MAGTEK®
SECURITY FROM THE INSIDE

IPAD, DynaPro, DynaPro Go, and DynaPro Mini

PIN Encryption Devices

Programmer's Reference (Java/Java Applet)

March 2023

Manual Part Number:
D99875633-180

REGISTERED TO ISO 9001:2015

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	Mar 14, 2014	Initial release
2.01	April 21, 2014	Fix sendBitmap that require to have opStatus parameter and add more detail description to data parameter.
3.01	April 23, 2014	Add getDeviceList API Add minimum JRE Requirement
4.01	May 01, 2014	Fix onDataReady callback is byte array instead of string.
5.01	May 28, 2014	Add BleConn.dll, remove MacOS X (Firefox) Add Windows 8.1 support. Add new appendix CBC-MAC reference information and change appendix B to C
6.01	July 18, 2014	Changed BleConn.Dll to MTPPBLE.DLL
70	January 20, 2015	Add Appendix D, Cryptography and how to handle L2 device Fix appendix B, where it point to wrong document. Add appendix E, Function Applicable table Update SendBitmap, bitmap size. Add getNextMSRKSN function. Update System Requirements. Change getDeviceConnected to getDeviceConnectedState. Added IPAD to supported device list. Update requestSmartcard Card Type having contactless. Update Appendix E with Contact Smart Card L1 Session. Move appendix E to appendix F. Update SendBitmap description with additional reference information.

80	February 6, 2015	Update instructions for setting up MagTek PCI PED Java Demo.
90	April 21, 2015	Add instructions on how to modify manifest and sign JAR.
100	May 27, 2015	Add new function: updateFirmware, requestGetEMVTags, requestSetEMVTags added. requestGETEMVTags and requestSetEMVTags are overloaded and having database and reserved parameters.
110	March 1, 2016	Replace reference of EMVAcqResponseCompleteEventHandler with onEMVDataComplete
120	April 25, 2016	Update requestGetEMVTags and requestSetEMVTags. Add requestSetEMVTagsEx.
130	March 6, 2017	Add support for DynaPro Go. Add note to getKSN function of its usage for token reversal.
140	August 21, 2017	Add loadClientCertificate.
150	September 10, 2018	Update requestSmartCard to increase reserve size from 29 bytes to 45 bytes. Add RequestTipOrCashback, GetSelectedMenuItem, DeviceMessageTipOrCashback, and DeviceMessageSelectedMenuItem, and TLS1.2 connection to openDevice.
160	February 26, 2019	Updated to correctly reference Bluetooth LE.
170	December 15, 2022	Added requestPINExt function.
180	March 1, 2023	Added description for including PAN to requestPINExt function at section 3.

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.

DEMO SOFTWARE / SAMPLE CODE: Unless otherwise stated, all demo software and sample code are to be used by Licensee for demonstration purposes only and MAY NOT BE incorporated into any production or live environment. The PIN Pad sample implementation is for software PIN Pad test purposes only and is not PCI compliant. To meet PCI compliance in production or live environments, a third-party PCI compliant component (hardware or software-based) must be used.

Table of Contents

1 Introduction

This document provides instructions for software developers who want to create software solutions that include a IPAD, DynaPro, DynaPro Go, or DynaPro Mini connected to a Windows-based host via USB or Bluetooth LE. 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 Java Demo

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

1.2 About the MagTek PIN Pad SCRA Java Applet Demo

The PCI PED Java applet, available from MagTek, provides demonstration source code and a reusable PCI PED Java applet that provides developers of custom HTML / JavaScript software solutions with a set of functions that parallels the functionality to the Java library, in applet form.

1.3 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**.

1.4 SDK Contents

File name	Description
MTPPSCRADemo.java	This is the PCI PED Java Demo sample code.
magtek-ppscra-lib.jar	This is the MTPPSCRA Java library.
MTPPSCRA.dll	This is a native DLL to be copied to the system folder.
MTPPSCRAJ.dll	This is a native DLL to be copied to the system folder.
MTPPBLE.dll	DLL require to interact with Bluetooth LE device. For Bluetooth LE to work, Windows 8 and Above is required.
Build.bat	This .bat file builds the PCI PED Java Demo.
Test.bat	This.bat file launches the PCI PED Java Demo software.
magtek-ppscra-applet.jar	This .jar file contains the implementation of the applet.
magtek-ppscra.html	This sample web page demonstrates how to use the applet.
magtek-ppscra.js	This sample JavaScript code demonstrates how to use the applet.

1.5 System Requirements

1.5.1 Java Library

Tested operating systems:

- Windows 7
- Windows 8, 8.1
- Windows 10

Java Build Platform: JDK 1.7, 32-bit and above.

Minimum Java Runtime requirements: Java 7u51 and above.

Tested Java Runtime Environments: Java 7u51 to 8u181.

1.5.2 Java Applet

Tested operating systems:

- Windows 7
- Windows 8, 8.1
- Windows 10

Tested web browsers:

- Internet Explorer 11
- Firefox (32 bit only) 52 esr to 60 esr

Minimum Java Runtime requirements: Java 7u51 and above.

Tested Java Runtime Environments: Java 7u51 to 8u181.

2 How to Set Up the MagTek PCI PED Java Demo

2.1 How to Download and Set Up the MagTek PCI PED Java Demo

To set up the MTPPSCRANET Libraries, follow these steps:

1) Download the ***DynaPro/DynaPro Go/DynaPro Mini Windows SDK Install***, available from MagTek.com (**Support** > **Software** > **Programming Tools** > **IPAD, DynaPro, and DynaPro Mini** > **IPAD/DynaPro/DynaPro Go/DynaPro Mini Windows API**)
2) Right-click **99510127.exe** and select **Run as administrator**. The installer will place all dependencies in appropriate paths.

2.2 How to Set Up the Java Library With the 32-bit JRE/JVM

MagTek highly recommends using the 32-bit version of Java when using the PCI PED Java applet, regardless of whether you are using a 32-bit or 64-bit version of Windows.

To set up and run the Java Demo software using the 32-bit version of Java on either a 32-bit or 64-bit version of Windows, follow these steps:

1) Uninstall any existing instances of the 64-bit Java Runtime Environment (JRE) or Java Development Kit (JDK). Leaving them installed can cause runtime failures, as the library may fail to load.
2) Download and install the latest version of the 32-bit Java Development Kit (JDK).
3) Follow the steps in section **2.1 How to Download and Set Up the MagTek PCI PED Java Demo** to download and install the latest PCI PED Windows SDK. You may download and install it directly on the target workstation where it will be used, or you may opt to install it on a master development workstation and copy the dependencies to the target workstation manually.
4) If you opted to manually copy the PCI PED Windows SDK dependencies from a master development workstation to the target workstation where it will be used, follow these steps:
   a) On the master workstation, navigate to the root of the PCI PED Windows SDK. By default, it will be **C:\Program Files\MagTek\PCI PED Windows SDK\Dependencies** for 32-bit operating systems, or **C:\Program Files (x86)\MagTek\PCI PED Windows SDK\Dependencies** for 64-bit operating systems.
   b) Open the **\Win32** subfolder and copy all the files to the target workstation's **C:\Windows\System32** folder for x86 systems, or to the target workstation's **C:\Windows\SysWOW64** folder for x64 systems.
5) Connect the device to the workstation using a USB cable. Windows will install the device drivers automatically. Wait for Windows to report the driver installation is complete.
6) Launch a Windows command prompt as an Administrator.
7) **cd** to the root of the folders where the PCI PED Java Demo is installed. By default, it will be **C:\Program Files\MagTek\PCI PED Windows SDK\Sample Code\Java\Object** for 32-bit Windows, or **C:\Program Files (x86)\MagTek\PCI PED Windows SDK\Sample Code\Java\Object** for 64-bit Windows.
8) Type **build.bat** and press **Enter** to build the Java Demo software.
9) Type **test.bat** and press **Enter** to launch the Java Demo software.
10) Use the Java Demo software, and / or continue to the setup steps in section **2.4 How to Set Up the Applet With the 32-bit JRE/JVM**.

2.3 How to Manually Set Up the Java Library With the 64-bit JRE/JVM

MagTek highly recommends using the 32-bit version of Java if you intend to use the PCI PED Java applet as described in section 2.2, regardless of whether you are using a 32-bit or 64-bit version of Windows.

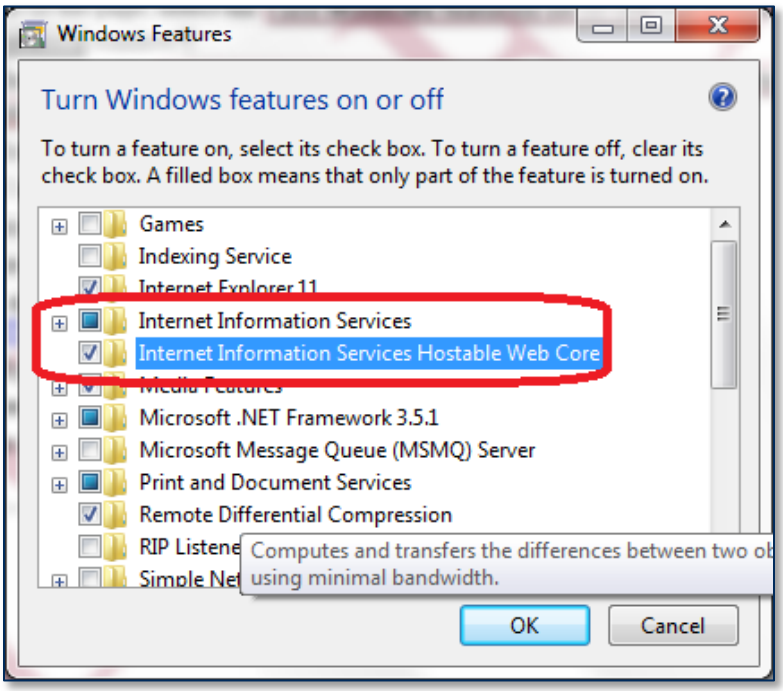
1) Uninstall any existing instances of the 32-bit Java Runtime Environment (JRE) or Java Development Kit (JDK). Leaving them installed can cause runtime failures, as the library may fail to load.
2) Download and install the latest version of the 64-bit Java Development Kit (JDK).
3) Follow the steps in section **2.1 How to Download and Set Up the MagTek PCI PED Java Demo** to download and install the latest PCI PED Windows SDK. You may download and install it directly on the target workstation where it will be used, or you may opt to install it on a master development workstation and copy the dependencies to the target workstation manually.
4) If you opted to manually copy the PCI PED Windows SDK dependencies from a master development workstation to the target workstation where it will be used, follow these steps:
   a) On the master workstation, navigate to the root of the PCI PED Windows SDK. By default, it will be **C:\Program Files (x86)\MagTek\PCI PED Windows SDK\Dependencies**.
   b) Open the **\x64** subfolder and copy all the files to the target workstation's **C:\Windows\System32** folder.
5) Connect the device to the workstation using a USB cable. Windows will install the device drivers automatically. Wait for Windows to report the driver installation is complete.
6) Launch a Windows command prompt as an Administrator.
7) **cd** to the root of the folders where the PCI PED Windows SDK is installed. By default, it will be **C:\Program Files (x86)\MagTek\PCI PED WINDOWS SDK\Sample Code\Java\Object** for 64-bit Windows.
8) Type **build.bat** and press **Enter** to build the Java Demo software.
9) Type **test.bat** and press **Enter** to launch the Java Demo software.
10) Use the Java Demo software.

