Track2Length;
      BYTE Track3Length;
      BYTE EncTrack1Length;
      BYTE EncTrack2Length;
      BYTE EncTrack3Length;
      BYTE EncMPLength;

      BYTE Track1Status;
      BYTE Track2Status;
      BYTE Track3Status;
      BYTE EncTrack1Status;
      BYTE EncTrack2Status;
      BYTE EncTrack3Status;
```

```
      BYTE EncMPStatus;
      BYTE MSStatus;

      char *MPSTS;
      char *Track1;
      char *Track2;
      char *Track3;
      char *EncTrack1;
      char *EncTrack2;
      char *EncTrack3;
      char *EncMP;
      char *KSN;

      char* CBCMAC;
      char* SerialNumber;
      BYTE  PANInfoLength;
      char* PANInfo;

      unsigned long reserved;
} CARD_DATA;
```

Return Value:
Returns an `int` value (0: Success, Non-Zero: Error)

3.80 RequestManualCardDataSync

This is a synchronous version of the **RequestManualCardDataWithWaitTime** function. The function will return after the user inputs card data or until `waitTime` has elapsed.

```
int _stdcall RequestManualCardDataSync(
      int waitTime,
      int beepTones,
      int options,
      CARD_DATA* card,
      int* opStatus);
```

Parameter	Description
waitTime	Time the device will wait for user to begin manual data entry
beepTones	Tone to use: 0 - None 1 - Single Beep 2 - Double Beep

option	This is an ORed combination of flags that changes the device's data entry request behavior as follows: Bits 0 and 1 0 = Acct,Date,CVC 1 = Acct,Date 2 = Acct,CVC 3 = Acct Bit 2 1=Use Qwick Codes entry Bit 3 1=Use PAN in PIN block creation Bit 4 0=Use PAN min 9, max 19 1=Use PAN min 14, max 21 Bits 5-7 are reserved and should be set to 0.
card	Pointer to a `CARD_DATA` structure to hold card data (see section **3.79 RequestCardSync** for type definition).
opStatus	Integer pointer to receive the command response or operation status. Zero value means OK. For more values, see **Appendix B**.

Return Value:
Returns an `int` value (0: Success, Non-Zero: Error)

3.81 RequestUserDataEntrySync

This is a synchronous version of the **RequestUserDataEntryWithWaitTime** function. The function will return after the user inputs data or until `waitTime` has elapsed.

```
int _stdcall RequestUserDataEntrySync(
     int waitTime,
     int displayMessageID,
     int beepTones,
     USER_ENTRY_DATA* userData,
     int* opStatus);
```

Parameter	Description
waitTime	Time the device will wait for the user to begin data entry
displayMessageID	Message to prompt the user with: 0 – SSN 1 – Zip code 2 – Birth (four-digit year) 3 – Birth (two-digit year)

beepTones	Tone to use: 0 - None 1 - Single Beep 2 - Double Beep
userData	Pointer to a `USER_ENTRY_DATA` structure to hold card data. See below for type definition.
opStatus	Integer pointer to receive the command response or operation status. Zero value means OK. For more values, see **Appendix B**.

```
typedef struct _USER_ENTRY_DATA
{
      BYTE OpStatus;   //Operation Status
      char * MSRKsn;
      char * EDB;
} USER_ENTRY_DATA;
```

Return Value:
Returns an `int` value (0: Success, Non-Zero: Error)

3.82 RequestClearTextUserDataEntrySync

This is a synchronous version of the **RequestClearTextUserDataEntryWithWaitTime** function. The function will return after the user inputs data or until `waitTime` has elapsed for devices that support this feature.

```
int _stdcall RequestClearTextUserDataEntrySync(
      int waitTime,
      int displayMessageID,
      int beepTones,
      CLEAR_TEXT_USER_ENTRY_DATA* userData,
      int* opStatus);
```

Parameter	Description
waitTime	Time the device will wait for the user to begin data entry
displayMessageID	Message to prompt the user with: 0 – SSN, 1 – Zip Code 2 – Birth (four-digit year) 3 – Birth (two-digit year)
beepTones	Tone to use: 0 - None 1 - Single Beep 2 - Double Beep
userData	Pointer to a `CLEAR_TEXT_USER_ENTRY_DATA` structure to hold card data (see below for type definition).

opStatus	Integer pointer to receive the command response or operation status. Zero value means OK. For more values, see **Appendix B**.

```
typedef struct _CLEAR_TEXT_USER_ENTRY_DATA
{
      BYTE OpStatus;   //Operation Status
      BYTE UserDataMode;
      BYTE DataLen;
      char *Data;

} CLEAR_TEXT_USER_ENTRY_DATA;
```

Return Value:
Returns an `int` value (0: Success, Non-Zero: Error)

3.83 RequestReponseSync

This is a synchronous version of the **RequestResponse** function. The function will return after the user presses a key or until `waitTime` has elapsed.

```
int  stdcall RequestResponseSync(
      int waitTime,
      int selectMsg,
      int keyMask,
      int beepTones,
      unsigned char* key,
      int* opStatus);
```

Parameter	Description
waitTime	Time the device will wait for the user to respond
selectMsg	Message to prompt the user with: 0 – Transaction type (Credit/Debit) 1 – Verify transaction amount 2 – Credit Other Debit 3 – Credit EBT Debit 4 – Credit Gift Debit 5 – EBT Gift Other 255 – User Defined Message
keyMask	Key codes to mask (combine using OR): 1 – Left 2 – Middle 4 – Right 8 – Enter
beepTones	Tone to use: 0 - None, 1 - Single Beep 2 - Double Beep

key	Pointer to a byte to receive the key value.
opStatus	Integer pointer to receive the command response or operation status. Zero value means OK. For more values, see **Appendix B**.

Return Value:
Returns an `int` value (0: Success, Non-Zero: Error)

3.84 ConfirmAmountSync

This is a synchronous version of the **ConfirmAmount** function. The function will return after the user presses key or until `waitTime` has elapsed.

```
int _stdcall ConfirmAmountSync(
      int waitTime,
      int beepTones,
      unsigned char* key,
      int* opStatus);
```

Parameter	Description
waitTime	Time the device will wait for the user to confirm the amount
beepTones	Tone to use: 0 - None 1 - Single Beep 2 - Double Beep
key	Pointer to a byte to receive the key value.
opStatus	Integer pointer to receive the command response or operation status. Zero value means OK. For more values, see **Appendix B**.

Return Value:
Returns an `int` value (0: Success, Non-Zero: Error)

3.85 SelectCreditDebitSync

This is a synchronous version of the **SelectCreditDebit** function. The function will return after the user presses a key or until `waitTime` has elapsed.

```
int _stdcall SelectCreditDebitSync(
      int waitTime,
      int beepTones,
      unsigned char* key,
      int* opStatus);
```

Parameter	Description
waitTime	Time the device will wait for the user to select Credit or Debit

beepTones	Tone to use: 0 - None 1 - Single Beep 2 - Double Beep
key	Pointer to a byte to receive the key value
opStatus	Integer pointer to receive the command response or operation status. Zero value means OK. For more values, see **Appendix B**.

Return Value:
Returns an `int` value (0: Success, Non-Zero: Error)

3.86 RequestPINSync

This is a synchronous version of the **RequestPIN** command. The function will return after the user enters a PIN or until `waitTime` has elapsed.

```
int _stdcall RequestPINSync(
     int waitTime,
     int pinMode,
     int minPINLength,
     int maxPINLength,
     int beepTones,
     int option,
     PIN_DATA* pin,
     int* opStatus);
```

Parameter	Description
waitTime	Time the device should wait for the user to begin PIN entry
pinMode	Message to display as a user prompt: 0 = PINsgEnterPIN 1 = PINMsgEnterPINAmt 2 = PINMsgReenterPINAmt 3 = PINMsgReenterPIN 4 = PINMsgVerifyPIN
minPINLength	Minimum PIN length. Must be greater than 3.
maxPINLength	Maximum PIN length. Must be less than 13.
beepTones	Tone to use: 0 = No sound 1 = Single beep 2 = Double beep
option	PIN verification and format: 0 = ISO0 Format, No verify PIN 1 = ISO3 Format, No verify PIN 2 = ISO0 Format, Verify PIN 3 = ISO3 Format, Verify PIN

pin	A pointer to `PIN_DATA` structure include secured pin information. See below for type definition.
opStatus	Integer pointer to receive the command response or operation status. Zero value means OK. For more values, see **Appendix B**.

