MAGTEK®
SECURITY FROM THE INSIDE

MagTek Universal SDK

MagTek Devices

Programmer's Manual (Microsoft .NET)

March 2026

Manual Part Number:
D998200380-508

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

Copyright © 2006 – 2026 MagTek, Inc.
Printed in the United States of America

Information in this publication is subject to change without notice and may contain technical inaccuracies or graphical discrepancies. Changes or improvements made to this product will be updated in the next publication release. No part of this document may be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose, without the express written permission of MagTek, Inc.

MagTek® is a registered trademark of MagTek, Inc.
MagnePrint® is a registered trademark of MagTek, Inc.
MagneSafe® is a registered trademark of MagTek, Inc.
Magensa™ is a trademark of MagTek, Inc.
aDynamo™, iDynamo™, and uDynamo are trademarks of MagTek, Inc.
eDynamo™, Dynamag, and DynaMAX are trademarks of MagTek, Inc.
mDynamo™, DynaWave™, and tDynamo™ are trademarks of MagTek, Inc.
DynaPro Go™, DynaPro™, and DynaPro Mini™ are trademarks of MagTek, Inc.
DynaFlex™, DynaFlex Pro™, DynaProx™, DynaFlex II PED™, and DynaFlex II Go are trademarks of MagTek, Inc.

The Bluetooth® word mark and logos are registered trademarks owned by Bluetooth SIG, Inc. and any use of such marks by MagTek is under license.

Google Play™ store, Google Wallet™ payment service, and Android™ platform are trademarks of Google LLC.

Apple Pay®, iPhone®, iPod®, iPod touch®, Mac®, and OS X® are registered trademarks of Apple Inc., registered in the U.S. and other countries. iPad™ is a trademark of Apple. Inc. App Stores℠ is a service mark of Apple Inc., registered in the U.S. and other countries. IOS is a trademark or registered trademark of Cisco in the U.S. and other countries and is used by Apple Inc. under license. MAC and macOS are trademarks of Apple Inc., registered in the U.S. and other countries.

Microsoft® and Windows® are registered trademarks of Microsoft Corporation.

EMV® is a registered trademark in the U.S. and other countries and an unregistered trademark elsewhere. The EMV trademark is owned by EMVCo, LLC. The Contactless Indicator mark, consisting of four graduating arcs, is a trademark owned by and used with permission of EMVCo, LLC.

Mifare is a registered trademark of NXP Semiconductors.

All other system names and product names are the property of their respective owners.

Table 0.1 – Revisions

Rev Number	Date	Notes
10	October 6, 2020	Initial release
11	October 22, 2020	Extended the ITransaction parameter for startTransaction().

Rev Number	Date	Notes
12	January 25, 2021	Replaced setDisplay() with displayMessage(). Added getFile(), PreventMSRSignatureForCardWithICC(), and enum TechnicalFallback to TransactionStatus. Removed showImage() from IdeviceConfiguration.
20	June 25, 2021	Corrected the description of PreventMSRSignatureForCardWithICC in the ITransaction interface. Added sample of getConfiguration and setConfiguration in section B6 IdeviceConfiguration Walk Through. Added DataEntryType, DeviceEvent, OperationStatus, and UserEvent enumerations. Added Manual Card Entry, SuppressThankYouMessage, and OverrideFinalTransactionMessage properties to ITransaction interface.
30	January 17, 2022	Extended showImage(). Extended EventType enum. Added enum DeviceFeature and FeatureStatus. Added support for Barcode display, Barcode reader, and DynaProx.
40	March 11, 2022	Added requestPIN(), requestPAN() to Idevice. Updated requestSignature() to include a timeout parameter. Updated EMV transaction flow in the appendix.
50	March 14, 2023	Added support for WebSocket, WebSocket TLS, and WebSocket TLS Trust at section 3. Added Wireless as firmware type to updateFirmware() at section 9. Added BMP image format at section 6. Added CertificateInfo class, AppleVASMode and AppleVASProtocol to ITransaction at section 10. Added VASMode and VASProtocol enumerations, Apple VAS and BarCode to PaymentMethods, BarcodeRead and VASError to TransactionStatus, and WEBSOCKET and WEBSOCKET_TRUST to ConnectionType at section 13.
500	September 26, 2023	Added a supported function table at Appendix A. Added Display Amount for Quick Chip to ITransaction class at section 10.4. Added NFC Tag support to ITransaction at section 10.4 and NFC enumerations as section 13.13.
501	November 13, 2023	Added Tip and Tax support to ITransaction at section 10. Added TransactionStartedFromDevice, TransactionStartedFromDeviceQuickChip, and TransactionCancelledFromDevice to TransactionStatus enumeration at section 13.16. Added EnhancedInputRequest to EventType enumeration at section 13.9. Added DirectoryEntry, EnhancedInputRequest, InputRequest, and NFCData to section 10. Added MifareClassic1K, MifareClassic4K, IOFailed, and AuthenticationFailed to NFCEvent enumeration at section 13.13.

