MAGTEK®
SECURITY FROM THE INSIDE

IPAD, DynaPro, DynaPro Go, and DynaPro Mini

PIN Encryption Devices

Programmer's Reference (.NET Wrapper)

March 2017

Manual Part Number:
D99875684-60

REGISTERED TO ISO 9001:2008

MagTek I 1710 Apollo Court I Seal Beach, CA 90740 I Phone: (562) 546-6400 I Technical Support: (888) 624-8350
www.magtek.com

Table 0.1 – Revisions

Rev Number	Date	Notes
1.01	May 29, 2014	Initial release
2.01	July 18, 2014	Add updateFirmware and OnProgressUpdate
30	January 20, 2015	Add cryptography in appendix C Add appendix D, Function Applicable table Update SendBitmap, bitmap size Added IPAD to the supported devices list.
40	April 21, 2015	Update the function updateFirmware to load firmware from a data buffer.
50	April 25, 2016	Update setCAPublicKey, requestGetEMVTags and requestSetEMVTags.
60	March 3, 2017	Add support for DynaPro Go. Add note to getKSN function of its usage for token reversal.

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, DynaPro Go, or DynaPro Mini connected to a Windows-based host via USB, Ethernet, or BLE. 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:

- ***99875586 DynaPro Installation and Operation Manual***
- ***99875642 DynaPro Mini Installation and Operation Manual***
- ***99875585 DynaPro Programmer's Reference (Commands)***
- ***99875629 DynaPro Mini Programmer's Reference (Commands)***
- ***99875656 DynaPro and DynaPro Mini Programmer's Reference (C++)***
- ***99875668 DynaPro and DynaPro Mini Programmer's Reference (Android)***
- ***99875633 DynaPro and DynaPro-Mini Programmer's Reference (Java and Java Applet)***
- ***99875654 DynaPro Mini Programmer's Reference (iOS)***

1.1 About the MagTek PIN Pad SCRA .NET Demo

The MTPPSCRACS Demo software, available from MagTek, provides C# demonstration source code and a reusable MTPPSCRANET .NET library that provides developers of custom software solutions with an easy-to-use interface for IPAD, DynaPro, DynaPro Go, and DynaPro Mini. Developers can include the MTPPSCRANET library 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.”

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

1.3 SDK Contents

File name	Description
MTPPSCRACS Demo\	Includes a Visual Studio project and source code to build **MTPPSCRACS Demo.exe**.
MTPPSCRANET.dll	This is a Microsoft .NET DLL wrapper around `MTPPSCRA.dll`.
MTPPSCRA.dll	DLL required to interact with the PIN Encryption Device.

File name	Description
MTPPBLE.dll	DLL required for interaction with Bluetooth Low Energy (BLE) devices.

1.4 System Requirements

Tested operating systems:

- Windows 7
- Windows 8, 8.1
- Windows 10

Microsoft .NET Framework 4.0 or above

BLE requires Windows 8 or higher.

2 How to Set Up the MagTek PIN Pad SCRA Demo

To set up the MTPPSCRANET 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 build the MTPPSCRACS Demo software, follow these steps:

1) Launch Visual Studio and open **MTPPSCRACS Demo.csproj**
2) In the **Solution Explorer**, select **MTPPSCRACS Demo**.
3) Open Reference and add MTPPSCRANET.dll
4) Select **Build** Configuration and Platform
5) Select **Build** > **Build MTPPSCRACS Demo**.

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

1) Select Build Configuration equal to Debug and Build platform to either x86 or x64
2) In VisualStudio, select **Debug**->**Start Without Debugging** to run the MTPPSCRACS Demo, or select **Debug**->**Start Debugging** to run it in debug mode.

3 MTPPSCRANET Functions

If you are developing Microsoft .NET software, follow the setup steps in section 2 **How to Set Up the MagTek PIN Pad SCRA Demo**, then create an instance of the MTPPSCRANET object in your software project, then use the method calls to invoke the functions described in this chapter to communicate with the device. For sample code that demonstrates how to use these functions, see `MTPPSCRACS Demo` in the SDK files.

Generally, these functions will run in one of two modes:

- **Asynchronous** functions will return data using the event handlers (callback functions) defined in section 4 **MTPPSCRANET** .
- **Synchronous** functions will return requested data immediately in the function's return value. If the requested data is not available immediately, synchronous calls will generally block until a specified wait time has elapsed.

Most calls that wait for input from the user will run in the asynchronous mode.

3.1 getSDKVersion

This function retrieves the library version information.

```
String getSDKVersion( );
```

Return Value: String containing the Version of the Java library.