```
typedef struct _PIN_DATA
{
      BYTE OpStatus;
      char *KSN;
      char * EPB;
} PIN_DATA;
```

Return Value:
Returns an `int` value (0: Success, Non-Zero: Error)

3.87 RequestSignatureSync

This is a synchronous version of the **RequestSignature** function. The function will return after the user selects CANCEL or OK, or until `waitTime` has elapsed.

```
int _stdcall RequestSignatureSync(
      int waitTime,
      int beepTones,
      int option,
      SIG_DATA* sig,
      int* opStatus);
```

Parameter	Description
waitTime	Time the device will wait for the user to sign and press CANCEL or OK.
beepTones	Tone to use: 0 - None 1 - Single Beep 2 - Double Beep
option	PIN verification option: 0 – Timeout to clean data 1 – Timeout with available data, signature can retrieved if exists
sig	A `SIG_DATA` structure to hold signature data. See type definition below. Software should `alloc` buffer space before calling this function.
opStatus	Integer pointer to receive the command response or operation status. Zero value means OK. For more values, see **Appendix B**.

```
typedef struct _SIG_DATA
{
      BYTE *Data;
      int Sig_Length;
}SIG_DATA;
```

Return Value:
Returns an `int` value (0: Success, Non-Zero: Error)

3.88 RequestSmartCardSync

This is a synchronous version of the **RequestSmartCard** function.

```
int _stdcall RequestSmartCardSync(
     int cardType,
     int confirmationTime,
     int pinEnteringTime,
     int beepTones,
     int option,
     char* Amount,
     int transactionType,
     char* cashBack,
     char* rfu,
     ACQUIRER_DATA* arqcTag,
     int* opStatus);
```

Parameter	Description
cardType	Card type that can be used for the transaction: 1 – Magnetic stripe 2 – Contact smart card 3 – Magnetic stripe or contact smart card 4 – Contactless smart card (not supported on DynaPro Mini) 5 – Contactless smart card + magnetic stripe (not supported on DynaPro Mini) 6 – Contactless smart card + contact smart card (not supported on DynaPro Mini) 7 – Contactless smart card + contact smart card + magnetic stripe (not supported on DynaPro Mini)
comfirmationTime	Time the device will wait for the user to begin the transaction
pinEnteringTime	Time the device will wait for the user to enter the PIN
beepTones	Tone to use: 0 - None 1 - Single Beep 2 - Double Beep
option	Transaction options: 0 – Normal 1 – Bypass PIN 2 – Force Online 4 – Acquirer not available
Amount	The amount to be used and authorized, EMV Tag 9F02, format n12. It should be a 6-byte array.

transactionType	Type of transaction to be used: 0x02 = Cash back 0x04 = Goods 0x08 = Services
cashBack	Amount of cash back to be used, EMV Tag 9F02, format n12. It should be a 6-byte array.
rfu	Reserved bytes
arqcTag	Pointer to an `ACQUIRER_DATA` structure to holdARQC tags. See type definition below.
opStatus	Integer pointer to receive the command response or operation status. Zero value means OK. For more values, see **Appendix B**.

```
typedef struct _ACQUIRER_DATA
{
      BYTE OpStatus;
      CByte *Data;
      int Length;
}ACQUIRER_DATA;
```

Return Value:
Returns an `int` value (0: Success, Non-Zero: Error)

3.89 RequestSmartCardWithCardTypeSync [Deprecated: V5.01]

This is a synchronous version of the **RequestSmartCard** function.

```
int _stdcall RequestSmartCardWithCardTypeSync(
      int cardType,
      int confirmationTime,
      int pinEnteringTime,
      int beepTones,
      int option,
      char* Amount,
      int transactionType,
      char* cashBack,
      char* rfu,
      ACQUIRER_DATA* arqcTag,
      int* opStatus);
```

Parameter	Description
cardType	Card type that can be used for the transaction: 1 – Magnetic stripe 2 – Contact smart card 3 – Magnetic stripe or contact smart card
comfirmationTime	Time the device will wait for the user to begin the transaction
pinEnteringTime	Time the device will wait for the user to enter the PIN

beepTones	Tone to use: 0 - None 1 - Single Beep 2 - Double Beep
option	Transaction options: 0 – Normal 1 – Bypass PIN 2 – Force Online 4 – Acquirer not available
Amount	The amount to be used and authorized, EMV Tag 9F02, format n12. It should be a 6-byte array.
transactionType	Type of transaction to be used: 0x02 = Cash back 0x04 = Goods 0x08 = Services
cashBack	Amount of cash back to be used, EMV Tag 9F02, format n12. It should be a 6-byte array.
rfu	Reserved bytes
arqcTag	Pointer to an `ACQUIRER_DATA` structure to holdARQC tags. See type definition below.
opStatus	Integer pointer to receive the command response or operation status. Zero value means OK. For more values, see **Appendix B**.

```
typedef struct _ACQUIRER_DATA
{
      BYTE OpStatus;
      CByte *Data;
      int Length;
}ACQUIRER_DATA;
```

Return Value:
Returns an `int` value (0: Success, Non-Zero: Error)

3.90 SendAcquirerResponseSync

This is a synchronous version of the **SendAcquirerResponseWithResponse** function.

```
int __stdcall SendAcquirerResponseSync(
      unsigned char* responseData,
      int responseDataLength,
      EMV_DATA* merchantData,
      int* opStatus);
```

Parameter	Description
responseData	Byte array to contain the Acquirer Response data

responseDataLength	Response data length in bytes
merchantData	Pointer to an `EMV_DATA` structure to hold merchant data. See section **3.70 RequestPowerUpResetICCWithWaitTimeSync** for type definition.
opStatus	Integer pointer to receive the command response or operation status. Zero value means OK. For more values, see **Appendix B**.

Return Value:
Returns an `int` value (0: Success, Non-Zero: Error)

3.91 SendAcquirerResponseWithResponseSync [Deprecated: V5.01]

This is a synchronous version of the **SendAcquirerResponseWithResponse** function.

```
int _stdcall SendAcquirerResponseWithResponseSync(
      unsigned char* responseData,
      int responseDataLength,
      EMV_DATA* merchantData,
      int* opStatus);
```

Parameter	Description
responseData	Byte array to contain the Acquirer Response data
responseDataLength	Response data length in bytes
merchantData	Pointer to an `EMV_DATA` structure to hold merchant data. See section **3.70 RequestPowerUpResetICCWithWaitTimeSync** for type definition.
opStatus	Integer pointer to receive the command response or operation status. Zero value means OK. For more values, see **Appendix B**.

Return Value:
Returns an `int` value (0: Success, Non-Zero: Error)

3.92 GetProductID

This function returns the product ID of the device.

```
const char* __stdcall GetProductID();
```

Return Value:
Returns an pointer of product ID.

3.93 GetDeviceSerial

This function returns the serial number of the device.

```
const char* __stdcall GetDeviceSerial();
```

Return Value:
Returns an pointer of device serial number.

3.94 GetDeviceModel

This function returns the model number of the device.

```
const char* __stdcall GetDeviceModel();
```

Return Value:
Returns an pointer of device model number.

3.95 GetDeviceFirmwareVersion

This function returns the firmware revision of the device.

```
const char* __stdcall GetDeviceFirmwareVersion();
```

Return Value:
Returns an pointer of device firmware revision.

3.96 GetPINKSN

This function returns the PIN Key Serial Number (KSN).

```
const char* __stdcall GetPINKSN();
```

Return Value:
Returns an pointer of PIN Key Serial Number.

3.97 GetPINStatusCode

This function returns the status of PIN request.

```
const char* __stdcall GetPINStatusCode();
```

Return Value:
Returns an pointer of hex string. (“0” for Success).

3.98 GetPINData

This function returns the PIN data structure.

```
const PIN_DATA* __stdcall GetPINData();
```

Return Value:
Returns an pointer of PIN_DATA structure containing PIN data.

3.99 GetEPB

This function returns the encrypted PIN block (EPB) data.

```
const char* __stdcall GetPINKSN();
```

Return Value:
Returns an pointer of hex string containing encrypted PIN block data.

3.100 GetCardDataInfo

This function returns the card data.

```
const CARD_DATA* __stdcall GetCardDataInfo();
```

Return Value:
Returns an pointer of CARD_DATA structure containing detailed information of the card.

3.101 GetPAN

This function returns the PAN of the transaction.

```
const char* __stdcall GetPAN();
```

Return Value:
Returns an pointer PAN of the transaction.

3.102 GetEncodeType

This function returns the card type.