Rev Number	Date	Notes
502	January 10, 2024	Added DynaFlex II Go (Bluetooth LE) at section 13.4. Added NFCAPDUResponse to EventType at section 13.9. Added MifareDESFire to NFCEvent at section 13.13. Added ConnectionStateBuilder at section 10.3 Added NFCRAPDUData class at section 10.10. Added sendDESFireNFCCommand at section 4.14.
503	February 22, 2024	Added GoogleVAS to PaymentMethod enum at section 13.15. Added Apple VAS details as Appendix D. Added Google VAS details as Appendix E.
504	July 1, 2024	Added UI page support to IDeviceControl as sections 6.31 to 6.35. Added FunctionalButtonRightOption to ITransaction at section 10.8. Added UI TouchScreen events to EventType at section. 13.9.
505	May 8, 2025	Added support for MQTT API. Added MQTT methods to CoreAPI at section 3. Added StatusCallback as sections 13 and 14. Added ErrorType enum at section 15. Added MQTT to ConnectionType at section 15.4.
506	June 23, 2025	Updated DeviceEvent enumeration with battery and temperature status at section 15.6. Implemented mTLS for setMQTTClientCertificateInfo at section 3.10.
507	March 5, 2026	Added support for PersonalInfoEntry at sections 6.23, 6.26, 10.12, 10.13, 15.3, 15.11. Added support for mDL at sections 15.16, 15.20 Extended support for Mifare at sections 4.16, 10.11, 15.15, 15.16, 15.17, 15.21. Added support for card emulation at sections 6.22, 6.24. Added support for delete file at section 9.1. Extended InfoType for WLAN and BLE at section 15.14. Added support for Apple ECP2 at sections 10.8, 15.21.
508	March 19, 2026	Added support for NFC Pass-Through mode at sections 6, 15.11, 15.16.

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

Table of Contents 7
1 Introduction 12
1.1 About the MagTek Sample Code 12
1.2 Nomenclature 12
1.3 SDK Contents 12
1.3.1 System Requirements 12
2 How to Set Up the SDK 14
3 CoreAPI 15
3.1 createCMSDevice 15
3.2 createDevice 15
3.3 createMMSDevice 17
3.4 createPPSCRA 17
3.5 createSCRA 17
3.6 getDeviceList 17
3.7 getAPIVersion 18
3.8 setMQTTBrokerInfo 18
3.9 setMQTTClientID 19
3.10 setMQTTClientCertificateInfo 19
3.11 setMQTTDeviceDiscoveryTimeout 19
3.12 startMQTTDeviceStatusMonitoring 19
3.13 setMQTTPublishTopic 20
3.14 setMQTTQos 20
3.15 setMQTTSubscribeTopic 21
3.16 stopMQTTDeviceStatusMonitoring 21
3.17 setSystemStatusCallback 21
4 IDevice 22
4.1 cancelTransaction 22
4.2 getCapabilities 22
4.3 getConnectionInfo 22
4.4 getConnectionState 22
4.5 getDeviceConfiguration 22
4.6 getDeviceControl 23
4.7 getDeviceInfo 23
4.8 Name 23
4.9 requestPAN 23
4.10 requestPIN 25
4.11 requestSignature 26
4.12 sendAuthorization 26
4.13 sendClassicNFCCommand 26