3.2 openDevice

This function opens a connection to the device.

```
int openDevice(String deviceURI);
```

Parameter	Description
deviceURI	URI of the device. For USB devices, deviceURI should be an empty string. For Ethernet devices, deviceURI should be in the form `IP://IP-Address:PORT`, for example, `Ip://10.57.10.180:26`

Return Value:
Returns a value (0: Success, Non-Zero: Error).

3.3 closeDevice

This function closes the connection to the device. The event associated with this command is **OnErrorEvent.**

```
int closeDevice();
```

Return Value:
Returns a value (0: Success, Non-Zero: Error).

3.4 getDeviceList

This function enumerates all PIN Encryption Devices.

```
String getDeviceList();
```

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

3.5 isDeviceOpened

This function retrieves the device’s open status.

```
boolean isDeviceOpened();
```

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

3.6 deviceReset

This function sends a reset command to the device.

```
int deviceReset();
```

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

3.7 getStatusCode

This function retrieves the current status of the issued report.

```
int getStatusCode();
```

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

3.8 cancelOperation

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

```
int cancelOperation();
```

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

3.9 requestBypassPINCommand

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

```
int requestBypassPINCommand();
```

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

3.10 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 setPAN(String lpPAN);
```

Parameter	Description
lpPAN	PAN data (8 - 19 ASCII digits) in a null-terminated String

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

3.11 setAmount

This function sets the transaction amount before beginning a transaction.

```
int setAmount(byte amountType, String lpAmount, ref int opStatus);
```

Parameter	Description
amountType	RFU
lpAmount	Amount to be used for the transaction, should be a null terminated string. For example "20.56"
opStatus	A byte array to receive the command response or operation status. Zero value means OK. For more values, see **Appendix A**.

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

3.12 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 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 a value (0: Success, Non-Zero: Error)

3.13 requestChallengeAndSession (EMV L1 only) [Deprecated: V1.04]

This function retrieves the challenge key and session key for a smart card transaction. For additional information, see **Appendix B EMV CBC-MAC**.

```
byte[] requestChallengeAndSession();
```

Return Value:
A byte array containing 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.

3.14 requestChallengeAndSessionForInformation (EMV L1 only)

This function retrieves the challenge key and session key for a smart card transaction. For additional information, see **Appendix B EMV CBC-MAC**.

```
byte[] requestChallengeAndSession();
```

Return Value:

A byte array containing 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.

3.15 requestConfirmSession (EMV L1 only)

This function sends a CMAC message to the device to confirm the session key. For additional information, see **Appendix B EMV CBC-MAC**.

```
int requestConfirmSession(
     int mode,
     byte [] encryptedRandomNumber,
     byte [] encryptedSerialNumber,
     byte [] cmac,
     ref 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	An integer pointer to receive the command response or operation status. Zero value means OK. For more values, see **Appendix A**.

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

3.16 endL1Session (EMV L1 only)

This function ends the session with the device.

```
int endL1Session(ref int opStatus);
```

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

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

3.17 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 requestPowerUpResetICC(byte waitTime, byte operation);
```

Parameter	Description
waitTime	Time the device will wait for the user to insert a smart card
operation	Operation ID 0 = Power down 1 = Power up or warm reset

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

3.18 requestPowerDownICC (EMV L1 only)

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

```
int requestPowerDownICC(byte waitTime);
```

Parameter	Description
waitTime	Not used.

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

3.19 requestICCAPDU (EMV L1 only)

This function sends the ICC APDU report to the device. The event associated with this command is **OnAPDUArrived**. For additional information, see **Appendix B EMV CBC-MAC**.

```
int requestICCAPDU(byte [] apdu, int apduLen);
```

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

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

3.20 requestICCAPDUForInformation (EMV L1 only)) [Deprecated: V1.04]

This function sends the ICC APDU report to the device. The event associated with this command is **OnAPDUArrived**. For additional information, see **Appendix B EMV CBC-MAC**.

```
int requestICCAPDU(byte [] apdu, int apduLen);
```

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

Return Value:

Returns a value (0: Success, Non-Zero: Error)

3.21 sendSpecialCommand

This function sends a direct "SET" 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)***. The event associated with this command is **OnDataReady**.

```
int sendSpecialCommand(String lpCommand);
```

Parameter	Description
lpCommand	A hexadecimal command string to send to device.

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

3.22 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)***.

```
byte[] getSpecialCommand(String lpCommand);
```

Parameter	Description
lpCommand	A hexadecimal command string will send to device.

Return Value: A byte array containing the response from the device. See the ***(Commands)*** references above for details.

3.23 requestGetEMVTags

This function sends the EMV Tag report to the device to read or write EMV Tags. For additional information, see **Appendix B EMV CBC-MAC**.