2.4 How to Set Up the Applet With the 32-bit JRE/JVM

MagTek highly recommends using the 32-bit version of Java when using the PCI PED Java applet, regardless of whether you are using a 32-bit or 64-bit version of Windows.

To set up the Java applet using the 32-bit version of Java on either a 32-bit or 64-bit version of Windows, follow these steps:

1) Follow the steps in section **2.2 How to Set Up the Java Library With the 32-bit JRE/JVM**. Having a working JVM, working Java library, working drivers, and working DLLs are prerequisites for using the applet.
2) Verify Java is installed and that the Internet Explorer Java plugin is working correctly by using Oracle's Java applet test page, usually provided as a link or auto-launch at the end of installation.
3) On the Windows 7 workstation you will use for development, enable Internet Information Services 7 (IIS) as follows:
   a) Log in to a Windows 7 workstation using an administrator account.
   b) Launch the Windows **Control Panel**.
   c) Select the **Programs and Features** item to open the **Programs and Features** page.
   d) On the left side of the page, select the **Turn Windows features on or off** link to launch the **Windows Features** window.
   e) Turn on the checkboxes for **Internet Information Services** and **Internet Information Services Hostable Web Core**.



   f) Press the **OK** button to launch a progress window. Wait for Windows to install IIS.
4) Launch a web browser and navigate to **//localhost**. Verify the IIS default page appears as shown in **Figure 2-1**.

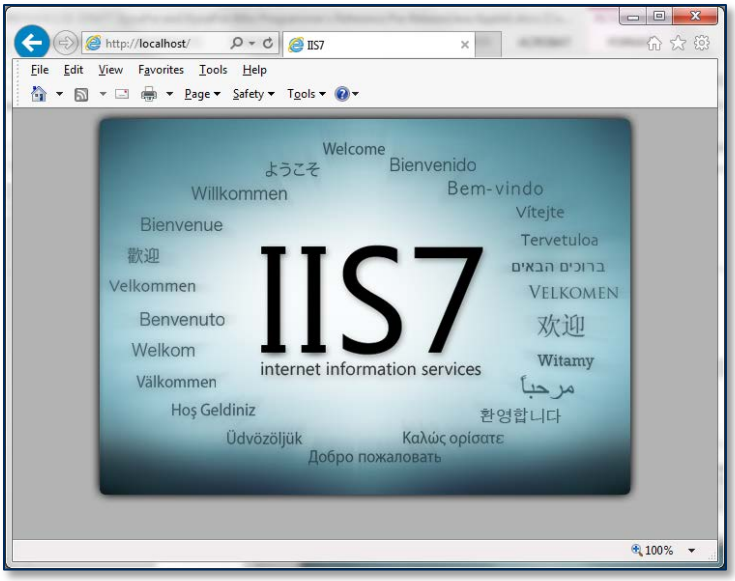


Figure 2-1 - IIS Default Page

5) If it does not already exist, create a **MTPPSCRA** folder in **C:\inetpub\wwwroot**. If it does exist, delete its contents.
6) On the workstation where the PCI PED Windows SDK is installed, navigate to the folder where it is installed. By default, it will be **C:\Program Files\MagTek\PCI PED Windows SDK** for 32-bit operating systems, or **C:\Program Files (x86)\MagTek\PCI PED Windows SDK** for 64-bit operating systems.
7) Open the **Sample Code\Java Applet\Object** subfolder.
8) Copy the contents of the subfolder to **C:\inetpub\wwwroot\MTPPSCRA**.
9) Open the **Sample Code\Java Applet\Object\Win32** subfolder.
10) Copy the contents of the subfolder to **C:\inetpub\wwwroot\MTPPSCRA**.

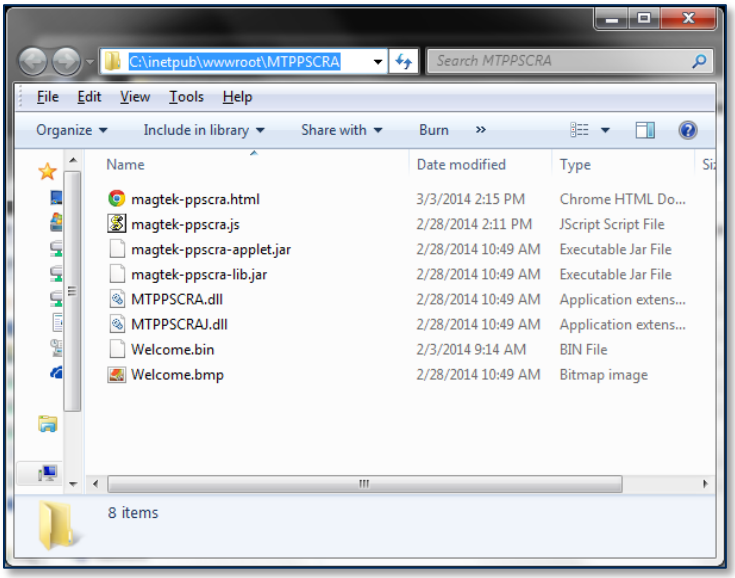
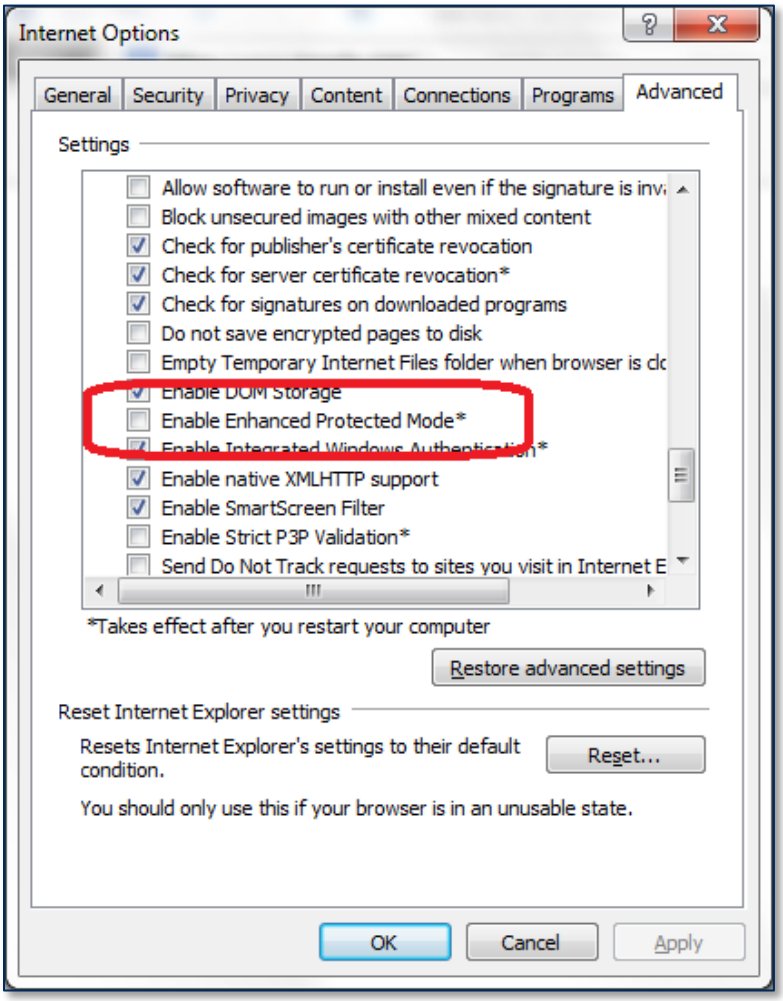
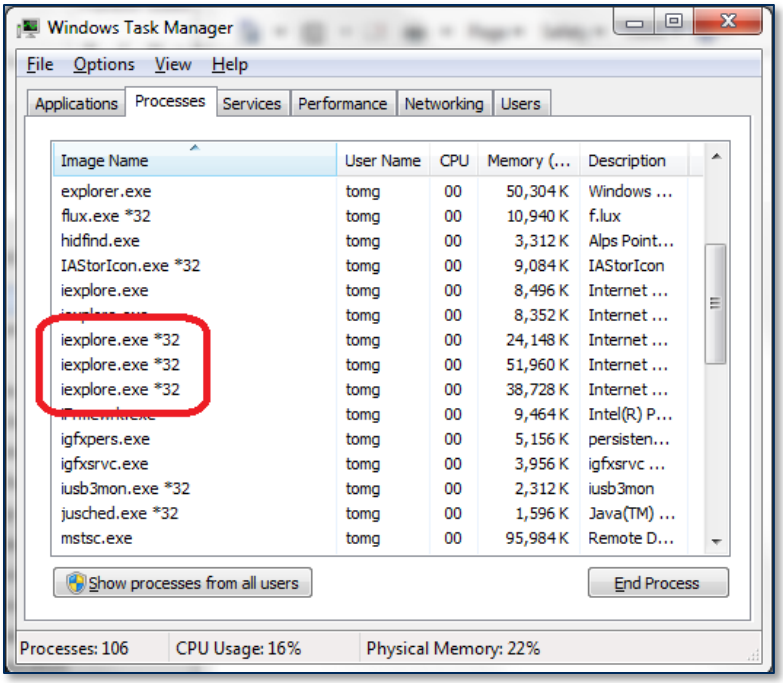


Figure 2-2 - inetpub Structure

11) Connect the device to the workstation using a USB cable. Windows will install the device drivers automatically. Wait for Windows to report the driver installation is complete.
12) Open Internet Explorer as an administrator.
13) If you are using a 64-bit version of Windows with IE8 or IE9, make sure to launch directly in 32-bit mode using the iexplore.exe found in **C:\Program Files (x86)**. Verify you are running in 32-bit mode using the **Help** > **About** menu.
14) If you are running a 64-bit version of Windows with IE10 or higher, choose the **Internet options** that enable 32-bit mode / disable **Protected Mode** for the zone you are accessing. Also turn **OFF** the checkbox for **Enhanced Protected Mode** in the **Internet Options** > **Advanced** tab.