4.14 sendDESFireNFCCommand 29
4.15 sendNFCCommand 31
4.16 sendPlusNFCCommand 35
4.17 sendSelection 36
4.18 startTransaction 37
4.19 subscribeAll 37
4.20 unsubscribeAll 37
5 IDeviceCapabilities 38
5.1 AutoSignatureCapture 38
5.2 BatteryBackedClock 38
5.3 Display 38
5.4 MSRPowerSaver 38
5.5 PaymentMethods 38
5.6 PINPad 39
5.7 Signature 39
5.8 SRED 39
6 IDeviceControl 40
6.1 close 40
6.2 deviceReset 40
6.3 displayMessage 40
6.4 endSession 41
6.5 getInput 41
6.6 open 41
6.7 playSound 41
6.8 send 42
6.9 sendExtendedCommand 42
6.10 sendNFCPassThroughCommand 42
6.11 sendSync 42
6.12 setDateTime 43
6.13 setLatch 43
6.14 setLEDStatus 43
6.15 setNFCPassThroughMode 44
6.16 setNFCPollingMode 44
6.17 showBarCode 45
6.18 showImage 46
6.19 showImage 46
6.20 showUIPage 48
6.21 showUIPageWithAmountButtons 51
6.22 showUIPageWithImage 53
6.23 showUIPageWithTextButtons 54
6.24 showUIPageWithTextLines 55

6.25 startBarCodeReader 56
6.26 startCardEmulation 56
6.27 startPersonalInfoEntry 57
6.28 stopBarCodeReader 57
6.29 stopCardEmulation 58
6.30 stopPersonalInfoEntry 58
7 ConnectionInfo 59
7.1 getAddress 59
7.2 getConnectionType 59
7.3 getDeviceType 60
7.4 getCertificateInfo 60
8 DeviceInfo 61
8.1 getModel 61
8.2 getName 61
8.3 getSerial 61
9 IDeviceConfiguration 61
9.1 deleteFile 61
9.2 getChallengeToken 62
9.3 getConfigInfo 62
9.4 getDeviceInfo 62
9.5 getFile 63
9.6 getKeyInfo 63
9.7 sendFile 63
9.8 sendImage 64
9.9 sendSecureFile 64
9.10 setConfigInfo 65
9.11 setDisplayImage 65
9.12 updateFirmware 65
9.13 updateKeyInfo 66
10 Classes 67
10.1 BarCodeData 67
10.2 CertificateInfo 67
10.3 ConnectionStateBuilder 68
10.4 DirectoryEntry 69
10.5 EnhancedInputRequest 70
10.6 InputRequest 70
10.7 IData 71
10.8 ITransaction 72
10.9 NFCData 77
10.10 NFCRAPDUData 77
10.11 NFCEventBuilder 78

10.12 PersonalInfoEntry 79
10.13 PersonalInfoEntryBuilder 79
11 IEventSubscriber Delegates 81
11.1 OnEvent 81
12 IConfigurationCallback Delegates 83
12.1 OnCalculateMAC 83
12.2 OnProgress 83
12.3 OnResult 83
13 IMQTTDeviceStatusCallback Delegates 85
13.1 OnConnected 85
13.2 OnDisconnected 85
14 ISystemStatusCallback Delegates 86
14.1 OnError 86
15 Enumerations 87
15.1 BarCodeFormat 87
15.2 BarCodeType 87
15.3 CaptureType 87
15.4 ConnectionState 87
15.5 ConnectionType 88
15.6 DataEntryType 89
15.7 DeviceEvent 89
15.8 DeviceFeature 90
15.9 DeviceType 90
15.10 ErrorType 91
15.11 EventType 91
15.12 FeatureStatus 93
15.13 ImageType 93
15.14 InfoType 93
15.15 NFCDataMode 94
15.16 NFCEvent 94
15.17 NFCTransactionMode 95
15.18 OperationStatus 95
15.19 PaymentMethod 96
15.20 TransactionStatus 96
15.21 UserEvent 97
15.22 VASMode 98
15.23 VASProtocol 99
Appendix A Status Codes 100
A.1 Library Status Codes 100
A.2 Supported Function List 101
Appendix B API Walk Through 105