```
int requestGetEMVTags(
      int tagType,
      int tagOperation,
      byte [] inputTLVData,
      int inputDataLength,
      byte database,
      byte option,
      byte[] 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

Parameter	Description
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 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 – EspressPay-AMEX 04 – Discover
option	Response type: 0x00 – Normal 0x01 – Delay Response
reserved	Reserved Bytes

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

3.24 requestSetEMVTags

This function sends the EMV Tag report to the device to read or write EMV Tags. For additional information, see **Appendix B EMV CBC-MAC**.

```
int requestSetEMVTags(
     int tagType,
     int tagOperation,
     byte [] inputTLVData,
     inputDataLength,
     byte database,
     byte option,
     byte[] 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: 0x04 – Write EMV Reader tags 0x05 – Write EMV Application tags 0xFF – Set to factory defaults

Parameter	Description
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 – EspressPay-AMEX 04 – Discover
option	Response type: 0x00 – Normal 0x01 – Delay Response
reserved	Reserved Bytes

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

3.25 setCAPublicKey

This function sets / deletes the corresponding CA Public Key, depending on the operation specified. For additional information, see **Appendix B EMV CBC-MAC**.

```
int setCAPublicKey(
     int operation,
     byte [] 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 send
keyBlockLength	Length of the CA Public Key

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

3.26 setDisplayMessage

This function shows a predefined message or bitmap on the device's LCD display. The event associated with this function is **OnDisplayRequestComplete**.

```
int setDisplayMessage(
```

```
     int waitTime,
     int messageID
     ref int opStatus);
```

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

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

3.27 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 sendBigBlockData(
     int dataTypeID,
     byte [] data,
     ref 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
opStatus	Integer pointer to receive the command response or operation status. Zero value means OK. For more values, see **Appendix A**.

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

3.28 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.

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 sendBitmap(
     int slot,
     int option,
```

```
      byte [] bitmapData
      ref int opStatus);
```

Parameter	Description
slot	Device bitmap slot, 1 - 4.
option	Options flag: 0 = Clear 1 = Save 2 = Invert and Save
data	Array of bytes containing the bitmap block data to send to the device. The bitmap block data is raw bitmap exclude bitmap header information.
opStatus	An integer array to receive the command response or operation status. Zero value means OK. For more values, see **Appendix A**.

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

3.29 getIPADInfoData

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

```
MagTekPPUSCRAEvent.IPADDevInfo getIPADInfoData();
```

Return Value: `MagTekPPUSCRAEvent.IPADDevInfo`, structured as follows:

```
public struct IPADDevInfo
{
      public string Model;
      public string DevicePath;
      public string Serial;
      public string FWVersion;
      public int Version;
      public int PID;
      public int VID;
      public void clear();

     public override string ToString();
}
```

3.30 requestDeviceInformation

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

```
String requestDeviceInformation(
      int mode,
      ref int opStatus);
```

Parameter	Description
mode	ID for information the device should return: 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
opStatus	An integer array to receive the command response or operation status. Zero value means OK. For more values, see **Appendix A**.

Return Value:
If successful, the function returns a String containing device information. On error it will return null.

3.31 requestDeviceStatus

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

```
DEV_STATE_STAT requestDeviceStatus(ref int opStatus);
```

Parameter	Description
opStatus	An integer array to receive the command response or operation status. Zero value means OK. For more values, see **Appendix A**.

Return Value:
Returns an object of the `DEV_STATE_STAT` class below.

```
public struct DEV_STATE_STAT
{
     public byte nDeviceState;
     public byte nSessionState;
     public byte nDeviceStatus;
     public byte nDevCertStatus;
     public byte nHWStatus;
     public byte nICCMasterSessKeyStatus;
}
```

3.32 requestKernelInformation

This function retrieves the device's kernel information.

```
int requestKernelInformation(
```

```
    int kernelInfoID,
    byte [] kernelInfoBuffer);
```

Parameter	Description
kernelInfoID	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.

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

3.33 getBINTableData

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

```
byte[] getBINTableData();
```

Return Value: A byte array of device BIN data.

3.34 setBINTableData

This function retrieves the BIN table data. See report `0x32`. For additional information, see **Appendix B EMV CBC-MAC**.

```
int setBINTableData(byte[]binTable, byte reserved, ref int opStatus);
```

Parameter	Description
binTable	Buffer pointer to the BIN table byte array
reserved	RFU
opStatus	Byte array to receive the command response or operation status. Zero value means OK. For more values, see **Appendix A**.

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

3.35 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.