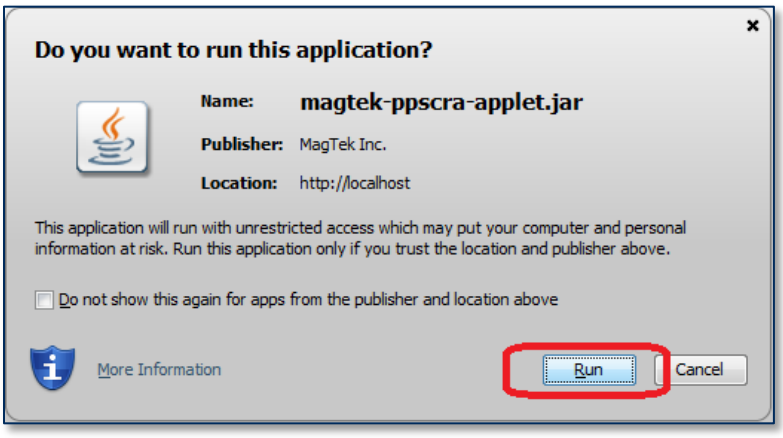


15) If you changed the value of the **Enable Enhanced Protected Mode** checkbox, restart Windows.
16) Open Windows Task Manager (**Ctrl-Alt-Del** > **Start Task Manager**).
17) Open the **Processes** tab and sort by **Image Name**.
18) Note the number and location of all **iexplore.exe *32** processes.
19) In Internet Explorer, navigate to http://localhost/MTPPSCRA/magtek-ppscra.html.
20) In the Windows Task Manager **Processes** tab, find the new process for the Internet Explorer tab you just opened and make sure it is running in 32-bit mode (**iexplore.exe *32** instead of **iexplore.exe**).



21) Close the **Windows Task Manager** window.

22) Internet Explorer will display a welcome page and will pop up a **Do you want to run this application?** window. Press the **Run** button to run the Java applet.



23) On the welcome page, press the **Open Device** button. The large text box in the browser will display the text **openDevice : OK**.

24) Press the **GetDevInfo** button. The large text box in the browser will display device information.

25) Use the buttons and fields on the welcome page to test the connection to the device.

26) After installation on the workstation is complete, future browser sessions do not require the user to launch Internet Explorer as an administrator to use the applet.

2.5 How to Modify Manifest

The **Caller-Allowable-Codebase** attribute is used to identify the domains from which JavaScript code can make calls to your RIA without security prompts. Set this attribute to the domain that hosts the JavaScript code. If a call is made from JavaScript code that is not located in a domain specified by the **Caller-Allowable-Codebase** attribute, the call is blocked. To specify more than one domain, separate the domains by a space, for example:

```
Caller-Allowable-Codebase: *.yahoo.com *.google.com *.magtek.com *
```

The **Application-Library-Allowable-Codebase** attribute identifies the locations where your signed RIA is expected to be found. This attribute is used to determine what is listed in the Location field for the security prompt that is shown to users when the JAR file for your RIA is in a different location than the JNLP file or HTML page that starts your RIA. If the files are not in the locations identified, the RIA is blocked. Set this attribute to the domains where the JAR file, JNLP file, and HTML page are located. To specify more than one domain, separate the domains by a space, for example:

```
Application-Library-Allowable-Codebase: *.yahoo.com *.google.com
*.magtek.com *
```

For more information regarding the JAR File Manifest Attributes for Security, please visit this website http://docs.oracle.com/javase/7/docs/technotes/guides/jweb/security/manifest.html

In order to modify the Manifest file, please follow these steps. You need to do this for both **magtek-ppscra-applet.jar** and **magtek-ppscra-lib.jar**

1) Find installation folder by default, the installation folder is:
   \Sample Code\Java Applet\Object\Unsigned
2) Launch the command prompt and extract the META-INF/MANIFEST.MF from the jar file.
   ```
   jar xf magtek-ppscra-applet.jar  META-INF/MANIFEST.MF
   ```
3) Open **MANIFEST.MF** and look for the **Caller-Allowable-Codebase** and **Application-Library-Allowable-Codebase** and add your website URL to the list like the example above.
4) Update the manifest to the jar file.
   ```
   jar umf META-INF/MANIFEST.MF magtek-ppscra-applet.jar
   ```

2.6 How to Sign JAR

These instructions provide an overview of obtaining and using Sun Java signing and a digital certificate. Please follow this instruction to sign and verify both **magtek-ppscra-applet.jar** and **magtek-ppscra-lib.jar**

1) Make sure your machine has the latest Java JDK installed.
2) Generate a public/private key pair by entering the following command, specifying an alias for your keystore:
   ```
   keytool -genkey -keyalg rsa -alias MyCert
   ```
3) Generate a certificate signing request (CSR) by entering the following command:
   ```
   keytool -certreq -alias MyCert
   ```

After prompting you to enter the password for your keystore, keytool will generate a CSR.

4) Save the certificate received from the Certificate provider as Certname.p7b.
5) Import your Digital Certificate by entering the following command:

```
keytool -import -alias MyCert -file Certname.p7b
```

In this string, keytool is requested to import the Digital ID “Certname.cer” into the keystore MyCert.

6) Bundle your applet into a Java Application Resource (JAR) file by entering the following command:

```
jar cvf C:\Magtek-ppscra-applet.jar
```

7) Sign your applet by using jarsigner to sign the JAR file, using the private key you saved in your keystore:

```
jarsigner C:\Magtek-ppscra-applet.jar MyCert
```

8) Verify the output of your signed JAR file by entering the following command:

```
jarsigner -verify -verbose -certs C:\Magtek-ppscra-applet.jar
```

Please visit this website https://docs.oracle.com/javase/tutorial/deployment/jar/signing.html for more information regarding signing JAR files.

3 MTPPSCRA Functions

If you are developing Java software, follow the setup steps in section **How to Set Up the MagTek PCI PED Java Demo**, then create an instance of the MTPPSCRA object in your software project, then use Java 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 `MTPPSCRADemo.java` in the SDK files.

If you are developing HTML / JavaScript software using the Java applet, follow the setup steps in section **How to Set Up the MagTek PCI PED Java Demo**, create an instance of the applet in your HTML, then use JavaScript to invoke the functions described in this chapter to communicate with the device. For sample code that demonstrates how to use these functions, see the `magtek-ppscra.html` and `magtek-ppscra.js` sample code 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 **MTPPSCRA Events**.
- **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 Java 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. The event associated with this command is **onDeviceConnectionStateChanged**.

```
long openDevice();
```

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

3.3 openDevice

This function opens a connection to the device. The event associated with this command is **onDeviceConnectionStateChanged**.

```
long 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`
	example:
	`IP://10.57.10.180:26`
	
	For TLS1.2 Wireless devices, deviceURI should be in the form:
	`TLS12://TLS(DeviceSerialNumber)`
	`TLS12TRUST://TLS(DeviceSerialNumber)`
	example:
	`TLS12TRUST://TLS98AF14220612070C`
	`TLS12://TLS98AF14220612070C`
	
	For Bluetooth LE devices, deviceURI should be in the form:
	`BLEEMV://(DeviceInstanceName)`
	example:
	`BLEEMV://Dynapro Go-EB58`

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

3.4 closeDevice

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

```
long closeDevice();
```

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

3.5 getDeviceList

This function enumerate all IPAD.

```
String getDeviceList();
```

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

3.6 isDeviceOpened

This function retrieves the device’s open status.

```
boolean isDeviceOpened();
```

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

3.7 deviceReset

This function sends a reset command to the device.

```
long deviceReset();
```

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

3.8 getStatusCode

This function retrieves the current status of the issued report.

```
long getStatusCode();
```

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

3.9 cancelOperation

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

```
long cancelOperation();
```

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

3.10 requestBypassPINCommand

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

```
long requestBypassPINCommand();
```

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

3.11 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.

```
long 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)

3.12 setAmount

This function sets the transaction amount before beginning a transaction.

```
long setAmount(
     byte amountType,
     String lpAmount,
     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)

3.13 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).

```
long endSession(int displayMessageID);
```

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)

3.14 requestChallengeAndSessionForInformation (EMV L1 only)

This function retrieves the challenge key and session key for a smart card transaction. For additional information, see appendix B.

```
byte[] requestChallengeAndSessionForInformation();
```

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

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

```
long requestConfirmSessionWithMode(
     int mode,
     byte [] encryptedRandomNumber,
     byte [] encryptedSerialNumber,
     byte [] 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
CMAC Length	Length of CMAC

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)

3.16 requestConfirmSession (EMV L1 only)

This function sends a CMAC message to the device to confirm the session key. For additional information, see appendix B.

```
long requestConfirmSession(
     int mode,
     byte [] encryptedRandomNumber,
     byte [] encryptedSerialNumber,
     byte [] 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
CMAC Length	Length of 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)

3.17 endL1Session (EMV L1 only)

This function ends the session with the device.

```
long endL1Session(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)

3.18 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**.

```
long requestPowerUpResetICCWithWaitTime(
     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)

3.19 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**. For additional information, see appendix B.

```
long 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)

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

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

```
long requestPowerDownICCWithWaitTime(byte waitTime);
```

Parameter	Description
waitTime	Not used.

Return Value:

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

3.21 requestPowerDownICC (EMV L1 only)

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

```
long requestPowerDownICC(byte waitTime);
```

Parameter	Description
waitTime	Not used.

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

3.22 requestICCAPDUForInformation (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.

```
long requestICCAPDUForInformation(
     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)

3.23 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 Reference (Commands)*** and/or ***99875629 DynaPro Mini Programmer's Reference (Commands)***. The event associated with this command is **onDataReady**.

```
long sendSpecialCommandWithCommand(String lpCommand);
```

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

Return Value:
Returns a 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 Reference (Commands)*** and/or ***99875629 DynaPro Mini Programmer's Reference (Commands)***. The event associated with this command is **onDataReady**.

```
long sendSpecialCommand(String lpCommand);
```

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

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

3.25 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)***.

```
byte[] getSpecialCommandWithCommand (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.

3.26 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)***.

```
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.

3.27 requestGetEMVTagsWithTagType [Deprecated: V5.01]

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

```
long requestGetEMVTagsWithTagType(
     int tagType,
     int tagOperation,
     byte [] inputTLVData,
     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 send to the device
inputDataLength	Length of the TLV data block

Return Value:
Returns a 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. For additional information, see **Example of requestGetEMVTags**.

```
long requestGetEMVTags(
     int tagType,
     int tagOperation,
     byte [] inputTLVData,
     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 send to the device
inputDataLength	Length of the TLV data block

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

3.29 requestGetEMVTags

This function sends the EMV Tag report to the device to read or write EMV Tags. For additional information, see **Example of requestGetEMVTags**.

```
long requestGetEMVTags(
     int tagType,
     int tagOperation,
     byte [] inputTLVData,
     int inputDataLength,
     int database,
     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: 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	0 = Contact L2 EMV Tags 1 = PayPass - MasterCard 2 = PayWave - VISA 3 = ExpressPay - AMEX 4 = Discover
reserved	Null – reserved for future use

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

3.30 requestSetEMVTagsWithTagType [Deprecated: V5.01]

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

```
long requestSetEMVTagsWithTagType(
     int tagType,
     int tagOperation,
     byte [] inputTLVData,
     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

Parameter	Description
tagOperation	Type of operation to be performed: 0x01 = 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 a 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. For additional information, see **Example of requestGetEMVTags**.