```
const CARD_DATA* __stdcall GetEncodeType();
```

Return Value:
Returns an hex string indicating the type of card.

- "0" = Other
- "1" = Financial
- "2" = AAMVA
- "3" = Manual
- "4" = Unknown
- "5" = ICC
- "6" = Contactless ICC
- "7" = Financial ICC

GetTrack1
This function returns the encrypted track 1 data.

```
const char* __stdcall GetTrack1();
```

Return Value:
Returns an pointer of the encrypted track 1 data.

3.103 GetTrack2

This function returns the encrypted track 2 data.

```
const char* __stdcall GetTrack2();
```

Return Value:

Returns an pointer of the encrypted track 2 data.

3.104 GetTrack3

This function returns the encrypted track 3 data.

```
const char* __stdcall GetTrack3();
```

Return Value:
Returns an pointer of the encrypted track 3 data.

3.105 GetTrack1Masked

This function returns the masked track 1 data.

```
const char* __stdcall GetTrack1Masked();
```

Return Value:
Returns an pointer of the masked track 1 data.

3.106 GetTrack2Masked

This function returns the masked track 2 data.

```
const char* __stdcall GetTrack2Masked();
```

Return Value:
Returns an pointer of the masked track 2 data.

3.107 GetTrack3Masked

This function returns the masked track 3 data.

```
const char* __stdcall GetTrack3Masked();
```

Return Value:
Returns an pointer of the masked track 3 data.

3.108 GetMaskedTracks

This function returns the data of all masked tracks.

```
const char* __stdcall GetMaskedTracks();
```

Return Value:
Returns an pointer of the string of all masked tracks.

3.109 GetMagnePrint

This function returns the MagnePrint data.

```
const char* __stdcall GetMagnePrint();
```

Return Value:

Returns an pointer of the MagnePrint data.

3.110 GetMagnePrintStatus

This function returns the status of the MagnePrint data.

```
const char* __stdcall GetMagnePrintStatus();
```

Return Value:
Returns an pointer of status of the MangePrint data.

3.111 GetTrack1DecodeStatus

This function returns the decode status of track 1 data.

```
const char* __stdcall GetTrack1DecodeStatus();
```

Return Value:
Returns an pointer of the decode status value of track 1 data (“0” for Success).

3.112 GetTrack2DecodeStatus

This function returns the decode status of track 2 data.

```
const char* __stdcall GetTrack2DecodeStatus();
```

Return Value:
Returns an pointer of the decode status value of track 2 data (“0” for Success).

3.113 GetTrack3DecodeStatus

This function returns the decode status of track 3 data.

```
const char* __stdcall GetTrack3DecodeStatus();
```

Return Value:
Returns an pointer of the decode status value of track 3 data (“0” for Success).

3.114 GetLastName

This function returns the last name.

```
const char* __stdcall GetLastName();
```

Return Value:
Returns an pointer of the last name.

3.115 GetFirstName

This function returns the first name.

```
const char* __stdcall GetFirstName();
```

Return Value:

Returns an pointer of the first name.

3.116 GetMiddleName

This function returns the middle name.

```
const char* __stdcall GetMiddleName();
```

Return Value:
Returns an pointer of the middle name.

3.117 GetExpDate

This function returns the expiration date.

```
const char* __stdcall GetExpDate();
```

Return Value:
Returns an pointer of the expiration date.

3.118 ClearBuffer

This function clears all the transaction data cached in the library.

```
void __stdcall ClearBuffer();
```

Return Value:
None.

3.119 LoadClientCertificate

This function loads a TLS client certificate into the SDK for the purpose of mutual authentication.

```
int LoadClientCertificate(
      char* format,
      unsigned char* data,
      int dataLength,
      char* password,
      int passwordLength);
```

Parameter	Description
format	Format of the certificate file. Use: "PKCS12"
data	Pointer to the data for the certificate or certificate chain. Data format is PKCS12.
dataLength	Length of the data array.
password	Pointer to the password for the data.
passwordLength	Length of the password.

Return Value:
Returns an int value (0: Success, Non-Zero: Error)

3.120 RequestTipOrCashback

This method directs the device to request tip or cashback information from user.

```
int __stdcall RequestTipOrCashback(
      int waitTime,
      int mode,
      int tone,
      unsigned char* amount,
      unsigned char* taxAmount,
      unsigned char* taxRate,
      int tipMode,
      int option1,
      int option2,
      int option3,
      unsigned char* reserved);
```

Parameter	Description
waitTime	Wait time in seconds for user to enter the information.
mode	Values: `0x00`= Tip Mode `0x01`= Cashback Mode

tone	Tone to be used:
	`0x00=No Sound`
	`0x01=One Beep`
	`0x02=Two Beeps`

amount	Byte array value of the transaction amount (6 bytes).
tax	Byte array value of the calculated tax amount (6 bytes).
taxRate	Byte array value of the tax rate percentage (3 bytes).
tipMode	Values: `0x00`= Percent Mode `0x01`= Amount Mode
option1	Byte value of the percent or amount shown on the left side of the display above the selection buttons.
option2	Byte value of the percent or amount shown in the middle of the display above the selection buttons.
option3	Byte value of the percent or amount shown on the right side of the display above the selection buttons.
reserved	Reserved bytes.

Return Value:
Returns an `int` value (0: Success, Non-Zero: Error)

3.121 GetSelectedMenuItem

This method directs the device to present a list of menu items for user to select from.

```
int __stdCall GetSelectedMenuItem(
     int waitTime,
     int mode,
     int tone,
     unsigned char* data,
     int dataLength);
```

Parameter	Description
waitTime	Wait time in seconds for user to enter the information.
mode	Values: `0x00`= Selection Table `0x01`= Selection Bill

tone	Tone to be used:
	`0x00=No Sound`
	`0x01=One Beep`
	`0x02=Two Beeps`

data	Menu selection list to send to the device. Each line item must be delimited by a carriage return. Example of sending 3 lines: “Table1<CR>Table2<CR>Table3<CR><CR>”
dataLength	Data length

Return Value:
Returns an `int` value (0: Success, Non-Zero: Error)

4 MTPPSCRA Callback Functions

After calling the functions in section 3 **MTPPSCRA Library Functions**, the MTPPSCRA SDK libraries will invoke callback functions to provide the requested data and/or a detailed response. Custom software must define callback functions that match the `typedef`s provided in the following sections, then register them using the corresponding On* functions. For sample code that demonstrates how to define and register callback functions, see the sample code included with the SDK files, and **Appendix A Code Examples**.

4.1 OnError

Call this function to register an event handler for error conditions.

```
void _stdcall OnError(
     OnErrorEvent lpFuncNotify,
     void *object = NULL);
```

Parameter	Description
lpFuncNotify	Pointer to a custom `OnErrorEvent` callback function (see `typedef` below)
object	Pointer to forward to the `OnErrorEvent` type function when the event is triggered. For example, can be used to pass `this` to preserve the calling context.

```
typedef void ( _stdcall * OnErrorEvent)(
     int errorCode,
     void* param);
```

Parameter	Description
errorCode	A string value containing the error description
param	Pointer forwarded by `OnError`

4.2 OnDataReady

Call this function to register a return event handler for **SendSpecialCommandWithCommand**.

```
void _stdcall OnDataReady(
     OnDataReadyEvent lpFuncNotify,
     void *object = NULL);
```

Parameter	Description
lpFuncNotify	Pointer to a custom `OnDataReadyCompleteEvent` callback function (see `typedef` below)
object	Pointer to forward to the `OnDataReadyEvent` type function when the event is triggered. For example, can be used to pass `this` to preserve the calling context.

```
typedef void (_stdcall * OnDataReadyEvent)(
     BYTE * lpResult,
     int dwResultLen,
     void *object);
```

Parameter	Description
lpResult	A byte array for the raw data.
dwResultLen	Length of data in bytes
object	Pointer forwarded by `OnDataReady`

4.3 OnPowerUpICC

Call this function to register a return event handler for **RequestPowerUpResetICC (EMV L1 only)**.

```
void _stdcall OnPowerUpICC(
     OnPowerUpICCEvent lpFuncNotify,
     void *object = NULL);
```

Parameter	Description
lpFuncNotify	Pointer to a custom `OnPowerUpICCEvent` callback function (see `typedef` below)
object	Pointer to forward to the `OnPowerUpICCEvent` type function when the event is triggered. For example, can be used to pass `this` to preserve the calling context.