```
String getKSN();
```

Return Value: 20-digit hexadecimal key serial number or an empty string.

3.36 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 requestCard(
      int waitTime,
      int displayMessage,
      int beepTones
      String lpFieldSep);
```

Figure 3-3 - DynaPro Swipe Prompts

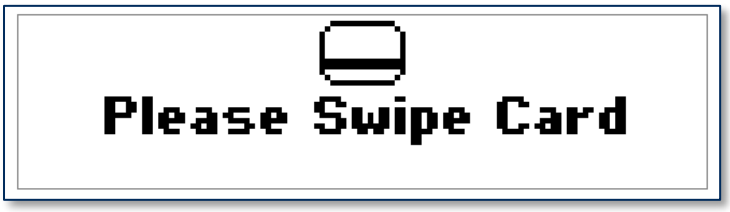

Figure 3-4 - DynaPro Mini Initial Swipe Prompt

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
beepTones	Tone to use: 0x00 – No Sound 0x01 – Single Beep 0x02 – Double Beeps
lpFieldSep	Delimiter to separate the output data

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

3.37 requestManualCardData

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

```
int requestManualCardData(
     int waitTime,
     int beepTones,
     int options,
     ref 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
options	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 QwickCodes 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.
opStatus	An integer pointer to receive the command response or operation status. Zero value means OK. For more values, see **Appendix A**.

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

3.38 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 requestUserDataEntry(
     int waitTime,
     int displayMessageID,
     int beepTones,
```

```
      ref 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
opStatus	An integer pointer to receive the command response or operation status. Zero value means OK. For more values, see **Appendix A**.

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

3.39 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 requestResponse(
      int waitTime,
      int selectMsg,
      int keyMask,
      int beepTones);
```

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

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

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

3.40 confirmAmount

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

```
int confirmAmount(
      int waitTime,
      int tones);
```

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 a value (0: Success, Non-Zero: Error)

3.41 selectCreditDebit

This function prompts the user to confirm the card type. The event associated with this function is **OnKeyInput**.

```
int selectCreditDebit(
      int waitTime,
      int beepTones);
```

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

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

3.42 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 **OnPINRequestCompleteEvent**.

```
int requestPIN(
      int waitTime,
      int pinMode,
      int maxPINLength,
      int minPINLength,
      int beepTones,
      int option
      String lpFieldSep);
```

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
maxPINLength	Minimum PIN length. Must be greater than 3.
minPINLength	Maximum PIN length. Must be less than 13.
beepTones	Tone to use: 0 = No sound 1 = Single beep 2 = Double beep

Parameter	Description
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
lpFieldSep	Delimiter to separate the data

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

3.43 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 **OnSignatureArrive**.

```
int 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 retrieved if exists

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

3.44 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 EMVAcqResponseCompleteEventHandler 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 event associated with this command is **OnEMVComplete**.

```
int requestSmartCard (
     int cardType
     int comfirmationTime,
     int pinEnteringTime,
     int beepTones,
     int option,
```

```
     byte [] Amount,
     int transactionType,
     byte [] cashback,
     byte [] reserved);
```

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 6 – Contactless smart card + contact smart card 7 – Magnetic stripe + contact smart card + contactless smart card.
confirmationTime	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.
reserved	29-byte array reserved for future use

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

3.45 sendAcquirerResponse

This function sends the Acquirer report to the device. For additional information, see **Appendix B EMV CBC-MAC**.

```
int sendAcquirerResponse(
     byte [] responseData,
     int responseDataLength);
```

Parameter	Description
responseData	Byte array to contain the Acquirer Response data
responseDataLength	responseData length in bytes

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

3.46 getCardDataInfo

This function returns card data.

```
MagTekPPUSCRAEvent.CARD_DATA_INFO getCardDataInfo();
```

Return Value:
`MagTekPPUSCRAEvent.CARD_DATA_INFO` structured as follows:

```
public struct CARD_DATA
{
      public byte CardOperationStatus;
      public byte CardOperationStatus;
      public byte CardStatus;
      public byte CardType;
      public byte DataType;
      public byte EncMPLength;
      public byte EncMPStatus;
      public byte EncTrack1Length;
      public byte EncTrack1Status;
      public byte EncTrack2Length;
      public byte EncTrack2Status;
      public byte EncTrack3Length;
      public byte EncTrack3Status;
      public byte MSStatus;
      public uint reserved;
      public byte Track1Length;
      public byte Track1Status;
      public byte Track2Length;
      public byte Track2Status;
      public byte Track3Length;
      public byte Track3Status;

      public string CBCMAC { get; }
      public string EncMP { get; }
      public string EncTrack1 { get; }
      public string EncTrack2 { get; }
      public string EncTrack3 { get; }
      public string KSN { get; }
      public string MPSTS { get; }
      public string Track1 { get; }
      public string Track2 { get; }
```

```
        public string Track3 { get; }

        public void alloc(int size);
        public void clear();
        public void free();
        public string getExpDate();
        public string getFirstName();
        public string getLastName();
        public string getMiddleName();
        public string getPAN();
        public string ToSeparatedString(string separator);
        public override string ToString();
}
```

3.47 requestDeviceConfiguration

This function retrieves the device configuration.