```
long requestSetEMVTags(
     int tagType,
     int tagOperation,
     byte [] inputTLVData,
     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: 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 a value (0: Success, Non-Zero: Error)

3.32 requestSetEMVTags

This function sends the EMV Tag report to the device to read or write EMV Tags. For additional information, see **Example of requestGetEMVTags**.

```
long requestSetEMVTags(
     int tagType,
     int tagOperation,
     byte [] inputTLVData,
     int inputDataLength,
     int database,
```

```
     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
inputTLVData	TLV data block to send to the device
inputDataLength	Length of the TLV data block
database	0 = Contact L2 EMV Tags 1 = PayPass - MasterCard 2 = PayWave - VISA 3 = ExpressPay - AMEX 4 = Discover
reserved	Null – reserved for future use

Return Value:
Returns a 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. For additional information.

```
long requestSetEMVTagsEx(
     int tagType,
     int tagOperation,
     byte [] inputTLVData,
     int inputDataLength,
     int 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	0 = Contact L2 EMV Tags 1 = PayPass – MasterCard 2 = PayWave – VISA 3 = ExpressPay - AMEX 4 = Discover
option	Response type: 0x00 = Single ACK Response 0x01 = Delay ACK Response
reserved	Null – reserved for future use

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

3.34 setCAPublicKeyWithOperation [Deprecated: V5.01]

This function sets / deletes the corresponding CA Public Key, depending on the operation specified.

```
long setCAPublicKeyWithOperation(
     int operation,
     byte [] keyBlock,
     int keyBlockLength);
```

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	CA Public Key to send
keyBlockLength	Length of the CA Public Key

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

3.35 setCAPublicKey

This function sets / deletes the corresponding CA Public Key, depending on the operation specified. For additional information, see appendix B.

```
long setCAPublicKey(
     int operation,
```

```
     byte [] keyBlock,
     int keyBlockLength);
```

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	CA Public Key to send
keyBlockLength	Length of the CA Public Key

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

3.36 setDisplayMessage

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

```
long 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	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)

3.37 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)***.

```
long sendBigBlockData(
     int dataTypeID,
     byte [] data,
     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)

3.38 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.

```
long sendBitmap(
     int slot,
     int option,
     byte [] bitmapData
     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)

3.39 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 class IPADDevInfo
{
      public String Model;
      public String DevicePath;
      public String Serial;
      public String FWVersion;
      public int Version;
      public int PID;
      public int VID;
}
```

3.40 requestDeviceInformationWithMode [Deprecated: V5.01]

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

```
String requestDeviceInformationWithMode(
      int mode,
      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.

3.41 requestDeviceInformation

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

```
String requestDeviceInformation(
      int mode,
      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.

3.42 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**.

```
DEV_STATE_STAT requestDeviceStatusForInformation(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.

```
class DEV_STATE_STAT
{
     byte nDeviceState;
     byte nSessionState;
     byte nDeviceStatus;
     byte nDevCertStatus;
     byte nHWStatus;
     byte nICCMasterSessKeyStatus;

     long nReturnCode;
     byte nOpStatus;

     public short DevState;
```

```
     public short SessState;
     public short DevStatus;
     public short DevCertStatus;
     public short HWStatus;
     public short ICCMasterSessKeyStatus;
}
```

3.43 requestKernelInformation

This function retrieves the device's kernel information.

```
long 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)

3.44 getBINTableData

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

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

Return Value: A byte array of device BIN data.

3.45 setBINTableData

This function retrieves the BIN table data. See report `0x32`. For additional information, see appendix B.

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

Parameter	Description
binTable	Buffer pointer to the BIN table byte array
reserved	RFU

Parameter	Description
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)

3.46 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.

3.47 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**.

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

Figure 3-3 - DynaPro Swipe Prompts

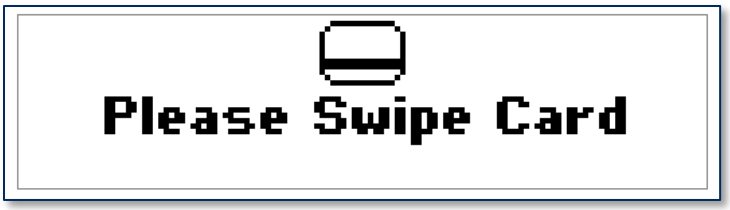


Figure 3-4 - DynaPro Mini Initial Swipe Prompt

Parameter	Description
waitTime	Time the device will wait for the user to complete a card swipe

Parameter	Description
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)

3.48 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**.

```
long requestManualCardDataWithWaitTime(
     int waitTime,
     int beepTones,
     int options,
     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

Parameter	Description
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.
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)

3.49 requestManualCardData

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

```
long requestManualCardData(
     int waitTime,
     int beepTones,
     int options,
     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

Parameter	Description
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.
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)

3.50 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**.

```
long requestUserDataEntryWithWaitTime(
     int waitTime,
     int displayMessageID,
     int beepTones,
     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)

Parameter	Description
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)

3.51 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**.

```
long requestUserDataEntry(
     int waitTime,
     int displayMessageID,
     int beepTones,
     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)

3.52 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**.

```
long 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
beepTones	Tone to use: 0 = None, 1 = Single Beep 2 = Double Beep

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

3.53 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**.

```
long 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)

3.54 selectCreditDebit

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

```
long 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)

3.55 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 software can call the **getPINData** function to retrieve data. The event associated with this function is **onPINRequestComplete**.

```
long requestPIN(
     int waitTime,
     int pinMode,
     int minPINLength,
     int maxPINLength,
     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
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
lpFieldSep	Delimiter to separate the data

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

3.56 requestPINExt

This function wraps device command `0x40`. It directs the device to prompt the user to enter a PIN by displaying one of three predetermined messages and playing a specified sound. The messages on the device's screen look like the figures below. The software can call the **getPINData** function to retrieve data. The event associated with this function is **onPINRequestComplete**.

```
long requestPINExt(
     int waitTime,
     int pinMode,
     int minPINLength,
     int maxPINLength,
     int beepTones,
     int option,
```

```
    String PAN,
    String lpFieldSep);
```

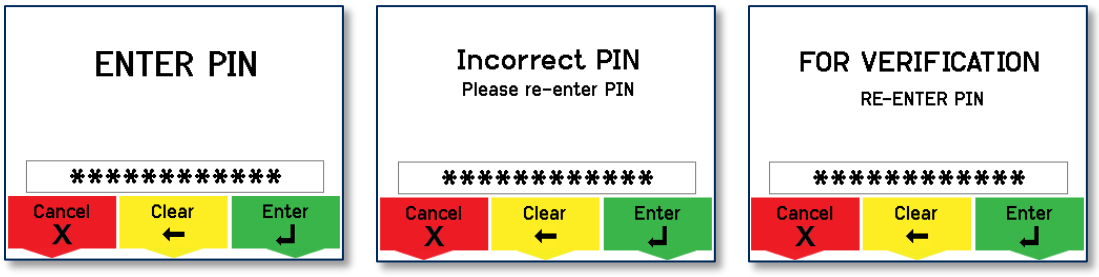


Figure 3-7 – DynaPro Go PIN Prompt

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 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. Bit 0 0 = ISO0 Format 1 = ISO3 Format Bit 1 0 = No Verify PIN 1 = Verify PIN Bit 2 0 = PAN not included (Encrypted PIN Block defaults to ISO1 format) 1 = PAN included
PAN	PAN (Primary Account Number) in ASCII format 12 characters. Example: “923456789012”
lpFieldSep	Delimiter to separate the data

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

3.57 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**.

```
long 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)

3.58 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 event associated with this command is **onEMVDataComplete**.

```
long requestSmartCardWithCardType (
     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
confirmationTime	Time the device will wait for the user to begin the transaction

Parameter	Description
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	45-byte array reserved for future use

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

3.59 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 event associated with this command is **onEMVDataComplete**.

```
long 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 + 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 = Contactless smart card + contact smart card + magnetic stripe
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	45-byte array reserved for future use

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

3.60 sendAcquirerResponseWithResponse [Deprecated: V5.01]

This function sends the Acquirer report to the device.

```
long sendAcquirerResponseWithResponse(
     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)

3.61 sendAcquirerResponse

This function sends the Acquirer report to the device.

```
long 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)

3.62 getCardDataInfo

This function returns card data.

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

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

```
public class CARD_DATA_INFO
{
     public byte DataType;
     public byte OperationStatus;
     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 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 TLVStringList;
}
```

3.63 requestDeviceConfigurationForInformation

This function retrieves the device configuration.

```
short[] requestDeviceConfigurationForInformation( 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 the current device configuration, which is an array of short.

3.64 getProductID

This function returns the device's product identifier.

```
String getProductID();
```

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

3.65 getDeviceSerial

This function returns the device's serial number.

```
String getDeviceSerial();
```

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

3.66 getDeviceModel

This function returns the device's model number.

```
String getDeviceModel();
```

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

3.67 getDeviceFirmwareVersion

This function returns the device's firmware revision number.

```
String getDeviceFirmwareVersion();
```

Return Value:
Returns a null terminated string.

3.68 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.

3.69 getPINKSN

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

```
String getPINKSN();
```

Return Value: String

3.70 getDeviceConnectedState

This function returns the device connection status.

```
int getDeviceConnectedState();
```

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

3.71 getSessionState

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

```
long 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)***.

3.72 getPAN

This function retrieves device PAN value. Requires that the software first call **requestCard** for valid data.

```
String getPAN();
```

Return Value: PAN

3.73 getEncodeType

This function returns a string value for a card type.

```
String getEncodeType();
```

Return Value:
0 = Other
1 = Financial
2 = AAMVA
3 = Manual
4 = Unknown
5 = ICC
6 = Contactless IC
7 = Financial ICC

3.74 getTrack1

This function returns the encrypted track 1 data of the swiped card, or an empty string.

```
String getTrack1();
```

Return Value: String

3.75 getTrack2

This function returns the encrypted track 2 data of the swiped card, or an empty string.

```
String getTrack2();
```

Return Value: String

3.76 getTrack3

This function returns the encrypted track 3 data of the swiped card, or an empty string.

```
String getTrack3();
```

Return Value: String

3.77 getTrack1Masked

This function return the masked track 1 data of the swiped card, or an empty string.

```
String getTrack1Masked();
```

Return Value: String

3.78 getTrack2Masked

This function return the masked track 2 data of the swiped card, or an empty string.

```
String getTrack2Masked();
```

Return Value: String

3.79 getTrack3Masked

This function returns the masked track 3 data of the swiped card, or an empty string.