```
typedef void (_stdcall * OnPowerUpICCEvent)(
     BYTE opStatus,
     void * pData,
     void *object);
```

Parameter	Description
opStatus	Byte value indicates device operation status. Zero value means OK. For more values, see **Appendix B**.
pData	A pointer to structure `EMV_DATA`, ATR should saved in encrypted. See section **3.70 RequestPowerUpResetICCWithWaitTimeSync** for type definition.
object	Pointer forwarded by `OnPowerUpICC`

4.4 OnAPDUArrived

Call this function to register a return event handler for **RequestICCAPDUForInformation (EMV L1 only)**.

```
void _stdcall OnAPDUArrived(
```

```
      OnAPDUArrivedEvent lpFuncNotify,
      void *object = NULL);
```

Parameter	Description
lpFuncNotify	Pointer to a custom `OnAPDUArrivedCompleteEvent` callback function (see `typedef` below)
object	Pointer to forward to the `OnAPDUArrivedCompleteEvent` type function when the event is triggered. For example, can be used to pass `this` to preserve the calling context.

```
typedef void (_stdcall * OnAPDUArrivedEvent)(
      BYTE opStatus,
      BYTE* pData,
      DWORD pdwDataLen,
      void *object);
```

Parameter	Description
opStatus	Byte value indicates device operation status. Zero value means OK. For more values, see **Appendix B**.
pData	Pointer to receive encrypted APDU.
pdwDataLen	Encrypted APDU length
object	Pointer forwarded by `OnAPDUArrived`

4.5 OnGetCAPublicKey

Call this function to register a return event handler for the `0x0F` or `0x04` operations of **SetCAPublicKeyWithOperation** [Deprecated: V5.01]

.

```
void _stdcall OnGetCAPublicKey(
      OnGetCAPublicKeyEvent lpFuncNotify,
      void *object = NULL);
```

Parameter	Description
lpFuncNotify	Pointer to a custom `OnGetCAPublicKeyCompleteEvent` callback function (see `typedef` below)
object	Pointer to forward to the `OnGetCAPublicKeyCompleteEvent` type function when the event is triggered. For example, can be used to pass `this` to preserve the calling context.

```
typedef void (_stdcall * OnGetCAPublicKeyEvent)(
      BYTE opStatus,
      BYTE* pData,
      DWORD pdwDataLen,
```

```
     void *object);
```

Parameter	Description
opStatus	Byte value indicates device operation status. Zero value means OK. For more values, see **Appendix B**.
pData	A pointer to received key.
pdwDataLen	Key length
object	Pointer forwarded by `OnGetCAPublicKey`

4.6 OnEMVTagsComplete

Call this function to register a return event handler for **RequestGetEMVTagsWithTagType**.

```
void _stdcall OnEMVTagsComplete(
     OnEMVTagsCompletedEvent lpFuncNotify,
     void *object = NULL);
```

Parameter	Description
lpFuncNotify	Pointer to a custom `OnEMVTagsCompleteEvent` callback function (see `typedef` below)
object	Pointer to forward to the `OnEMVTagsCompleteEvent` type function when the event is triggered. For example, can be used to pass `this` to preserve the calling context.

```
typedef void (_stdcall * OnEMVTagsCompleteEvent)(
     BYTE opStatus,
     void* pEMVTag_DATA,
     void *object);
```

Parameter	Description
opStatus	Byte value indicates device operation status. Zero value means OK. For more values, see **Appendix B**.
pEMVTag_DATA	Pointer to a `EMV_DATA` structure to hold tags. See section **3.70 RequestPowerUpResetICCWithWaitTimeSync** for type definition.
object	Pointer forwarded by `OnEMVTagsComplete`

4.7 OnPINRequestComplete

Call this function to register a return event handler for **RequestPIN**.

```
void _stdcall OnPINRequestComplete(
     OnPINRequestCompleteEvent PINEvent,
     void *object = NULL);
```

Parameter	Description
lpFuncNotify	Pointer to a custom `OnPINRequestCompleteEvent` callback function (see `typedef` below)
object	Pointer to forward to the `OnPINRequestCompleteEvent` type function when the event is triggered. For example, can be used to pass `this` to preserve the calling context.

```
typedef void (_stdcall * OnPINRequestCompleteEvent) (
     BYTE opStatus,
     void* pPIN_DATA,
     void* object);
```

Parameter	Description
opStatus	Byte value indicates device operation status. Zero value means OK. For more values, see **Appendix B**.
pData	A pointer to `PIN_DATA` structure. See section **3.86 RequestPINSync** for type definition.
object	Pointer forwarded by `OnPINRequestComplete`

4.8 OnKeyInput

Call this function to register a return event handler for **RequestResponse**, **ConfirmAmount**, and **SelectCreditDebit**.

```
void _stdcall OnKeyInput(
     OnKeyInputEvent GetKeyEvent,
     void *object = NULL);
```

Parameter	Description
lpFuncNotify	Pointer to a custom `OnKeyInputCompleteEvent` callback function (see `typedef` below)
object	Pointer to forward to the `OnKeyInputCompleteEvent` type function when the event is triggered. For example, can be used to pass `this` to preserve the calling context.

```
typedef void (_stdcall * OnKeyInputEvent) (
     BYTE opStatus,
     BYTE keyPressed ,
     void* param);
```

Parameter	Description
opStatus	Byte value indicates device operation status. Zero value means OK. For more values, see **Appendix B**.

keyPressed	Value of pressed key.
object	Pointer forwarded by `OnKeyInput`

4.9 OnDisplayRequestComplete

Call this function to register a return event handler for **SetDisplayMessage**.

```
void _stdcall OnDisplayRequest(
     OnDisplayRequestCompleteEvent lpFuncNotify,
     void *object = NULL);
```

Parameter	Description
lpFuncNotify	Pointer to a custom `OnDisplayRequestCompleteEvent` callback function (see `typedef` below)
object	Pointer to forward to the `OnDisplayRequestCompleteEvent` type function when the event is triggered. For example, can be used to pass `this` to preserve the calling context.

```
typedef void (_stdcall * OnDisplayRequestCompleteEvent)(
     BYTE opStatus,
     void *object);
```

Parameter	Description
opStatus	Byte value indicates device operation status. Zero value means OK. For more values, see **Appendix B**.
object	Pointer forwarded by `OnDisplayRequest`

4.10 OnSignatureArrived

Call this function to register a return event handler for **RequestSignature**.

```
void  _stdcall OnSignatureArrived(
     OnSignatureArrivedEvent DataEvent,
     void *object = NULL);
```

Parameter	Description
lpFuncNotify	Pointer to a custom `OnSignatureArrivedCompleteEvent` callback function (see `typedef` below)
object	Pointer to forward to the `OnSignatureArrivedCompleteEvent` type function when the event is triggered. For example, can be used to pass `this` to preserve the calling context.

```
typedef void (_stdcall * OnSignatureArrivedEvent) (
     BYTE opStatus,
     void* pSIG_DATA,
```

```
      void* object);
```

Parameter	Description
opStatus	Byte value indicates device operation status. Zero value means OK. For more values, see **Appendix B**.
pSIG_DATA	Pointer to a `SIG_DATA` structure to hold signature data. See section **3.87 RequestSignatureSync** for type definition.
object	Pointer forwarded by `OnSignatureArrived`

4.11 OnCardRequestComplete

Call this function to register a return event handler for **RequestCard** and **RequestManualCardDataWithWaitTime**.

```
void _stdcall OnCardRequestComplete(
      OnCardRequestCompleteEvent CardEvent,
      void *object = NULL);
```

Parameter	Description
lpFuncNotify	Pointer to a custom `OnCardRequestCompleteEvent` callback function (see `typedef` below)
object	Pointer to forward to the `OnCardRequestCompleteEvent` type function when the event is triggered. For example, can be used to pass `this` to preserve the calling context.

```
typedef void (_stdcall * OnCardRequestCompleteEvent)(
      BYTE opStatus,
      void* pCARD_DATA,
      void* param);
```

Parameter	Description
opStatus	Byte value indicates device operation status. Zero value means OK. For more values, see **Appendix B**.
pCARD_DATA	Pointer to a `CARD_DATA` structure stored card data (see section **3.79 RequestCardSync** for type definition).
object	Pointer forwarded by `OnCardRequest`

4.12 OnUserDataEntry

Call this function to register a return event handler for **RequestUserDataEntryWithWaitTime**.

```
void _stdcall OnUserDataEntry(
      OnUserDataEntryEvent UserDataEntryEvent,
      void *object = NULL);
```

Parameter	Description
lpFuncNotify	Pointer to a custom `OnUserDataEntryEvent` callback function (see `typedef` below)
object	Pointer to forward to the `OnUserDataEntryEvent` type function when the event is triggered. For example, can be used to pass `this` to preserve the calling context.