```
byte [] requestDeviceConfiguration( ref int  opStatus);
```

Parameter	Description
opStatus	An integer value to receive the command response or operation status. Zero value means OK. For more values, see **Appendix A**.

Return Value:
Returns the current device configuration, which is an array of bytes.

3.48 requestDeviceConfigurationForInformation [Deprecated: V1.04]

This function retrieves the device configuration.

```
byte [] requestDeviceConfiguration( ref int  opStatus);
```

Parameter	Description
opStatus	An integer value to receive the command response or operation status. Zero value means OK. For more values, see **Appendix A**.

Return Value:
Returns the current device configuration, which is an array of bytes.

3.49 getProductID

This function returns the device's product identifier.

```
String getProductID();
```

Return Value:
Returns a null terminated string. For example - "3004"

3.50 getDeviceSerial

This function returns the device's serial number.

```
String getDeviceSerial();
```

Return Value:
Returns a null terminated string. For example - "12345678"

3.51 getDeviceModel

This function returns the device's model number.

```
String getDeviceModel();
```

Return Value:
Returns a null terminated string. For example, "DynaPro SC."

3.52 getDeviceFirmwareVersion

This function returns the device's firmware revision number.

```
String getDeviceFirmwareVersion();
```

Return Value:
Returns a null terminated string.

3.53 isDeviceConnected

This function returns the device connection status.

```
boolean isDeviceConnected ();
```

Return Value:
`True` if the device is attached to computer and device is opened.
`False` if the device is not attached to computer.

3.54 getPINKSN

This function returns a 20-hexadecimal key serial number string after the completion of the **requestPIN** operation.

```
String getPINKSN ( );
```

Return Value: String

3.55 getDeviceConnected

This function returns the device connection status.

```
int getDeviceConnected ();
```

Return Value:
1 if the device is attached to a computer.
0 if the device is not attached to computer.

3.56 getSessionState

This function gets the device session state. The value is valid after calling **requestDeviceStatus**.

```
int getSessionState ( );
```

Return Value: Positive value of session state. For details, see the "Status and Messages" appendices in ***99875585 DynaPro Programmer's Reference (Commands)*** and/or ***99875629 DynaPro Mini Programmer's Reference (Commands)***.

3.57 requestClearTextUserDataEntry

This function sends the Clear Text User Data Entry report to the device that support 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 **requestClearTextUserDataEntry**, but data will displayed and returned in clear text. The event associated with this command is **OnClearTextUserDataEntry**,

```
int requestClearTextUserDataEntry(
     byte waitTime,
     byte displayMessageID,
     byte 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 updateFirmware

This function sends a PIN Encryption Device firmware data to update the firmware.

```
int updateFirmware ( byte[] firmwareData, ref int opStatus );
```

Parameter	Description
firmwareData	Binary data of firmware.
opStatus	An integer value to receive the command response or operation status. Zero value means OK. For more values, see **Appendix A**.

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

4 MTPPSCRANET Delegate

If you are using the library, after calling the functions in section 3 **MTPPSCRANET Functions**, the MTPPSCRANET SDK libraries for will invoke the callback functions in this chapter to provide the requested data and/or a detailed response. Custom software that uses the MTPPSCRANET libraries should create an object that implements the following functions to process the returning data, then register it as a delegate. For sample code that demonstrates how to use these functions, see `MTPPSCRACS Demo` in the SDK files.

4.1 OnErrorEvent

```
public delegate void OnErrorEvent(int errorCode);
```

Parameter	Description
errorCode	An integer error code for an error handler.

4.2 OnDataReadyCompleteEvent

Return event for **sendSpecialCommand.**

```
Public delegate void OnDataReadyCompleteEvent(byte[] lpData);
```

Parameter	Description
lpData	A response string for sendSpecialCommand function.

4.3 OnPowerUpICCCompleteEvent

Return event for **requestPowerUpResetICC (EMV L1 only)**.

```
public delegate void OnPowerUpICCCompleteEvent(byte status, byte[]
emvData);
```

Parameter	Description
status	Status code
emvData	EMV response byte array

4.4 OnAPDUArrivedCompleteEvent

Return event for **requestICCAPDU (EMV L1 only)**.

```
Public delegate void OnAPDUArrivedCompleteEvent(byte status, byte[]
RAPDU);
```

Parameter	Description
Status	Status code
RAPDU	Response APDU byte array from the device.