```
String getTrack3Masked();
```

Return Value: String

3.80 getMaskedTracks

This function returns the masked tracks of the swiped card, or an empty string.

```
String getMaskedTracks();
```

Return Value: String

3.81 getMagnePrint

This function returns the hexadecimal MagnePrint string of the swiped card, or an empty string.

```
String getMagnePrint();
```

Return Value: String

3.82 getMagnePrintStatus

This function returns the MagnePrint status of the swiped card, or an empty string.

```
String getMagnePrintStatus();
```

Return Value: String containing MagnePrint status.

3.83 getTrack1DecodeStatus

This function returns the track 1 decode status of the swiped card, or an empty string.

```
String getTrack1DecodeStatus();
```

Return Value: String
0 = no error
1 = error detected

3.84 getTrack2DecodeStatus

This function returns the track 2 decode status of the swiped card, or an empty string.

```
String getTrack2DecodeStatus();
```

Return Value: String
0 = no error
1 = error detected

3.85 getTrack3DecodeStatus

This function returns the track 3 decode status of the swiped card, or an empty string.

```
String getTrack3DecodeStatus ( );
```

Return Value: String
0 = no error
1 = error detected

3.86 getLastName

This function returns the cardholder's last name from the swiped card, or an empty string.

```
String getLastName();
```

Return Value: String

3.87 getFirstName

This function returns the cardholder's first name from the swiped card, or an empty string.

```
String getFirstName();
```

Return Value: String

3.88 getMiddleName

This function returns the cardholder's middle name from the swiped card, or an empty string.

```
String getMiddleName();
```

Return Value: String

3.89 getExpDate

This function returns the expiration date of the swiped card, or an empty string.

```
String getExpDate();
```

Return Value: String

3.90 getPINStatusCode

This function returns the status of the PIN request, or an empty string.

```
String getPINStatusCode();
```

Return Value: String
0 = no error
1 = error detected

3.91 getPINData

This function returns PIN data. It requires that the software first call **requestPIN**. If there no PIN data is available, the function will return null.

```
MagTekPPUSCRAEvent.PIN_DATA getPINData();
```

Return Value:
Null if no PIN data is available, or `MagTekPPUSCRAEvent.PIN_DATA`, with structure as follows:

```
public class PIN_DATA
{
     public int opStatus;
     public String KSN;
     public String EPB;
}
```

3.92 setParameters

This function sets the file path where the Java Native Interface (JNI) will look for the PCI PED Windows SDK native DLLs. See section **How to Set Up the MagTek PCI PED Java Demo** for information about installation of the native DLLs.

```
void setParameters(String lpParameter);
```

Parameter	Description
lpParameter	String containing the JNI path. Use **magtek.ppmsr.CustomJNIPath=<path>**, for example, `"magtek.ppmsr.CustomJNIPath=C:\Windows\SysWOW64"`

3.93 getParameters

This function returns the parameters that were set using the **setParameters**.

```
String getParameters();
```

Return Value:
Returns a String parameter value or null if value not set

3.94 getEPB

This function retrieves the device's EPB value. It requires that the software first call **requestPIN** for valid EPB data.

```
String getEPB();
```

Return Value: 16-digit hexadecimal encrypted PIN block, or an empty String.

3.95 clearBuffer

This function clears the library's local cache of all data.

```
void clearBuffer();
```

3.96 requestClearTextUserDataEntryWithWaitTime [Deprecated: V5.01]

This function sends the Clear Text 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 preset messages. Similar to **requestUserDataEntry**, but data will displayed and returned in clear text. The event associated with this command is **onClearTextUserDataEntry**,

```
long requestClearTextUserDataEntryWithWaitTime(
     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)

3.97 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**,

```
long 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)

3.98 getClearTextUserDataEntry

This function returns clear text data entered by the user from the device that supports this feature. It requires that the software first call **requestClearTextUserDataEntry**.

```
MagTekPPUSCRAEvent.CLEAR_TEXT_USER_ENTRY_DATA
getClearTextUserDataEntry(boolean clearBuffer);
```

Return Value: `MagTekPPUSCRAEvent.CLEAR_TEXT_USER_ENTRY_DATA`, with structure as follows:

```
public class CLEAR_TEXT_USER_ENTRY_DATA
{
     public int opStatus;
     public int userDataMode;
     public int dataLen;
     public byte [] data;
     void clear()
}
```

3.99 getEMVTagsData

This function returns smart card tag data similar to the data from the **onEMVTagsComplete** event. It requires that the software first call **requestGetEMVTags.** For additional information, see appendix B.

```
byte[] getEMVTagsData(boolean clearBuffer);
```

Parameter	Description
clearBuffer	Option to direct the SDK to clear the returned data.

Return Value: byte array of data

3.100 getCAPublicKeyData

This function returns smart card tag data similar to the data from the **onGetCAPublicKey** event. It requires that the software first call **setCAPublicKey.**

```
byte[] getCAPublicKeyData(boolean clearBuffer);
```

Parameter	Description
clearBuffer	Option to direct the SDK to clear the returned data.

Return Value: byte array of data

3.101 getEMVCompleteData

This function returns smart card tag data similar to the data from the **onEMVDataComplete** event. It requires that the software first call **requestSmartCard**.

```
byte[] getEMVCompleteData(boolean clearBuffer);
```

Parameter	Description
clearBuffer	Option to direct the SDK to clear the returned data.

Return Value: byte array of data

3.102 getAPDUData

This function returns smart card tag data similar to the data from the **onAPDUArrived** event. It requires that the software first call **requestICCAPDUForInformation (EMV L1 only).** For additional information, see appendix B.

```
byte[] requestICCAPDUForInformation(boolean clearBuffer);
```

Parameter	Description
clearBuffer	Option to direct the SDK to clear the returned data.

Return Value: byte array of data

3.103 getPowerUpICC

This function returns smart card tag data similar to the data from the **onPowerUpICC** event. It requires that the software first call **requestPowerUpResetICC (EMV L1 only)**.

```
byte[] getPowerUpICC(boolean clearBuffer);
```

Parameter	Description
clearBuffer	Option to direct the SDK to clear the returned data.

Return Value: byte array of data

3.104 getEMVTransactionComplete

This function returns smart card tag data similar to the data from the **onEMVTransactionComplete** event. It requires that the software first call **sendAcquirerResponse**.

```
byte[] getEMVTransactionComplete(boolean clearBuffer);
```

Parameter	Description
clearBuffer	Option to direct the SDK to clear the returned data.

Return Value: byte array of data

3.105 getUserEntryData

This function returns smart card tag data similar to the data from the **onUserDataEntry** event. It requires that the software first call **requestUserDataEntry**.

```
byte[] getUserEntryData(boolean clearBuffer);
```

Parameter	Description
clearBuffer	Option to direct the SDK to clear the returned data.

Return Value: byte array of data

3.106 getSignatureData

This function returns smart card tag data similar to the data from the **onSignatureArrived** event. It requires that the software first call **requestSignature**.

```
byte[] getSignatureData(boolean clearBuffer);
```

Parameter	Description
clearBuffer	Option to direct the SDK to clear the returned data.

Return Value: byte array of data

3.107 getFnKeyPressed

This function returns smart card tag data similar to the data from the **onKeyInput** event. It requires that the software first call **requestResponse**.

```
long getFnKeyPressed();
```

Return Value: Number corresponding to the key pressed (9 if an error occurred, or the number of the key the user pressed. See ***99875585 DynaPro Programmer's Reference (Commands)*** and/or ***99875629 DynaPro Mini Programmer's Reference (Commands)*** for details).

3.108 isDeviceSRED

This function returns true for SRED device and return false if not SRED device .

```
boolean isDeviceSRED();
```

3.109 AMKInfo getAMKInfo

This function returns Master Key Info.

```
AMKInfo getAMKInfo();
```

```
public class AMKInfo
{
     public int Status = 0;
     public String  KCV;
     public String  KeyLabel;
}
```

3.110 getKeyInfo

This function returns KeyData.

- ***99875585 DynaPro PIN Encryption device Programmer's Reference Section 3.6.16***

```
String getKeyInfo(int keyInfoId);
```

3.111 getNextMSRKSN

This function retrieves the device KSN value from device

- ***99875585 DynaPro PIN Encryption device Programmer's Reference Section 3.6.28***

```
String getNextMSRKSN();
```

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

3.112 setMTPPSCRALibrary

This function provides the SDK with a reference to the object instance that implements the events described in section 4 **MTPPSCRA Events**.

```
void setMTPPSCRALibrary(MagTekPPUSCRAEvent event);
```

Parameter	Description
event	An instance that implements the callback events.

3.113 updateFirmware

This function update the device firmware.

```
long updateFirmware(
     byte[] data,
     int[] opStatus);
```

Parameter	Description
data	Byte array contain device firmware data.
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)

3.114 loadClientCertificate

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

```
long loadClientCertificate(
     string format,
     byte[] data,
     string password);
```

Parameter	Description
format	Format of the certificate file. Use: "PKCS12"
data	The data for the certificate or certificate chain. Data format is PKCS12.
password	The password for the certificate data.

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

3.115 requestTipOrCashback

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

```
long requestTipOrCashback(
     byte waitTime,
     byte mode,
     byte tone,
     byte[] amount,
     byte[] taxAmount,
     byte[] taxRate,
     byte tipMode,
     byte option1,
     byte option2,
     byte option3,
     byte[] reserved);
```

Parameter	Description
waitTime	Wait time in seconds for user to enter the information.

mode	Values: 0 = Tip Mode 1 = Cashback Mode
tone	Tone to use: 0 = None 1 = Single Beep 2 = Double Beep
amount	Byte array value of the transaction amount (format n12, 6 bytes).
tax	Byte array value of the calculated tax amount (format n12, 6 bytes).
taxRate	Byte array value of the tax rate percentage (format n6, 3 bytes).
tipMode	Values: 0 = Percent Mode 1 = 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 a value (0: Success, Non-Zero: Error)

3.116 getSelectedMenuItem

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

```
long getSelectedMenuItem(
     byte waitTime,
     byte mode,
     byte tone,
     byte[] data);
```

Parameter	Description
waitTime	Wait time in seconds for user to enter the information.
mode	Values: 0=Selection Table 1=Selection Bill
tone	Tone to use: 0 = None 1 = Single Beep 2 = Double Beep

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\rTable2\rTable3\r\r”

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

4 MTPPSCRA Events

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

If you are using the Java applet, after calling the functions in section 3 **MTPPSCRA Functions**, the PCI PED Java applet will invoke the callback functions in this chapter to provide the requested data and/or a detailed response. Custom code that uses the PCI PED Java applet should implement the following JavaScript functions to process the returning data. For sample code that demonstrates how to use these functions, see the `magtek-ppscra.html` and `magtek-ppscra.js` sample code in the SDK files.

4.1 onError

```
public void onError(int errorCode);
```

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

4.2 onDataReady

Return event for **getSpecialCommand**.