```
typedef void (_stdcall * OnUserDataEntryEvent) (
     BYTE opStatus,
     void* pUSERDATA,
     void* param);
```

Parameter	Description
opStatus	Byte value indicates device operation status. Zero value means OK. For more values, see **Appendix B**.
pUSERDATA	Pointer to a `USER_ENTRY_DATA` structure stored card data (see section **3.81 RequestUserDataEntrySync** for type definition).
object	Pointer forwarded by `OnUserDataEntry`

4.13 OnClearTextUserDataEntry

Call this function to register a return event handler for **RequestClearTextUserDataEntryWithWaitTime**. This is only for supported devices.

```
void _stdcall OnClearTextUserDataEntry(
     OnClearTextUserDataEntryEvent lpFuncNotify,
     void *object = NULL);
```

Parameter	Description
lpFuncNotify	Pointer to a custom `OnClearTextUserDataEntryEvent` callback function (see `typedef` below)
object	Pointer to forward to the `OnClearTextUserDataEntryEvent` type function when the event is triggered. For example, can be used to pass `this` to preserve the calling context.

```
typedef void (_stdcall * OnClearTextUserDataEntryEvent)(
     BYTE opStatus,
     void* pClearTextUserEntryData,
     void *object);
```

Parameter	Description
opStatus	Byte value indicates device operation status. Zero value means OK. For more values, see **Appendix B**.
pClearTextUserEntryData	Pointer to a `CLEAR_TEXT_USER_ENTRY_DATA` structure stored card data. See section **3.82 RequestClearTextUserDataEntrySync** for type definition.
object	Pointer forwarded by `OnClearTextUserDataEntry`

4.14 OnDeviceStateUpdated

Call this function to register a return event handler for **RequestDeviceStatusForInformation**.

```
void _stdcall OnDeviceStateUpdated(
     OnDeviceStateUpdatedEvent DeviceStateEvent,
     void *object = NULL);
```

Parameter	Description
DeviceStateEvent	Pointer to a custom `OnDeviceStateUpdatedEvent` callback function (see `typedef` below)
object	Pointer to forward to the `OnDeviceStateUpdatedEvent` type function when the event is triggered. For example, can be used to pass `this` to preserve the calling context.

```
typedef void (_stdcall * OnDeviceStateUpdatedEvent) (
     DEV_STATE_STAT* devState,
     void* param);
```

Parameter	Description
devState	A pointer to `DEV_STATE_STAT` structure to hold device states. See section **3.45 RequestDeviceStatusForInformation** for type definition.
object	Pointer forwarded by `OnDeviceStateUpdated`

4.15 OnEMVDataComplete

Call this function to register a return event handler for **RequestSmartCard**.

```
void _stdcall OnEMVDataComplete(
     OnEMVDataCompleteEvent lpFuncNotify,
     void *object = NULL);
```

Parameter	Description
lpFuncNotify	Pointer to a custom `OnEMVDataCompleteEvent` callback function (see `typedef` below)

object	Pointer to forward to the `OnEMVDataCompleteEvent` type function when the event is triggered. For example, can be used to pass `this` to preserve the calling context.

```
typedef void (_stdcall * OnEMVDataCompleteEvent)(
     BYTE opStatus,
     void* pACQUIRER_DATA,
     void *object);
```

Parameter	Description
opStatus	Byte value indicates device operation status. Zero value means OK. For more values, see **Appendix B**.
pACQUIRER_DATA	Pointer to a ACQUIRER_DATA structure storing ARQC tags.
object	Pointer forwarded by `OnEMVDataComplete`

4.16 OnEMVTransactionComplete

Call this function to register a return event handler for **SendAcquirerResponseWithResponse**.

```
void _stdcall OnEMVTransactionComplete(
     OnEMVTransactionCompleteEvent lpFuncNotify,
     void *object = NULL);
```

Parameter	Description
lpFuncNotify	Pointer to a custom `OnEMVTransactionCompleteEvent` callback function (see `typedef` below)
object	Pointer to forward to the `OnEMVTransactionCompleteEvent` type function when the event is triggered. For example, can be used to pass `this` to preserve the calling context.

```
typedef void (_stdcall * OnEMVTransactionCompleteEvent)(
     BYTE opStatus,
     void* pData,
     void *object);
```

Parameter	Description
opStatus	Byte value indicates device operation status. Zero value means OK. For more values, see **Appendix B**.
pData	Pointer to a `EMV_DATA` structure stored merchant data. See section **3.70 RequestPowerUpResetICCWithWaitTimeSync** for type definition.
object	Pointer forwarded by `OnEMVTransactionComplete`

4.17 OnCardHolderStateChanged

This event will trigger when the cardholder performs state-changing actions (such as insertion or removal of a smart card) in response to **RequestSmartCard**.

```
void _stdcall OnCardHolderStateChanged(
     OnCardHolderStateChangedEvent lpFuncNotify,
     void *object = NULL);
```

Parameter	Description
lpFuncNotify	Pointer to a custom `OnCardHolderStateChangedEvent` callback function (see `typedef` below)
object	Pointer to forward to the `OnCardHolderStateChangedEvent` type function when the event is triggered. For example, can be used to pass `this` to preserve the calling context.

```
typedef void (_stdcall * OnCardHolderStateChangedEvent)(
     BYTE opStatus,
     BYTE stateId,
     void *object);
```

Parameter	Description
opStatus	Byte value indicates device operation status. Zero value means OK. For more values, see **Appendix B**.
stateId	EMV Cardholder interaction ID. 0x01 = Waiting for amount confirmation selection 0x02 = Amount confirmation selected 0x03 = Waiting for multi-payment application selection 0x04 = Application selected 0x05 = Waiting for signature capture 0x06 = Signature captured 0x07 = Waiting for language selection 0x08 = Language selected 0x09 = Waiting for credit/debit selection 0x0A = Credit/Debit selected 0x0B = Waiting for PIN Entry for ICC 0x0C = PIN entered for ICC 0x0D = Waiting for PIN Entry for MSR 0x0E = PIN entered for MSR
object	Pointer forwarded by `OnCardHolderStateChanged`

4.18 OnProgressUpdate

Call this function to register a updating event handler for SendBitmap or UpdateFirmware.

```
void _stdcall OnProgressUpdate (
     OnProgressUpdateEvent lpFuncNotify,
```

```
      void *object = NULL);
```

Parameter	Description
lpFuncNotify	Pointer to a custom `OnProgressUpdateEvent` callback function (see `typedef` below)
object	Pointer to forward to OnProgressUpdateEvent type function when the event is triggered. For example, can be used to pass `this` to preserve the calling context.

```
typedef void (_stdcall * OnProgressUpdateEvent)(
      BYTE opStatus,
      int UpdateItem,
      double updateProgress,
      void *object);
```

Parameter	Description
opStatus	Byte value indicates device operation status. Zero value means OK. For more values, see **Appendix B**.
UpdateItem	0x0C – SendBitmap 0x17 – UpdateFirmware
updateProgress	From 0.0 to 1.0. value large than 1 mean extra action notified 1.0 means sending data completed and delay action received
object	Pointer forwarded by `OnEMVTransactionComplete`

4.19 DeviceMessageTipOrCashback

This event will trigger when the cardholder selects a menu item in response to RequestTipOrCashback.

```
void _stdcall DeviceMessageTipOrCashback(
      DeviceMessageTipOrCashbackEvent lpFuncNotif,
      void *object = NULL);
```

Parameter	Description
lpFuncNotify	Pointer to a custom `DeviceMessageTipOrCashback` callback function (see `typedef` below)
object	Pointer to forward to the `DeviceMessageTipOrCashbackEvent` type function when the event is triggered. For example, can be used to pass `this` to preserve the calling context.

```
typedef void (_stdcall * DeviceMessageTipOrCashbackEvent)(
      BYTE opStatus,
      BYTE mode,
      BYTE amount[6],
```

```
      BYTE tax[6],
      BYTE taxRate[3],
      BYTE tipOrCashback[6],
      BYTE reserved,
      void* object);
```

Parameter	Description
opStatus	Byte value indicates device operation status. Zero value means OK. For more values, see **Appendix B**.
mode	0x00=Tip 0x01=Cashback
amount	N12 for amount
tax	N12 for tax
taxRate	N6 * 100 for tax rate Example: 0x098750 = 9.875%
tipOrCashback	N12 for tip amount or cashback amount depending on the mode.
reserved	reserved bytes
object	Pointer forwared by `DeviceMessageTipOrCashback`

4.20 DeviceMessageSelectedMenuItem

This event will trigger when the cardholder selects a menu item in response to GetSelectedMenueItem.