B.1 CoreAPI Walk Trough 105
B.2 IDevice Walk Through 106
B.2.1 Handling Events 107
B.3 IDeviceControl Walk Through 110
B.4 ConnectionInfo Walk Through 111
B.5 IDeviceCapabilities Walk Through 112
B.6 IDeviceConfiguration Walk Through 113
B.6.1 Handling Events 114
Appendix C EMV Transaction Flow 115
C.1 Flow Chart - QuickChip 115
C.2 Sample Code - QuickChip 116
C.3 Flow Chart – Signature Capture 118
C.4 Sample Code – Signature Capture 119
C.5 Flow Chart – With ARPC 122
C.6 Sample Code – With ARPC 123
C.7 MSR Fallback Flow 127
Appendix D Apple VAS 130
D.1 Merchant ID and URL Slots 130
D.2 POS Capabilities 130
D.3 Start Transaction 130
D.4 Transaction Response 130
Appendix E Google Wallet Smart Tap VAS 132
E.1 Mobile Device 132
E.2 Load Key 132
E.3 Collector ID Slots 132
E.4 POS Capabilities 132
E.5 Start Transaction 132
E.6 Transaction Response 132

1 Introduction

This document provides instructions for software developers who want to create .NET software solutions that include MagTek devices connected to a Windows based host. MagTek Universal SDK (MTUSDKNET) incorporates MagTek SCRA and MagTek PIN Pad SCRA devices into one SDK. This document is part of a larger library of documents designed to assist MagTek device implementers.

The following documents are essential:

- ***D998200383 DynaFlex Products Programmer's Manual (COMMANDS)***
- ***D998200382 DynaFlex Family of Products Installation and Operation Manual***

1.1 About the MagTek Sample Code

The sample code provides C# demonstration source code and a reusable MTUSDKNET API library that provides developers of custom software solutions with an easy-to-use interface for MagTek devices. Developers can distribute the MTUSDKNET API Library to customers or distribute internally as part of an enterprise solution.

1.2 Nomenclature

- **Device** refers to the MagTek devices that receives and responds to command set.
- **Host** refers to the piece of general-purpose electronic equipment the device is connected or paired to, which sends data to and receives data from the device. Host types include but not limited to PC and Mac computers, tablets, and smartphones. When "host" must be used differently, it is qualified as something specific, such as "USB host."
- **User** in this document generally refers to the **cardholder**.

1.3 SDK Contents

File name	Description
MTUSDKDemo.exe	Visual Studio built sample executable.
MTUSDKNET.dll	This is a Microsoft .NET DLL to implement MagTek device functions.

1.3.1 System Requirements

Operating systems:

- Windows 8.1
- Windows 10
- Windows 11

- Microsoft .NET Standard 4.6.1 and above.
- Bluetooth LE requires Windows 8.1 or higher.
- 802.11 Wireless requires Windows 7 or higher.

TLS Trust and WebSocket WSS connection:

- Root Certificate, Sub CA Certificate, and Client private key Certificate as referenced in document ***D998200550 DynaFlex II PED Using Wireless LAN Guide***. The client private key certificate (client.p12) must be accessible to the custom software that makes a secure WebSocket connection to the DynaFlex II PED.

2 How to Set Up the SDK

To set up the MT Universal Libraries, download and extract the package ***1000007351 MagTek Universal SDK for MMS Devices (Windows)*** available from MagTek.com.

To build the MTUSDKDemo software, follow these steps:

1) Launch Visual Studio and open MTUSDKDemo.**csproj**
2) In the **Solution Explorer**, select MTUSDKDemo.
3) Open Reference and add the DLLs: MTUSDK.dll
4) In the top navigation bar, select **Build** > **Build Solution.**

To Run/Debug the sample code, follow these steps:

1) In Visual Studio configuration manager, select Build Configuration equal to Debug and Build platform to either x86 or x64.
2) Select **Debug**->**Start Without Debugging** to run the sample code or select **Debug**->**Start Debugging** to run it in debug mode.

3 CoreAPI

Use the CoreAPI to create an IDevice. IDevice, which is the bases for the MagTek Universal SDK.