```
public void onDataReady(byte[] lpData);
```

Parameter	Description
lpData	A response string for **sendSpecialCommandWithCommand** function.

4.3 onPowerUpICC

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

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

Parameter	Description
status	Status code
emvData	EMV response byte array

4.4 onAPDUArrived

Return event for **requestICCAPDUForInformation (EMV L1 only)**. For additional information, see appendix B.

```
public void onAPDUArrived(
```

```
      byte status,
      byte[] RAPDU);
```

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

4.5 onGetCAPublicKey

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

Parameter	Description
status	Status code
key	CA public key byte array

4.6 onEMVTagsComplete

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

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

4.7 onPINRequestComplete

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 structure, or the other form, which receives the return data as a string.

```
public void onPINRequestComplete(MagTekPPUSCRAEvent.PIN_DATA lpData);
```

Parameter	Description
lpData	A response structure for **requestPIN** function. See below.

```
class PIN_DATA
{
      public byte opStatus;
      public String KSN;
      public String EPB;
}
```

4.8 onPINRequestComplete

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 void onPINRequestComplete (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).

4.9 onKeyInput

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

```
public void onKeyInput(
     byte status,
     byte key);
```

Parameter	Description
status	Status code.
key	Key pressed value.

4.10 onDisplayRequestComplete

Return event for **setDisplayMessage**.

```
public void onDisplayRequestComplete (int lpData);
```

Parameter	Description
lpData	Zero is returned.

4.11 onSignatureArrived

Response event for **requestSignature**.

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

Parameter	Description
status	Status code.
signature	Signature byte array.

4.12 onCardRequestComplete

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

```
public void onCardRequestComplete(String lpData);
```

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

4.13 onUserDataEntry

Return event for **requestUserDataEntry**.

```
public void onUserDataEntry(USER_ENTRY_DATA lpData);
```

Parameter	Description
lpData	See USER_ENTRY_DATA below.

```
class USER_ENTRY_DATA
{
      public byte opStatus;
      public String MSRKSN;
      public String EDB;

      public void clearUserEntry();
}
```

4.14 onDeviceStateUpdated

Response to **requestDeviceStatusForInformation**.

```
public void onDeviceStateUpdated(DEV_STATE_STAT deviceStateInfo);
```

Parameter	Description
deviceStateInfo	See the DEV_STATE_STAT class definition in section **requestDeviceStatusForInformation** for details.

4.15 onDeviceConnectionStateChanged

Response event for **openDevice**.

```
public void onDeviceConnectionStateChanged(int lpDevState);
```

Parameter	Description
lpDevState	A integer value: 0 = Device is disconnected 1 = Device is connected

4.16 onEMVDataComplete

Response event for **requestSmartCard**. For additional information, see appendix B.

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

Parameter	Description
Status	Status code
emvData	EMV response byte array

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**.

```
public 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

4.18 onEMVTransactionComplete

Return event for EMV data. For additional information, see appendix B.

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

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

4.19 onClearTextUserDataEntry

Return event for **requestClearTextUserDataEntry**.

```
public void onClearTextUserDataEntry(CLEAR_TEXT_USER_ENTRY_DATA
lpData);
```

Parameter	Description
lpData	See class definition below.

```
class CLEAR_TEXT_USER_ENTRY_DATA
{
     public int opStatus;
     public int userDataMode;
     public int dataLen;
     public byte [] data;

     public void clear();
}
```

4.20 onProgressUpdate

Return event for **sendBitmap**.

```
public void onProgressUpdate(
     int status,
     int updateItem,
     double updateProgress);
```

Parameter	Description
status	Byte value indicates device operation status. Zero value means OK. For more values, see Appendix A.2
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

4.21 onPayPassKernelMessage

Return event for **requestSmartCard** with contactless card type.

```
public void onPayPassKernelMessage(byte[] data);
```

Parameter	Description
data	EMV response byte array.

4.22 onDeviceMessageTipOrCashback

Return event for **requestTipOrCashback**.

```
public void onDeviceMessageTipOrCashback(
     int opStatus,
     int mode,
     byte[] amount,
     byte[] tax,
     byte[] taxRate,
     byte[] tipOrCashback,
     int reserved);
```

Parameter	Description
opStatus	Value indicates device operation status. Zero value means OK.
mode	0=Tip 1=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

4.23 onDeviceMessageSelectedMenuItem

Return event for **getSelectedMenuItem**.

```
public void onDeviceMessageSelectedMenuItem(
     int opStatus,
     int mode,
     int index,
     int reserved);
```

Parameter	Description
opStatus	Value indicates device operation status. Zero value means OK.
mode	0=Selection Table 1=Selection Bill
index	Index for Selected Menu Item. First item selected is index 0.
reserved	reserved

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

This Section contain reference to document with EMV related function use with L1 or L2 device

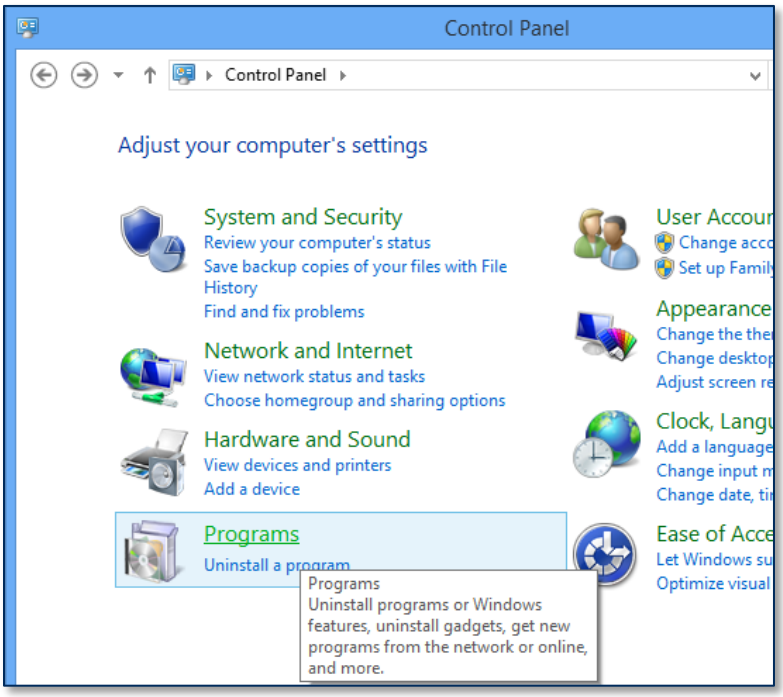
- ***99875585 DynaPro PIN Encryption device Programmer's Reference Section 3.5***

Appendix C Applet Troubleshooting

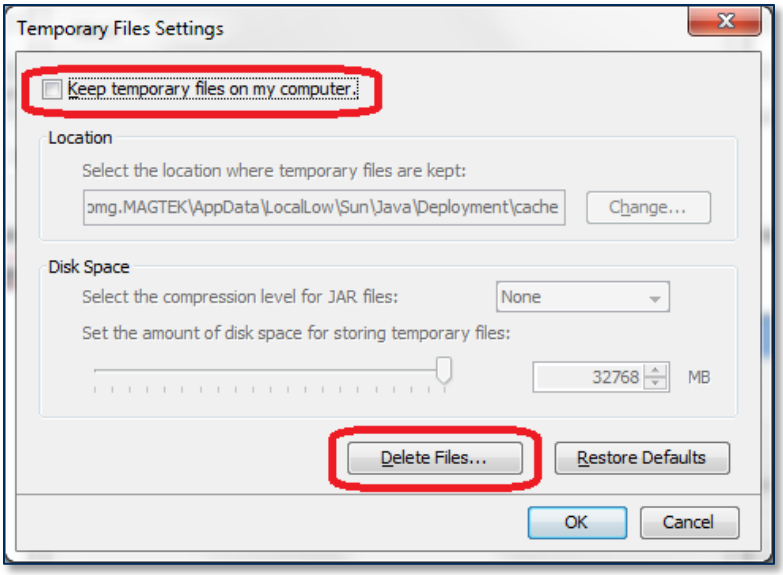
C.1 How to Clean Out Previous Applet Versions

If the Java applet is not launching from the web site correctly (such as "Unable to launch application" error messages or silent failures during applet load), follow these steps to completely remove any previously installed versions:

1) If you are using Windows 8, switch to Desktop mode.
2) Open the Windows **Control Panel**.
3) If the Control Panel is in **View by: Category** mode, select **Programs**.



4) Click the **Java (32-bit)** link (Windows 8) or the **Java** link (Windows 7) to open the **Java Control Panel** window.
5) Select the **General** tab.
6) Under the **Temporary Internet Files** heading, press the **Settings...** button to launch the **Temporary Files Settings** window.
7) Turn off the checkbox for **Keep temporary files on my computer**.
8) Press the **Delete Files...** button to launch the **Delete Files and Applications** window.



9) In the **Delete Files and Applications** window, turn on **all the checkboxes**, then press the **OK** button to clear all downloaded Java-based software and log files.
10) Press the **OK** button to close the **Temporary Files Settings** window.
11) Press the **OK** button to close the **Java Control Panel** window.
12) Launch Windows Explorer. If you are using a 32-bit version of Windows, navigate to **C:\Windows\System32**. If you are using a 64-bit version of Windows, navigate to **C:\Windows\SysWOW64.**
13) In that folder, search for the following files and delete any that exist:
    a) MTIPADLIB.dll
    b) JMTIPADLIB.dll
    c) MTPPSCRA.dll
    d) MTPPSCRAJ.dll
14) Re-install the applet by following the steps in section 2 **How to Set Up the MagTek PCI PED Java Demo**
15) How to Download and Set Up the MagTek PCI PED Java Demo
To set up the MTPPSCRANET Libraries, follow these ***steps:***
Download the DynaPro/DynaPro Go/DynaPro Mini Windows SDK Install, available from MagTek.com (Support > **Software** > Programming Tools > **IPAD, DynaPro, and DynaPro Mini > IPAD/DynaPro/DynaPro Go/DynaPro Mini Windows API)**
Right-click 99510127.exe **and** select **Run as administrator**. The installer will place all dependencies in appropriate paths.
16) **How** to Set Up **the Java Library**.