```
void _stdcall DeviceMessageSelectedMenuItem(
      DeviceMessageSelectMenuItemEvent lpFuncNotif,
      void *object = NULL);
```

Parameter	Description
lpFuncNotify	Pointer to a custom `DeviceMessageSelectedMenuItem` callback function (see `typedef` below)
object	Pointer to forward to the `DeviceMessageSelectedMenuItemEvent` type function when the event is triggered. For example, can be used to pass `this` to preserve the calling context.

```
typedef void (_stdcall * DeviceMessageSelectedMenuItemEvent)(
      BYTE opStatus,
      BYTE mode,
      BYTE index,
      BYTE reserved,
      void* object);
```

Parameter	Description
opStatus	Byte value indicates device operation status. Zero value means OK. For more values, see **Appendix B**.
mode	0x00=Selection Table 0x01=Selection Bill
index	Index for Selected Menu Item. First item selected is index 0.
reserved	reserved bytes
object	Pointer forwarded by `DeviceMessageSelectedMenuItem`

Appendix A Code Examples

A.1 Open Device

```
if (OpenDevice() == 0) {
      …
}
```

A.2 Close Device

```
CloseDevice();
```

A.3 RequestCard

```
RequestCard(20, CardMsgSwipeCard, BuzzerSingleBeep);
```

A.4 ConfirmAmount

```
int opStatus;
int iResult = SetAmount(0x67, "20.67",&opStatus);
if (iResult == 0) {
    iResult = ConfirmAmount(30, BuzzerSingleBeep);
}
```

Appendix B Status Codes

B.1 Operation Status Codes

0x00 = OK / Done
0x01 = User Cancel
0x02 = Timeout
0x03 = Host Cancel
0x04 = Verify fail
0x05 = Keypad Security
0x06 = Calibration Done
0x07 = Write with duplicate RID and index
0x08 = Write with corrupted Key
0x09 = CA Public Key reached maximum capacity
0x0A = CA Public Key read with invalid RID or Index

B.2 Response Status Codes

0x00 = OK / Done
0x80 = System Error
0x81 = System not Idle
0x82 = Data Error
0x83 = Length Error
0x84 = PAN Exists
0x85 = No Key or Key is incorrect
0x86 = System busy
0x87 = System Locked
0x88 = Auth required
0x89 = Bad Auth
0x8A = System not Available
0x8B = Amount Needed
0x90 = Cert non-exist
0x91 = Expired (Cert/CRL)
0x92 = Invalid (Cert/CRL/Message)
0x93 = Revoked (Cert/CRL)
0x94 = CRL non-exist
0x95 = Cert exists
0x96 = Duplicate KSN/Key

B.3 Return Codes

0x00 = SUCCESS
0x01 = TIMEOUT
0x02 = USER_CANCELLED
0x03 = CREATEFILE_FAILED
0x04 = IPAD_NOT_FOUND
0x05 = DEVICE_NOT_OPEN
0x06 = INVALID_PARAM
0x07 = DEVICE_ERROR
0x08 = INVALID_MSG_ACK
0x09 = GENERAL_ERROR
0x0A = CARDREQUEST_COMPLETE_TIMEOUT

0x0B = INVALID_PINLENGTH
0x0C = INVALID_BUFFER
0x0D = INVALID_BUFFER_SIZE
0x0E = UNSUPPORT_FUNCTION
0x0F = BUSY
0x10 = CORRECT_DATA_NOT_EXIST
0xFF =UNKNOWN_ERROR

Appendix C Cryptography

C.1 Decrypt PIN

PIN_DATA structure contains the information need to decrypt PIN.

```
typedef struct _PIN_DATA
{
      BYTE OpStatus;
      char *KSN;
      char * EPB;
} PIN_DATA;
```

C.1.1 Get key for the PIN decryption from BDK and KSN

First, convert KSN and EPB from hex string to byte array for further calculation.

```
char bKSN[10];
char bEPB[8];

ConvertHexStringToByteArray(PinData.KSN, bKSN, 10);
ConvertHexStringToByteArray(PinData.EPB, bEPB, 8);
```

Then, derive key from BDK and KSN

```
char bPinKey[16];

// To get the bPinKey, reference to ANSI X9.24
```

C.1.2 Use Triple DES CBC to decrypt PIN block

Decrypt Encrypted PIN Block, use empty initial vector.

```
char bIsoPinBlock[8];
char iv[8]={0,0,0,0,0,0,0,0};

TDES_Decrypt_CBC(bPinKey, iv, bEPB, 8, bIsoPinBlock, 8);
```

C.1.3 Extract PIN from PIN block

Reference to ISO 9564-1.

C.2 Decrypt Card Track

CARD_DATA structure contains the information need to decrypt track 1,2 and 3.

```
typedef struct _CARD_DATA
{
      BYTE DataType;
      BYTE CardOperationStatus;
      BYTE CardStatus;
      BYTE CardType;

      BYTE Track1Length;
      BYTE Track2Length;
      BYTE Track3Length;
      BYTE EncTrack1Length;
      BYTE EncTrack2Length;
```

```
    BYTE EncTrack3Length;
    BYTE EncMPLength;

    BYTE Track1Status;
    BYTE Track2Status;
    BYTE Track3Status;
    BYTE EncTrack1Status;
    BYTE EncTrack2Status;
    BYTE EncTrack3Status;
    BYTE EncMPStatus;
    BYTE MSStatus;

    char *MPSTS;
    char *Track1;
    char *Track2;
    char *Track3;
    char *EncTrack1;
    char *EncTrack2;
    char *EncTrack3;
    char *EncMP;
    char *KSN;

    char* CBCMAC;
    char* SerialNumber;
    BYTE  PANInfoLength;
    char* PANInfo;

    unsigned long reserved;
} CARD_DATA;
```

C.2.1 Get encrypted track data from CARD_DATA

First, convert EncTrack1, EncTrack2, EncTrack3 and KSN from hex string to byte array for further calculation.

```
char bEncTrack1[256], bEncTrack2[256], bEncTrack3[256];
char bKSN[10];

ConvertHexStringToByteArray(CardData.KSN, bKSN, 10);
ConvertHexStringToByteArray(CardData.EncTrack1, bEncTrack1,
CardData.EncTrack1Length);
ConvertHexStringToByteArray(CardData.EncTrack2, bEncTrack2,
CardData.EncTrack2Length);
ConvertHexStringToByteArray(CardData.EncTrack3, bEncTrack3,
CardData.EncTrack3Length);
```

C.2.2 Get Key from KSN

```
char bDataKey[16];

// To get the bDataKey, reference to ANSI X9.24
```

C.2.3 Use Triple DES CBC to decrypt track data

Decrypt Encrypted card track data, use empty initial vector.

```
char iv[8]={0,0,0,0,0,0,0,0};
char bDecTrack1 = new char[CardData.EncTrack1Lenth];

TDES_Decrypt_CBC(bPinKey, iv, bEncTrack1, CardData.EncTrack1Length,
bDecTrack1, CardData.EncTrack1Length);

// Decrypted data of track 1 should use CardData.Track1Length as the
byte array length.
```

C.3 Calculate CBC MAC

CBC MAC (cipher block chaining message authentication code) uses empty initial vector to encrypt message block.

C.3.1 Get key

Derive key from BDK and KSN

```
char bDataKey[16];
char IV[8] = {0,0,0,0,0,0,0,0};

// To get the bDataKey, reference to ANSI X9.24
```

C.3.2 Padding data

Use 8 byte as a block and pad rest of space to zero.

```
int nPaddedLength = ((nDataLength + 7) & (~7));
char* bDataPadded = new char (nPaddedLength);
memset(bDataPadded, 0 , nPaddedLength);
memcpy(bDataPadded, data, nDataLength);
```

C.3.3 Calculate MAC by CBC

Use DES CBC to encrypt data, then use DES DECB and DES ECB to encrypt last block as MAC. Then the most left 32 bits as MAC value.