4.5 OnGetCAPublicKeyCompleteEvent

```
public delegate void OnGetCAPublicKeyCompleteEvent(byte status, byte[]
key);
```

Parameter	Description
status	Status code
key	CA public key byte array

4.6 OnEMVTagsCompleteEvent

```
public delegate void OnEMVTagsCompleteEvent(byte status, byte[]
tagResp);
```

Parameter	Description
status	Status code
tagResp	A byte array to contain the response for the requested tag

4.7 OnPINRequestCompleteEvent

Upon completion of a **requestPIN** call, the SDK will call both versions of onPINRequestComplete. Developers may choose to use this form, which receives the return data as a string, or the other form, which receives the return data as a structure.

```
public delegate void OnPINRequestCompleteEvent(String lpData);
```

Parameter	Description
lpData	A response string for **requestPIN** function. It is comma delimited and contains the KSN(20 chars), EBP(16 chars), and opStatus(2 chars).

4.8 OnKeyInputCompleteEvent

Response event for **requestResponse**, **confirmAmount**, and **selectCreditDebit**.

```
public delegate void OnKeyInputCompleteEvent(byte status, byte key);
```

Parameter	Description
status	Status code.
key	Key pressed value.

4.9 OnDisplayRequestCompleteEvent

Return event for **setDisplayMessage**.

```
public delegate void OnDisplayRequestCompleteEvent (int lpData);
```

Parameter	Description
lpData	Zero is returned.

4.10 OnSignatureArriveCompleteEvent

Response event for **requestSignature**.

```
public delegate void OnSignatureArriveCompleteEvent (byte status,
byte[] signature);
```

Parameter	Description
status	Status code.
signature	Signature byte array.

4.11 OnCardRequestCompleteEvent

Return event for **requestCard**. After receiving the return String data, software may also call **getCardDataInfo** to parse the array into a structure.

```
public delegate void OnCardRequestCompleteEvent(String lpData);
```

Parameter	Description
lpData	A response string for the **requestCard** function.

4.12 OnUserDataEntryCompleteEvent

Return event for **requestUserDataEntry**.

```
public delegate void OnUserDataEntryCompleteEvent(USER_ENTRY_DATA
userEntrydata);
```

Parameter	Description
lpData	See USER_ENTRY_DATA below.

```
public struct CLEAR_TEXT_USER_ENTRY_DATA
{
     public byte OpStatus; //Operation Status
     public byte UserDataMode;
     public byte DataLen;
     public string Data;
     public override string ToString();
}
```

4.13 OnDeviceStateUpdateCompleteEvent

Response to **requestDeviceStatus**.

```
public delegate void OnDeviceStateUpdateCompleteEvent (DEV_STATE_STAT
deviceStateInfo);
```

Parameter	Description
deviceStateInfo	See the `DEV_STATE_STAT` class definition in section **3.31 requestDeviceStatus** for details.

4.14 OnEMVCompleteDataCompleteEvent

Response event for **requestSmartCard**

```
public void OnEMVDataComplete(byte status, byte[] emvData);
```

Parameter	Description
Status	Status code
emvData	EMV response byte array

4.15 OnCardHolderStateChangeCompleteEvent

```
public delegate void OnCardHolderStateChanged(int stateId);
```

Parameter	Description
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

4.16 OnEMVTransactionCompleteEvent

```
public delegate void OnEMVTransactionCompleteEvent(byte status, byte[]
data);
```

Parameter	Description
status	Status code.
data	EMV response byte array.

4.17 OnClearTextUserDataEntryCompleteEvent

Return event for **requestClearTextUserDataEntry**.

```
public delegate void OnClearTextUserDataEntryCompleteEvent
(CLEAR_TEXT_USER_ENTRY_DATA userEntryData);
```

Parameter	Description
lpData	See class definition below.

```
public struct CLEAR_TEXT_USER_ENTRY_DATA
{
     public byte OpStatus; //Operation Status
     public byte UserDataMode;
     public byte DataLen;
     public string Data;
     public void clear();
     public override string ToString();
}
```

4.18 OnProgressUpdateEvent

Progress update event for SendBitmap and UpdateFirmware.

```
public delegate void OnProgressUpdateEvent(byte opStatus, int
UpdateItem, double updateProgress);
```

Parameter	Description
opStatus	Operation status
UpdateItem	0x0C – SendBitmap 0x17 – UpdateFirmware
updateProgress	From 0.0 to 1.0. 1.0 means sending data completed

4.19 OnPayPassMessageEvent

Response event for **requestSmartCard**

```
public delegate void OnPayPassMessageEvent(byte[] Message);
```

Parameter	Description
data	EMV response byte array.