C.2 Examining Java Console Outputs for the Applet

Troubleshooting the applet can sometimes involve examining Java console outputs to see where the software load / initialize / run process went wrong. For comparison purposes, this appendix contains a sample Java console output which shows a successful load and initialization of the PCI PED Java applet.

```
Java Plug-in 10.51.2.13
Using JRE version 1.7.0_51-b13 Java HotSpot(TM) Client VM
User home directory = C:\Users\username
----------------------------------------------------
c:   clear console window
f:   finalize objects on finalization queue
g:   garbage collect
h:   display this help message
l:   dump classloader list
m:   print memory usage
o:   trigger logging
q:   hide console
r:   reload policy configuration
s:   dump system and deployment properties
t:   dump thread list
v:   dump thread stack
x:   clear classloader cache
0-5: set trace level to <n>
----------------------------------------------------
cache: Initialize resource manager:
com.sun.deploy.cache.ResourceProviderImpl@1f49969
basic: Added progress listener:
sun.plugin.util.ProgressMonitorAdapter@102c002
basic: Plugin2ClassLoader.addURL parent called for
http://localhost/dynapro/magtek-ppscra-applet.jar
basic: Plugin2ClassLoader.addURL parent called for
http://localhost/dynapro/magtek-ppscra-lib.jar
network: Connecting http://localhost/dynapro/magtek-ppscra-applet.jar
with proxy=DIRECT
network: Connecting http://localhost:80/ with proxy=DIRECT
network: Connecting http://localhost/dynapro/magtek-ppscra-applet.jar
with proxy=DIRECT
network: Connecting http://localhost:80/ with proxy=DIRECT
network: ResponseCode for http://localhost/dynapro/magtek-ppscra-
applet.jar : 200
network: Encoding for http://localhost/dynapro/magtek-ppscra-
applet.jar : null
network: Server response: (length: 48703, lastModified: Tue Mar 04
09:07:56 PST 2014, downloadVersion: null, mimeType: application/java-
archive)
network: Downloading resource: http://localhost/dynapro/magtek-ppscra-
applet.jar
     Content-Length: 48,703
     Content-Encoding: null
```

```
network: Wrote URL http://localhost/dynapro/magtek-ppscra-applet.jar
to File
C:\Users\username\AppData\Local\Temp\jar_cache1268027845282507103.tmp
security: Blacklist revocation check is enabled
security: blacklist: created: NEED_LOAD, lastModified: xxxxxxxxxxxxx
security: blacklist: check contains
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx, state now NEED_LOAD
security: blacklist: loadCache
security: blacklist: not  found in cache
security: Trusted libraries list check is enabled
security: Trusted libraries list file not found
security: blacklist: check contains
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx, state now IN_MEMORY
security: blacklist: not  found in cache
network: Disconnect connection to http://localhost/dynapro/magtek-
ppscra-applet.jar
network: Downloaded http://localhost/dynapro/magtek-ppscra-applet.jar:
C:\Users\username\AppData\Local\Temp\jar_cache1268027845282507103.tmp
cache: Adding MemoryCache entry: http://localhost/dynapro/magtek-
ppscra-applet.jar
security: Loading Deployment certificates from
C:\Users\username\AppData\LocalLow\Sun\Java\Deployment\security\truste
d.certs
security: Loaded Deployment certificates from
C:\Users\username\AppData\LocalLow\Sun\Java\Deployment\security\truste
d.certs
security: Loading certificates from Deployment session certificate
store
security: Loaded certificates from Deployment session certificate
store
security: Loading certificates from Deployment session certificate
store
security: Loaded certificates from Deployment session certificate
store
security: Loading certificates from Deployment session certificate
store
security: Loaded certificates from Deployment session certificate
store
security: Loading certificates from Internet Explorer TrustedPublisher
certificate store
security: Loaded certificates from Internet Explorer TrustedPublisher
certificate store
security: Loading certificates from Internet Explorer DISALLOWED
certificate store
security: Loaded certificates from Internet Explorer DISALLOWED
certificate store
security: Validate the certificate chain using CertPath API
security: Loading certificates from Internet Explorer ROOT certificate
store
security: Loaded certificates from Internet Explorer ROOT certificate
store
```

```
security: Loading blacklisted.certs file:
C:\Users\username\AppData\LocalLow\Sun\Java\Deployment\security\blackl
isted.certs
security: SHA-256Certificate finger print:
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
security: Checking if certificate is in Internet Explorer DISALLOWED
certificate store
security: SHA-256Certificate finger print:
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
security: Checking if certificate is in Internet Explorer DISALLOWED
certificate store
security: SHA-256Certificate finger print:
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
security: Checking if certificate is in Internet Explorer DISALLOWED
certificate store
security: SHA-256Certificate finger print:
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
security: Checking if certificate is in Internet Explorer DISALLOWED
certificate store
security: The OCSP support is enabled
security: The CRL support is enabled
network: Connecting http://ocsp.verisign.com/ with proxy=DIRECT
network: Connecting http://ocsp.verisign.com:80/ with proxy=DIRECT
security: OCSP Response: GOOD
network: Connecting http://ocsp.verisign.com/ with proxy=DIRECT
security: OCSP Response: GOOD
network: Connecting http://ocsp.verisign.com/ with proxy=DIRECT
security: OCSP Response: GOOD
security: Certificate validation succeeded using OCSP/CRL
security: Checking if certificate is in Internet Explorer
TrustedPublisher certificate store
basic: Dialog type is not candidate for embedding
security: User has granted the privileges to the code for this session
only
security: Saving certificates in Deployment session certificate store
security: Saved certificates in Deployment session certificate store
security: Grant socket perm for http://localhost/dynapro/magtek-
ppscra-applet.jar : java.security.Permissions@xxxxx (
 ("java.net.SocketPermission" "localhost" "connect,accept,resolve")
)

security: Validate the certificate chain using CertPath API
basic: Plugin2ClassLoader.getPermissions CeilingPolicy allPerms
network: Connecting http://localhost/dynapro/magtek-ppscra-lib.jar
with proxy=DIRECT
network: Connecting http://localhost:80/ with proxy=DIRECT
network: Connecting http://localhost/dynapro/magtek-ppscra-lib.jar
with proxy=DIRECT
network: Connecting http://localhost:80/ with proxy=DIRECT
network: ResponseCode for http://localhost/dynapro/magtek-ppscra-
lib.jar : 200
```

```
network: Encoding for http://localhost/dynapro/magtek-ppscra-lib.jar :
null
network: Server response: (length: 75591, lastModified: Tue Mar 04
08:04:24 PST 2014, downloadVersion: null, mimeType: application/java-
archive)
network: Downloading resource: http://localhost/dynapro/magtek-ppscra-
lib.jar
     Content-Length: 75,591
     Content-Encoding: null
network: Wrote URL http://localhost/dynapro/magtek-ppscra-lib.jar to
File
C:\Users\username\AppData\Local\Temp\jar_cachexxxxxxxxxxxxxxxxxxx.tmp
security: blacklist: check contains
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx=, state now IN_MEMORY
security: blacklist: not  found in cache
security: Trusted libraries list file not found
security: blacklist: check contains
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx, state now IN_MEMORY
security: blacklist: not  found in cache
network: Disconnect connection to http://localhost/dynapro/magtek-
ppscra-lib.jar
network: Downloaded http://localhost/dynapro/magtek-ppscra-lib.jar:
C:\Users\username\AppData\Local\Temp\jar_cachexxxxxxxxxxxxxxxxxxx.tmp
cache: Adding MemoryCache entry: http://localhost/dynapro/magtek-
ppscra-lib.jar
security: Missing Codebase manifest attribute for:
http://localhost/dynapro/magtek-ppscra-lib.jar
security: Missing Application-Library-Allowable-Codebase manifest
attribute for: http://localhost/dynapro/magtek-ppscra-lib.jar
security: Validate the certificate chain using CertPath API
security: Grant socket perm for http://localhost/dynapro/magtek-
ppscra-lib.jar : java.security.Permissions@9a9631 (
 ("java.net.SocketPermission" "localhost" "connect,accept,resolve")
)

security: Missing Codebase manifest attribute for:
http://localhost/dynapro/magtek-ppscra-lib.jar
security: Missing Application-Library-Allowable-Codebase manifest
attribute for: http://localhost/dynapro/magtek-ppscra-lib.jar
security: Validate the certificate chain using CertPath API
basic: Plugin2ClassLoader.getPermissions CeilingPolicy allPerms
security: Validate the certificate chain using CertPath API
security: SSV validation:
    running: 1.7.0_51
    requested: null
    range: null
    javaVersionParam: null
    Rule Set version: null
network: Created version ID: 1.7.0.51
network: Created version ID: 1.7.0.51
security: continue with running version
```

```
network: Created version ID: 1.7.0.51
network: Created version ID: 1.7
network: Created version ID: 2.2.51
security:  --- parseCommandLine converted :
into:
[]
basic: Applet loaded.
basic: Applet resized and added to parent container
basic: PERF: AppletExecutionRunnable - applet.init() BEGIN ; jvmLaunch
dt xxxxxx us, pluginInit dt xxxxxxxxx us, TotalTime: xxxxxxxxx us
+ getParameterValue:magtek.ppmsr.CustomJNIPath
+ getParameterValue:magtek.ppmsr.Run64BitJNI
OS=Windows 7
DLLPATH=C:\Windows\sysWOW64\
DLLPATH=C:\Windows\sysWOW64\
LoadJNILib: START!
LoadJNILib: Prepare to Load DLL!
Loading file:C:\Windows\sysWOW64\MTPPSCRA.dll
Loading file:C:\Windows\sysWOW64\MTPPSCRAJ.DLL
LoadJNILib: END!
async mode enabled
basic: Applet initialized
basic: Starting applet
basic: completed perf rollup
basic: Applet made visible
basic: Applet started
basic: Told clients applet is started
```

Appendix D Cryptography

D.1 Decrypt PIN

getStatusCode, getKSN, getEPP

D.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[] = new byte[10];
byte bEPB[] = new byte[8];
```

Then, derive key from BDK and KSN

```
byte bPinKey[] = new byte[16];

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

D.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[]{0,0,0,0,0,0,0,0};

SecretKeySpec pKey = new SecretKeySpec(bPinKey, "DES");
IvParameterSpec ivectorSpecv = new IvParameterSpec(iv);
Cipher c = Cipher.getInstance("DES/CBC/NoPadding", "BC");
c.init(Cipher.DECRYPT_MODE, pKey, ivectorSpecv);
int ptLen = c.update(bEPB, 0, 8, bIsoPinBlock, 0);
ptLen += c.doFinal(bIsoPinBlock, ptLen);
```

D.1.3 Extract PIN from PIN block

Reference to ISO 9564-1.

D.2 Decrypt Card Track

D.2.1 Get key for the PIN decryption from BDK and KSN

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

```
byte bEncTrack1[] = convertHexStrToByteArray(getTrack1())
byte bEncTrack2[] = convertHexStrToByteArray(getTrack2())
byte bEncTrack3[] = convertHexStrToByteArray(getTrack3())
byte bKSN[] = convertHexStrToByteArray(getKSN())
```

Then, derive key from BDK and KSN

```
byte bDataKey[] = new byte16];

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

D.2.2 Use Triple DES CBC to decrypt track data

Decrypt track data, use empty initial vector.

```
char bDataKey[16];

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

D.2.3 Use Triple DES CBC to decrypt track data

Decrypt Encrypted PIN Block, use empty initial vector.