```
char* pLeftKey = bDataKey;
char* pRightKey = bDataKey+8;

char* bDataOutput = new char (nPaddedLength);

DES_Encrypt_CBC(pLeftKey, IV, bDataPadded, nPaddedLength, bDataOutput,
nPaddedLength);

// bDataOutput contain the encrypted data

char* pLastBlock = bDataOutput + nPaddedLength - 8;
char MAC[8];

DES_Decrypt_ECB(pRightKey, IV, pLastBlock, 8, MAC, 8);
DES_Encrypt_ECB(pLeftKey, IV, MAC, 8, MAC, 8);

// We only use first 4 bytes of MAC buffer.
```

C.4 Cryptography in CA Public Key, EMV Tag and EMV transaction

Get/Set CA Public Key, Get/Set EMV Tags and EMV transaction use TLV (type-length-value) format.

C.4.1 Send data to DynaPro/DynaPro Go/DynaPro Mini

Send data should use **SendBigBlockData** or use appropriate function like **SetCAPublicKey**.

```
// Compose TLV Message
TLV_ComposeMessage(message, SerialNumber, bOutMessage, &nOutMessage);

// Use CBC MAC to encrypt message and add MAC, reference to C4
CBC_MAC(bOutMessage, nOutMessage, bSecuredMessage,
nSecuredMessageLength);

SetCAPublicKey(operation, bSecuredMessage, nSecuredMessageLength);
```

C.4.2 Receive data from DynaPro/DynaPro Go/DynaPro Mini

Parse the data through TLV format. For encrypted data tag, use TDES_Decrypt_CBC to decrypt it.

C.5 Example of RequestSmartCard

Following sample code demonstrate an EMV transaction flow.

C.5.1 Host: RequestSmartCard

```
BYTE Amt[6] = {0x0,0x0,0x0,0x01,0x0,0x0};
BYTE Cashback[6] = {0x0,0x0,0x0,0x0,0x0,0x0};

int retCode =
RequestSmartCard(ContactSmartcard,20,20,1,2,(char*)Amt,4,(char*)(Cashb
ack),NULL);
```

C.5.2 Device: OnEMVDataComplete

Host will receive callback event OnEMVDataComplete. In the callback, will extract TLV data. Data format can reference to document 99875585 – 3.5.2 ARQC Request

```
void __stdcall OnEMVDataCompleteCallback(BYTE opStatus, void*
pEMVDATA, void* obj)
{
      ACQUIRER_DATA * pARQCData = (ACQUIRER_DATA *)pEMVDATA;

      TLVData* tlvData = TLVData.Parse(pARQCData->Data,pARQCData-
>Length);

      unsigned char KSN[10];
      int ksnlen = 10;
      tlvData->getKSN(KSN, &ksnlen);

      …
}
```

C.5.3 Host: SendAcquirerResponse

Host should send out acquirer response to complete the EMV transaction, or transaction will be denied with time out.

To generate the response data, reference to document 99875585 – 3.5.2.3 ARQC Response

```
unsigned char approve[6] = {0x70,0x04,0x8A,0x02,0x30,0x30};
unsigned char Msg[47];
memset(Msg, 0, 47);

// Set Data Length
Msg[0] = 0;
Msg[1] = 45;
Msg[2] = 0xF9;
Msg[3] = 0x82;
Msg[4] = 0;
Msg[5] = 6;

// Set KSN
Msg[6] = 0xDF;
Msg[7] = 0xDF;
Msg[8] = 0x54;
Msg[9] = 10;

memcpy(Msg+10, KSN, 10);

// Set Encryption Type
Msg[20] = 0xDF;
Msg[21] = 0xDF;
Msg[22] = 0x55;
Msg[23] = 0x01;
Msg[24] = 0x82;

// Set Serial Number
Msg[25] = 0xDF;
Msg[26] = 0xDF;
Msg[27] = 0x25;
Msg[28] = 0x08;

memcpy(Msg+29, SerialNumber, 8);

// Set Data
Msg[37] = 0xFA;
Msg[38] = 0x82;
Msg[39] = 0;
Msg[40] = 6;

memcpy(Msg+41, approve, 6);

unsigned char OutputMsg[52];

// Use CBC MAC to encrypt message and add MAC, reference to C.4
CBC_MAC(Msg, 47, OutputMsg, 52);

SendAcquirerResponse(OutputMsg, 52);
```

C.5.4 Device: OnEMVTransactionComplete

Host will receive callback event OnEMVTransactionComplete. In the callback, will extract TLV data.

```
void __stdcall OnEMVTransactionCompleteCallback(BYTE status, void*
pData, void *object)
{
      EMV_DATA* emvData = (EMV_DATA*)pData;

      …
}
```

C.6 Reference Documents

DUKPT - ANSI x9.24
DES – FIPS 46-3
TDES – ANSI X9.52
MAC – ANSI X9.19
ISO PIN BLOCK – ISO 9564
TLV – ASN.1 and ITU-T X.690

Appendix D Contact Smart Card L1 Session (DynaPro L1 Only)

D.1 Overview

DynaPro L1 has enabled host to communicate with Contact Smart Card in Application Protocol Data Unit (APDU) layer.

D.2 Create L1 Session

The secure communication starts with creating a secure session. Host request challenge and session from device, then confirm host has the right to do the secure communication.

The host must follow these steps to create L1 session:

1) Request an authentication token and session key from the device using RequestChallengeAndSession.
2) Decrypt the received token with the Acquirer Master key.
3) Transform the token and encrypt it with the Acquirer Master key.
4) Calculate 8-byte CMAC for the message.
5) Using RequestConfirmSession to create communicate session.

```
// predefined key deriving mask
unsigned char amkDerivedSessionCMAC_Mask[] = { 0x5e, 0x55, 0x00, 0xb7,
0x89, 0xc4, 0x76, 0xf3, 0x6d, 0xac, 0xdc, 0x90, 0x13, 0x2a, 0xbd,
0x16, 0x29, 0x2a, 0xaa, 0xce, 0xe2, 0x90, 0xb4, 0xee };
unsigned char derivedSessionCMAC_Mask[] = { 0xab, 0x54, 0x65, 0x7d,
0xff, 0x33, 0x31, 0xf7, 0xad, 0x22, 0x93, 0x11, 0x62, 0x48, 0xc5,
0xf3, 0x33, 0x31, 0x0b, 0x6e, 0x68, 0x25, 0xcc, 0xa3 };
unsigned char amkDerivedSessionMask[] = { 0x12, 0x10, 0x74, 0x10,
0x26, 0x75, 0x03, 0x08, 0x06, 0x04, 0x28, 0x08, 0x04, 0x02, 0x10,
0x10, 0x26, 0x75, 0x11, 0x08, 0x01, 0x11, 0x03, 0x91 };


// if your AMK is 16 bytes, extend AMK to 24 bytes before derive key.
// to extend AMK, append left 8 bytes to the end.
bAMKSessionKey = bitXor(AMK, amkDerivedSessionMask);
bAMKCMACKey = bitXor(AMK, amkDerivedSessionCMAC_Mask);

// Get Challenge And Session, 46 bytes of data is saved to buffer.
int retCode = RequestChallengeAndSession(buffer, bufferLen,
&opStatus);

// bytes 2-5 is encrypted partial serial number, byte 6-9 is encrypted
// random number, bytes 10-33 is encrypted session key, (see 99875585)

char iv[8]={0,0,0,0,0,0,0,0};

TDES_Decrypt_CBC(bAMKSessionKey, iv, buffer+2, 32, SessionInfo, 32);

// session info
// 0-3 partial serial number, 4-7, random number, 8-31, session key
```

```
Memcpy(SessionKey, SessionInfo+8 , 24);
// derive session cmac key
SessionCMACKey = bitXor(SessionKey, derivedSessionCMAC Mask)

// transform rndNumber
unsigned long rndNumber = GetInt32(SessionInfo+4);
rndNumber += 0x55555555; // Magic Number

unsigned char transformedNumberSerial[8];
SetInt32(transformedNumberSerial, rndNumber);

Memcpy(transformedNumberSerial+4, SessionInfo, 4);

// Encrypt Transformed Random Number Partial Serial Number.
unsigned char trns[8]
TDES_Encrypt_CBC(bAMKSessionKey, iv, transformedNumberSerial, 8,
trns,8);

// Calculate CMAC
unsigned char cmac[8];
CMAC(bAMKCMACKey, trns, 8, cmac, 8);

retCode = RequestConfirmSession(1, trns, trns+4, cmac, &opStatus);
```

D.3 Power Up ICC Card and Get ATR

Host use RequestPowerUpResetICC to power up smart card and get the card ATR by call back.