Appendix A Status Codes

A.1 Library Status 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

A.2 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

A.3 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

Appendix B EMV CBC-MAC

For additional information about EMV-related functions for use with L1 or L2 devices, see one of the following documents, available from MagTek:

- ***99875585 DynaPro Programmer's Reference (Commands)***
- ***99875629 DynaPro Mini Programmer's Reference (Commands)***

Appendix C Cryptography

C.1 Decrypt PIN

PIN_DATA structure contains the information need to decrypt PIN.

```
public struct _PIN_DATA
{
      public byte OpStatus;
      public string KSN;
      public string EPB;
}
```

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.

```
byte[] bKSN = ConvertHexStringToByteArray(PinData.KSN);
byte[] bEPB = ConvertHexStringToByteArray(PinData.EPB);
```

Then, derive key from BDK and KSN

```
byte[] bPinKey;

// 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.

```
byte[] bIsoPinBlock = new byte[8];
byte[] iv = new byte[8];

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.

```
public struct _CARD_DATA
{
      public byte DataType;
      public byte CardOperationStatus;
      public byte CardStatus;
      public byte CardType;

      public byte Track1Length;
      public byte Track2Length;
      public byte Track3Length;
      public byte EncTrack1Length;
      public byte EncTrack2Length;
      public byte EncTrack3Length;
      public byte EncMPLength;

      public byte Track1Status;
      public byte Track2Status;
      public byte Track3Status;
      public byte EncTrack1Status;
      public byte EncTrack2Status;
      public byte EncTrack3Status;
      public byte EncMPStatus;
      public byte MSStatus;

      public string MPSTS;
      public string Track1;
      public string Track2;
      public string Track3;
      public string EncTrack1;
      public string EncTrack2;
      public string EncTrack3;
      public string EncMP;
      public string KSN;

      public string CBCMAC;
      public string SerialNumber;
      public byte PANInfoLength;
      public string PANInfo;

      public UInt32 reserved;
}
```

C.2.1 Get Track binary from CARD_DATA

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

```
byte[] bKSN = ConvertHexStringToByteArray(CardData.KSN);
byte[] bEncTrack1 = ConvertHexStringToByteArray(CardData.EncTrack1);
```

```
byte[] bEncTrack2 = ConvertHexStringToByteArray(CardData.EncTrack2);
byte[] bEncTrack3 = ConvertHexStringToByteArray(CardData.EncTrack3);
```

C.2.2 Get Key from KSN

```
byte[] bDataKey;

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

C.2.3 Use Triple DES CBC to decrypt track data

Decrypt track data, use empty initial vector.

```
byte[] iv = new byte[8];
byte[] bDecTrack1 = new byte[bEncTrack1.Length];

TDES_Decrypt_CBC(bPinKey, iv, bEncTrack1, bEncTrack1.Length,
bDecTrack1, bDecTrack2);

Array.Resize(bDecTrack1, CardData.Track1Length);
```

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

```
byte[] bDataKey;
byte[] IV = new byte[8];

// 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 = ((Data.Length + 7) & (~7));
byte[] bDataPadded = new byte[nPaddedLength];
Array.Copy(Data, bDataPadded, Data.Length);
```

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.

```
byte[] pLeftKey = new byte[8];
byte[] pRightKey = new byte[8];

Array.Copy(bDataKey, pLeftKey, 8);
Array.Copy(bDataKey, 8, pRightKey, 0, 8);

byte[] bDataOutput = new byte[nPaddedLength];
```

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

// bDataOutput contain the encrypted data

byte[] pLastBlock = new byte[8];
Array.Copy(bDataOutput, nPaddedLength-8, pLastBlock, 0, 8);
byte[] MAC = new byte[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.

Array.Resize(MAC, 4);
```

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, out bOutMessage, out
nOutMessage);

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

SetCAPublicKeyWithOperation(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 = new byte[6]{0x0,0x0,0x0,0x01,0x0,0x0};
byte[] Cashback = new byte[6] {0x0,0x0,0x0,0x0,0x0,0x0};

int retCode = ipad.requestSmartCard
(ContactSmartcard,20,20,1,2,Amt,4,Cashback),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 onEMVDatatComplete(byte[] data)
{
      // for acquirer data
      TLVData tlv = new TLVData(data);

      Byte[] KSN = tlv.GetKSN();
}
```

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

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

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

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

Array.Copy(KSN,0,Msg,10,10);

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

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

Array.Copy(SerialNumber, 0, Msg, 29, 8);

// Set Data
```

```
Msg[37] = 0xFA;
Msg[38] = 0x82;
Msg[39] = 0;
Msg[40] = 6;

Array.Copy(approve, 0, Msg, 41, 6);

byte[] OutputMsg;
int OutputMsgLen;

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

ipad.sendAcquirerResponse(OutputMsg, 52);
```

C.5.4 Device: onEMVTransactionComplete

Host will receive callback event onEMVTransactionComplete. In the callback, TLV data includes the transaction result.