```
byte iv[]= new byte[]{0,0,0,0,0,0,0,0};
byte bDecTrack1[] = new byte[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.
```

D.3 Calculate CBC MAC

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

D.3.1 Get key

Derive key from BDK and KSN

```
byte bDataKey = new byte[16];
byte IV = new byte[] {0,0,0,0,0,0,0,0};

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

D.3.2 Padding data

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

```
int nPaddedLength = ((nDataLength + 7) & (~7));
byte bDataPadded[] = new byte [nPaddedLength];
System.arraycopy(data, 0, bDataPadded, 0, nDataLength);
```

D.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 left most 32 bits as MAC value.

```
byte[] pLeftKey = Arrays.copyOfRange(bDataKey, 0, 7);
byte[] pRightKey = Arrays.copyOfRange(bDataKey, 8, 15);

byte[] bDataOutput = new byte[nPaddedLength];

SecretKeySpec pKey = new SecretKeySpec(pLeftKey, "DES");
IvParameterSpec ivectorSpecv = new IvParameterSpec(IV);
Cipher c = Cipher.getInstance("DES/CBC/NoPadding", "BC");
c.init(Cipher.ENCRYPT_MODE, pKey, ivectorSpecv);
int l = c.update(bDataPadded, 0, bDataPadded.length, bDataOutput, 0);
l += c.doFinal(bDataOutput, l);
```

```
// bDataOutput contain the encrypted data
char pLastBlock[] = Arrays.copyOfRange(bDataOutput, bDataOutput+
nPaddedLength-8, bDataOutput+ nPaddedLength);
byte MAC[] = new byte[8];

//DES_Decrypt_ECB(pRightKey, IV, pLastBlock, 8, MAC, 8);
c.init(Cipher.DECRYPT_MODE, pKey, ivectorSpecv);
int ptLen = c.update(encr, 0, ctLen, MAC, 0);
ptLen += c.doFinal(decrpt, ptLen);

//DES_Encrypt_ECB(pLeftKey, IV, MAC, 8, MAC, 8);
c.init(Cipher.ENCRYPT_MODE, pKey, iv);
int ctLen = c.update(input, 0, input.length, MAC, 0);
ctLen += c.doFinal(encr, ctLen);

//Only use first 4 bytes of MAC buffer.
```

D.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.

D.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
CBC_MAC(bOutMessage, nOutMessage, bSecuredMessage,
nSecuredMessageLength);

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

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

Parse the data through TLV format. For encrypted data tag, use DES/CBC to decrypt it.

D.5 Example of requestSmartCard

D.5.1 Host: requestSmartCard

```
requestSmartCard (cardType, confirmWaitTime, pinWaitTime, tone,
option, amount, transType, cashBack, reserved);
```

D.5.2 Device: onEMVDataComplete

```
public void onEMVDataComplete(short status, byte[] emvData) {

     // ParseTLVData
```

```
    //ConstructAcqMsg
    byte[] outdatawithmac = null;
    byte[] serial = new byte[8];// copy 8 byte serial number here
    byte[] Msg = new byte[data.length + 41];
    Msg[0] = (byte)((data.length + 39 & 0x0000FF00) >> 8);
    Msg[1] = (byte)((data.length + 39 & 0x000000FF));
    Msg[2] = (byte)0xF9;
    byte len1 = (byte)(data.length + 35);
    Msg[3] = (byte)0x82;
    Msg[4] = (byte)((len1 >> 8) & 0xFF);
    Msg[5] = (byte)(len1 & 0xFF);

    Msg[6] = (byte)0xDF;
    Msg[7] = (byte)0xDF;
    Msg[8] = 0x54;
    Msg[9] = 0xA;

    System.arraycopy(ksn, 0, Msg, 10, ksn.length);

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

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

    System.arraycopy(serial, 0, Msg, 29, serial.length);

    Msg[37] = (byte)0xFA;
    int len2 = data.length;
    Msg[38] = (byte)0x82;
    Msg[39] = (byte)((len2 >> 8) & 0xFF);
    Msg[40] = (byte)(len2 & 0xFF);

    System.arraycopy(data, 0, Msg, 41, data.length);

    byte[] keyl = new byte[8];
    byte[] keyr = new byte[8];
    String err = "";
    byte[] iv = new byte[8];
    byte[] BDK = new byte[16];//HexToByteConversion(txtBDK);

    //TODO: DeriveMAC Key Here
    //DeriveCurrentMACReqKey(ksn, BDK, keyl, keyr, ref err);

    byte[] indata = new byte[Msg.length];
    System.arraycopy(Msg, 0, indata, 0, indata.length);
```

```
     int iPaddingBytes = 0;
     if ((indata.length > 0)  && ((indata.length % 8) > 0))
     iPaddingBytes = 8 - (indata.length % 8);

     byte[] outdata = new byte[indata.length + iPaddingBytes];
     byte[] mac = new byte[8];

     //TODO: Calculate mac and data
     //CalculateCBC_MAC(keyl, keyr, iv, indata, outdata, mac);

     outdatawithmac = new byte[outdata.length + 4];
     System.arraycopy(outdata, 0, outdatawithmac, 0, outdata.length);
     System.arraycopy(mac, 0, outdatawithmac, outdata.length, 4);
}
```

D.5.3 Host: sendAcquireResponse

```
sendAcquirerResponse(outdatawithmac, outdatawithmac.length)
```

D.5.4 Device: onEMVTransactionComplete

```
public void onEMVTransactionComplete(short status, byte[] data) {

// ParseTLVData

     //Update user with new Transaction Status Approve/Decline etc.

     endSession((byte) 1);
}
```

D.6 Example of requestGetEMVTags

D.6.1 Host: requestGetEMVTags

```
requestGetEMVTagsWithTagType(tagType, tagOperation,
tlvData,tlvDataLength, database, reserved);
```

D.6.2 Device: OnEMVTagsCompleteEvent

```
public void onEMVTagsComplete(short status, byte[] tags) {

     //ParseTLVData
     //DeriveCurrentMACReqKey
     //CalculateCBC_MAC
     //Show Tags data
}
```

D.7 Example of setCAPublicKey

D.7.1 Host: setCAPublicKey

```
public enum CAPublicKeyOperation : byte
{
      EraseAll = 0,
      EraseAllForGivenRid = 1,
      EraseAllForGivenRidAndRidKeyIndex = 2,
      AddNew = 3,
      ReadSinglePKey = 4,
      ReadAll = 0x0F
}
```

```
public void setCAPublicKey(byte [] CAPublicKeyBlock) {

//Construct Mac Message:
//ReadSinglePKey, AddNew, EraseAllForGivenRidAndRidKeyIndex.
EraseAllForGivenRid
byte[] MsgToSend = null;
byte[] serial =  new byte[8];// copy 8 byte serial number here;
byte[] Msg = new byte[data.length + 33];
Msg[0] = (byte)((data.length + 31 & 0x0000FF00) >> 8);
Msg[1] = (byte)((data.length + 31 & 0x000000FF));
Msg[2] = (byte)0xF9;

int len1 = data.length + 27;
Msg[3] = (byte)0x82;
Msg[4] = (byte)((len1 >> 8) & 0xFF);
Msg[5] = (byte)(len1 & 0xFF);

Msg[6] = (byte)0xDF;
Msg[7] = (byte)0xDF;
Msg[8] = 0x55;
Msg[9] = 0x01;
Msg[10] = 0x02;

Msg[11] = (byte)0xDF;
Msg[12] = (byte)0xDF;
Msg[13] = (byte)0x25;
Msg[14] = 0x08;

System.arraycopy(serial, 0, Msg, 0, serial.length);

Msg[23] = (byte)0xFA;
int len2 = data.length + 6;
Msg[24] = (byte)0x82;
Msg[25] = (byte)((len2 >> 8) & 0xFF);
Msg[26] = (byte)(len2 & 0xFF);

Msg[27] = (byte)0xDF;
```

```
Msg[28] = (byte)0xDF;
Msg[29] = 0x3F;
int len3 = data.length;
Msg[30] = (byte)0x82;
Msg[31] = (byte)((len3 >> 8) & 0xFF);
Msg[32] = (byte)(len3 & 0xFF);

System.arraycopy(data, 0, Msg, 33, data.length);

//TODO: Encrypt the Message
//MsgToSend = CBCMAC_Data(Msg);

setCAPublicKey ((byte)operation , CAPublicKeyBlock,
CAPublicKeyBlock.length);
lpStatus[0] = (int)m_MTSCRA.getStatusCode();

}
```

D.7.2 Device: onGetCAPublicKey

```
public void onGetCAPublicKey(short status, byte[] key) {

     // ParseTLVData
     if (Operation == CAPublicKeyOperation.ReadAll)
          GetPKeys(data);
     else
          GetPKeyData(data);
}
```

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

E.1 Overview

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

E.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 = 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 Copy int number to byte array
SetInt32(transformedNumberSerial, rndNumber);

//Copy first 4 bytes SessionInfo to transformedNumberSerial start
index equal to 4
System.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);
```

E.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
```

E.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]);
}


// Implements IMTPPSCRA onAPDUArrived(short status, byte[] APDU)
public void onAPDUArrived(short status, byte[] APDU){ }

// 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 onApduArrived
```

E.5 Power Down ICC

Host uses requestPowerDownICC to power down card.

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

E.6 End L1 Session

Host uses endL1Session to close the secure communication

```
retCode = endL1Session();
```

Appendix F 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
requestICCAPDUForInformation	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

Appendix F - Function Applicable Table

Function Name	IPAD	DYNAPRO	DYNAPRO MINI	DYNAPRO GO
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
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
isDeviceConnected	YES	YES	YES	YES
getPINKSN	YES	YES	YES	YES
getDeviceConnectedState	YES	YES	YES	YES
getSessionState	YES	YES	YES	YES
getPAN	YES	YES	YES	YES

Appendix F - Function Applicable Table

Function Name	IPAD	DYNAPRO	DYNAPRO MINI	DYNAPRO GO
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
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
getClearTextUserDataEntry	NO*	NO*	NO	YES
getEMVTagsData	NO	YES	YES	YES
getCAPublicKeyData	NO	YES	YES	YES
getEMVCompleteData	NO	YES	YES	YES

Appendix F - Function Applicable Table

Function Name	IPAD	DYNAPRO	DYNAPRO MINI	DYNAPRO GO
getAPDUData	NO	YES	YES	YES
getPowerUpICC	NO	YES	YES	YES
getEMVTransactionComplete	NO	YES	YES	YES
getUserEntryData	YES	YES	YES	YES
getSignatureData	YES	YES	NO	YES
getFnKeyPressed	YES	YES	YES	YES
setMTPPSCRALibrary	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
onDeviceConnectionStateChanged	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

Appendix F - Function Applicable Table

Function Name	IPAD	DYNAPRO	DYNAPRO MINI	DYNAPRO GO
loadClientCertificate	NO	NO	NO	YES
requestTipOrCashback	NO	NO	NO	YES
getSelectedMenuItem	NO	NO	NO	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.