```
void WINAPI myATRReceivedCallback (BYTE opStatus, void* pData, void
*context)
{
     EMV_DATA* data = (EMV_DATA*)pData;

     // Decrypt Secured ATR by SessionKey
     TDES_Decrypt_CBC(SessionKey, iv, data->Data, data->Length,
ATRBuffer,ATRBufferLen);

}

OnPowerUpICC(myATRReceivedCallback, myContext);

// 30 seconds to wait for present ICC. 1 for power up.
retCode = RequestPowerUpResetICC(30, 1);

// you will receive callback in myATRReceivedCallback
```

D.4 Send APDU to Card and Get Response

Host use RequestICCAPDU to communicate with card and get card returned APDU by callback.

```
Void WINAPI myApduArrived(
     BYTE opStatus,
     BYTE* pData,
     DWORD pdwDataLen,
     void *context)
{
     // Decrypt Secured APDU-R by SessionKey
     TDES_Decrypt_CBC(SessionKey, iv, pData, pdwDataLen,
APDUBuffer,APDUBufferLen);
     // byte 0 is the length of APDU returned
     backApdu = SubArray(APDUBuffer, APDUBuffer[0]);
}


// Set APDU Callback
OnAPDUArrived(myApduArrived, 0)

// Encrypt APDU by Session Key
     TDES_Encrypt_CBC(SessionKey, iv, Apdu, ApduLength,
EncApduBuffer,EncApduBufferLen);
// Generate CMAC for this APDU
unsigned char cmac[8];
     CMAC(SessionCMACKey, iv, EncApduBuffer, EncApduBufferLen, cmac,
8);

// Append cmac to secured apdu
AppendByteArray(EncApduBuffer, EncApduBufferLen, cmac, 8)

// 30 seconds to wait for present ICC. 1 for power up.
retCode = RequestICCAPDU(EncApduBuffer, EncApduBufferLen);

// you will receive callback in myATRReceivedCallback
```

D.5 Power Down ICC

Host use RequestPowerDownICC to power down card.

```
// 30 seconds to wait for present ICC. 1 for power up.
retCode = RequestPowerDownICC(1); // 1 second before power down
```

D.6 End L1 Session

Host use EndL1Session to close the secure communication

```
retCode = EndL1Session();
```

Appendix E Function Applicable Table

Function Name	IPAD	DYNAPRO	DYNAPRO MINI	DYNAPRO GO
GetSDKVersion	YES	YES	YES	YES
OpenDevice	YES	YES	YES	YES
CloseDevice	YES	YES	YES	YES
GetDeviceList	YES	YES	YES	YES
SetDeviceTypeParameter	YES	YES	YES	YES
IsDeviceOpened	YES	YES	YES	YES
DeviceReset	YES	YES	YES	YES
SetTimeOut	YES	YES	YES	YES
GetStatusCode	YES	YES	YES	YES
CancelOperation	YES	YES	YES	YES
RequestBypassPINCommand	NO	YES	YES	YES
SetPAN	YES	YES	YES	YES
SetAmount	YES	YES	YES	YES
EndSession	YES	YES	YES	YES
RequestChallengeAndSessionForInformation	NO	YES	YES	YES
RequestConfirmSession	NO	YES	YES	YES
EndL1Session	NO	YES	YES	YES
RequestPowerUpResetICC	NO	YES	YES	YES
RequestPowerDownICC	NO	YES	YES	YES
RequestICCAPDUForInformation	NO	YES	YES	YES
SendSpecialCommand	YES	YES	YES	YES
GetSpecialCommand	YES	YES	YES	YES
RequestGetEMVTags	NO	YES	YES	YES
RequestGetEMVTagsEx	NO	YES	YES	YES
RequestSetEMVTags	NO	YES	YES	YES
RequestGetEMVTagsEx	NO	YES	YES	YES
SetCAPublicKey	NO	YES	YES	YES
SetDisplayMessage	YES	YES	YES	YES

SendBigBlockData	YES	YES	YES	YES
SendBitmap	YES	YES	YES	YES
GetIPADInfoData	YES	YES	YES	YES
RequestDeviceInformation	NO	YES	YES	YES
RequestDeviceStatusForInformation	YES	YES	YES	YES
RequestKernelInformation	NO	YES	YES	YES
GetBINTableData	NO	YES	YES	YES
SetBINTableData	NO	YES	YES	YES
GetKSN	YES	YES	YES	NO
SetKSNEncryptedData	YES	YES	YES	YES
RequestCard	YES	YES	YES	YES
RequestManualCardData	YES	YES	YES	YES
RequestUserDataEntry	YES	YES	YES	YES
RequestResponse	YES	YES	YES	YES
ConfirmAmount	YES	YES	YES	YES
SelectCreditDebit	YES	YES	YES	YES
RequestPIN	YES	YES	YES	YES
RequestSignature	YES	YES	NO	RF
RequestSmartCard	NO	YES	YES	YES
SendAcquirerResponse	NO	YES	YES	YES
getCardDataInfo	YES	YES	YES	YES
RequestDeviceConfigurationForInformation	YES	YES	YES	YES
GetProductID	YES	YES	YES	YES
GetDeviceSerial	YES	YES	YES	YES
GetDeviceModel	YES	YES	YES	YES
GetDeviceFirmwareVersion	YES	YES	YES	YES
GetPINKSN	YES	YES	YES	YES
GetPAN	YES	YES	YES	YES
GetEncodeType	YES	YES	YES	YES
GetTrack1	YES	YES	YES	YES
GetTrack2	YES	YES	YES	YES

GetTrack3	YES	YES	YES	YES
GetTrack1Masked	YES	YES	YES	YES
GetTrack2Masked	YES	YES	YES	YES
GetTrack3Masked	YES	YES	YES	YES
GetMaskedTracks	YES	YES	YES	YES
GetMagnePrint	YES	YES	YES	YES
GetMagnePrintStatus	YES	YES	YES	YES
GetTrack1DecodeStatus	YES	YES	YES	YES
GetTrack2DecodeStatus	YES	YES	YES	YES
GetTrack3DecodeStatus	YES	YES	YES	YES
GetLastName	YES	YES	YES	YES
GetFirstName	YES	YES	YES	YES
GetMiddleName	YES	YES	YES	YES
GetExpDate	YES	YES	YES	YES
GetPINStatusCode	YES	YES	YES	YES
GetPINData	YES	YES	YES	YES
GetEPB	YES	YES	YES	YES
ClearBuffer	YES	YES	YES	YES
RequestClearTextUserDataEntry	NO*	NO*	NO	YES
RequestClearTextUserDataEntrySync	NO*	NO*	NO	YES
RequestGetEMVTagsSync	NO	YES	YES	YES
SetCAPublicKeySync	NO	YES	YES	YES
RequestSmartCardSync	NO	YES	YES	YES
RequestICCAPDUForInformationSync	NO	YES	YES	YES
RequestPowerUpResetICCSync	NO	YES	YES	YES
SendAcquirerResponseSync	NO	YES	YES	YES
RequestUserDataEntrySync	YES	YES	YES	YES
RequestSignatureSync	YES	YES	NO	RF
RequestReponseSync	YES	YES	YES	YES
OnError	YES	YES	YES	YES
OnDataReady	YES	YES	YES	YES

OnPowerUpICC	NO	YES	YES	YES
OnAPDUArrived	NO	YES	YES	YES
OnGetCAPublicKey	NO	YES	YES	YES
OnEMVTagsComplete	NO	YES	YES	YES
OnPINRequestComplete	YES	YES	YES	YES
OnKeyInput	YES	YES	YES	YES
OnDisplayRequestComplete	YES	YES	YES	YES
OnSignatureArrived	YES	YES	NO	RF
OnCardRequestComplete	YES	YES	YES	YES
OnUserDataEntry	YES	YES	YES	YES
OnDeviceStateUpdated	YES	YES	YES	YES
OnEMVDataComplete	NO	YES	YES	YES
OnCardHolderStateChanged	NO	YES	YES	YES
OnEMVTransactionComplete	NO	YES	YES	YES
OnClearTextUserDataEntry	NO*	NO*	NO	YES
IsDeviceSRED	YES	YES	YES	YES
UpdateFirmware	YES	YES	YES	YES
GetKeyInfo	YES	YES	YES	YES
GetAMKInfo	YES	YES	YES	YES
RequestSetEMVTagsSync	NO	YES	YES	YES
SetDisplayMessageSync	YES	YES	YES	YES
RequestCardSync	YES	YES	YES	YES
RequestManualCardDataSync	YES	YES	YES	YES
ConfirmAmountSync	YES	YES	YES	YES
RequestPINSync	YES	YES	YES	YES
OnProgressUpdate	YES	YES	YES	YES
SelectCreditDebitSync	YES	YES	YES	YES

YES – device support this function
NO – device does not support this function
NO* only some firmware for this device support this function.
RF – reserved for future use.