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 start with create 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
```

```
byte[] 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 };
byte[] 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 };
byte[] 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, a 46 bytes data will save to buffer.
byte[] buffer = ipad.requestChallengeAndSession();

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

byte[] iv = {0,0,0,0,0,0,0,0};


byte[] token = SubArray(buffer, 2,32)
TDES_Decrypt_CBC(bAMKSessionKey, iv, token, 32, SessionInfo, 32);

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

SessionKey = SubArray(SessionInfo,8,24);
// derive session cmac key
SessionCMACKey = bitXor(SessionKey, derivedSessionCMAC_Mask)

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

byte[] transformedNumberSerial = new byte[8];
SetInt32(transformedNumberSerial, rndNumber);

ArrayCopy(SessionInfo,0, transformedNumberSerial,4, 4);

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

// Calculate CMAC
byte[] cmac = new byte[8];
```

```
CMAC(bAMKCMACKey, trns, 8, cmac, 8);

byte[] ernd = SubArray(trns,0,4);
byte[] eserial = SubArray(trns,4,4);

retCode = ipad.requestConfirmSession(1, ernd, eserial, cmac, ref
opStatus);
```

D.3 Power Up ICC Card and Get ATR

Host uses requestPowerUpResetICC to power up smart card and get the card ATR by event.

```
void myATRReceivedCallback (byte status, byte[] eATR)
{
     // Decrypt Secured ATR by SessionKey
     TDES_Decrypt_CBC(SessionKey, iv, eATR, eATR.Length,
ATRBuffer,ATRBufferLen);

}

ipad.onPowerUpICC += myATRReceivedCallback;

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

// you will receive callback in myATRReceivedCallback
```

D.4 Send APDU to Card and Get Response

Host uses requestICCAPDU to communicate with card and get card returned APDU by event.

```
void myApduArrived (byte opStatus, byte[] securedAPDU)
{
      // Decrypt Secured APDU-R by SessionKey
      TDES_Decrypt_CBC(SessionKey, iv, securedAPDU, securedAPDU.Length,
APDUBuffer, APDUBufferLen);
     // byte 0 is the length of APDU returned
     backApdu = SubArray(APDUBuffer, APDUBuffer[0]);
}


// Set APDU Callback
ipad.OnAPDUArrived += myApduArrived;

// Encrypt APDU by Session Key
TDES_Encrypt_CBC(SessionKey, iv, Apdu, ApduLength, EncApduBuffer,
EncApduBufferLen);
// Generate CMAC for this APDU
byte[] cmac[8] = new byte[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 = ipad.requestICCAPDU(EncApduBuffer, EncApduBufferLen);

// you will receive callback in myApduArrived
```

D.5 Power Down ICC

Host uses requestPowerDownICC to power down card.

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

D.6 End L1 Session

Host uses endL1Session to close the secure communication

```
retCode = ipad.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
isDeviceOpened	YES	YES	YES	YES
deviceReset	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
requestICCAPDU	NO	YES	YES	YES
sendSpecialCommand	YES	YES	YES	YES
getSpecialCommand	YES	YES	YES	YES
requestGetEMVTags	NO	YES	YES	YES
requestSetEMVTags	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
requestDeviceStatus	YES	YES	YES	YES
requestKernelInformation	NO	YES	YES	YES

Appendix E - Function Applicable Table

Function Name	IPAD	DYNAPRO	DYNAPRO MINI	DYNAPRO GO
getBINTableData	NO	YES	YES	YES
setBINTableData	NO	YES	YES	YES
getKSN	YES	YES	YES	NO
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
requestDeviceConfiguration	YES	YES	YES	YES
getProductID	YES	YES	YES	YES
getDeviceSerial	YES	YES	YES	YES
getDeviceModel	YES	YES	YES	YES
getDeviceFirmwareVersion	YES	YES	YES	YES
isDeviceConnected	YES	YES	YES	YES
getPINKSN	YES	YES	YES	YES
getSessionState	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

Appendix E - Function Applicable Table

Function Name	IPAD	DYNAPRO	DYNAPRO MINI	DYNAPRO GO
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
setParameters	YES	YES	YES	YES
getParameters	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
requestICCAPDUSync	NO	YES	YES	YES
requestPowerUpResetICCSync	NO	YES	YES	YES
sendAcquirerResponseSync	NO	YES	YES	YES
requestUserDataEntrySync	YES	YES	YES	YES
requestSignatureSync	YES	YES	NO	RF
requestResponseSync	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

Appendix E - Function Applicable Table

Function Name	IPAD	DYNAPRO	DYNAPRO MINI	DYNAPRO GO
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
getAMKInfo	YES	YES	YES	YES
getKeyInfo	YES	YES	YES	YES

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