If accessing a device specific API outside of MagTek Universal SDK, use the various functions in this section to create an instance of that device's API. Once a device specific API is referenced, the associated library will need to be added into the application's development project.

3.1 createCMSDevice

This function creates an instance of a CMS type of device.
The API's of MagTek Universal SDK do not apply. See ***D998200160 MagTek Common Message Structure (MTCMS)*** for the MTCMS API.

```
MTDevice CoreAPI.createCMSDevice();
```

Return Value:
Returns an `MTDevice`.

3.2 createDevice

This function creates an instance of IDevice. All API's of MagTek Universal SDK can but utilized from.

```
IDevice CoreAPI.createDevice(
      DeviceType deviceType,
      ConnectionType connectionType,
      string address,
      string model,
      string name,
      string serial,
      CertificateInfo certificateInfo = null);
```

Parameter	Description
deviceType	Enumerated device type.
connectionType	Enumerated connection type.

Parameter	Description

address	Address for the device.
	
	For USB devices, address may be an empty string when only one device is attached. Otherwise, address should be in the form:
	`USB://DEVICESERIALNUMBER`
	for example, `USB://99261829170E0810`
	
	For Ethernet devices, address should be in the form:
	`IP://IP-Address:PORT`
	for example, `IP://10.57.10.180:26`
	
	For Wireless DynaPro Go devices, address should be in the form:
	`TLS12://TLSDEVICESERIALNUMBER`
	`TLS12TRUST://TLSDEVICESERIALNUMBER`
	for example, `TLS12://TLS99261829170E0810`
	`TLS12TRUST://TLS99261829170E0810`
	
	For Bluetooth LE devices, address should be in the form:
	`BLEEMV://DEVICENAME`
	for example, `BLEEMV://DynaPro Go-EB66`
	
	For WebSocket or Secure WebSocket devices, address should be in the form:
	`ws://IP-Address or DEVICEADDRESS`
	`wss://IP-Address or DEVICEADDRESS`
	for example, `ws://10.57.10.1 or ws://b512345.magtek.com`
	`wss://192.168.1.150 or wss://b512345.magtek.com`
	
	The client private key certificate (client.p12) must be accessible to the custom software that makes a secure WebSocket (wss://) connection to the DynaFlex II PED.
	
	For Serial devices, address should be in the form:
	PORT=**[PORT]**,
	BAUDRATE=**[BAUDRATE]**,
	DATABITS=**[DATABITS]**,
	PARITY=**[PARITY]**,
	STOPBITS=**[STOPBITS]**,
	HANDSHAKE=**[HANDSHAKE]**,
	STARTINGBYTE=**[STARTINGBYTE]**,
	ENDINGBYTE=**[ENDINGBYTE]**,
	CRCMODE=**[CRCMODE]**

model	Model name for the device.
name	Unique name for the device to distinguish between multiple devices of the same model.
serial	Serial number for the device.

Parameter	Description
certificateInfo	Certificate information for TLS Trust and WebSocket WSS connection. Optional. See the following for details on installing a certificate chain: ***D998200550 DynaFlex II PED Using Wireless LAN Guide***. The client private key certificate (client.p12) must be accessible to the custom software that makes a secure WebSocket (wss://) connection to the DynaFlex II PED. The client.p12 certificate is not needed if the connection type is not secure WebSocket (ws://)

Return Value:
Returns an IDevice.

3.3 createMMSDevice

This function creates an instance of an MMS type of device, DynaFlex Family.

```
MMXDevice CoreAPI.createMMSDevice();
```

Return Value:
Returns an `MMXDevice`.

3.4 createPPSCRA

This function creates an instance of an MTPPSCRA type of device.
The API's of MagTek Universal SDK do not apply. See ***D998200078 IPAD, DynaPro, DynaPro Go, and DynaPro Mini PIN Encryption Devices Programmer's Reference*** for the MTPPSCRA API.

```
MTPPSCRA CoreAPI.createPPSCRA();
```

Return Value:
Returns an `MTPPSCRA`.

3.5 createSCRA

This function creates an instance of a MTSCRA type of device.
The API's of MagTek Universal SDK do not apply. See ***D99875723 uDynamo, Dynamag, DynaMAX, eDynamo, mDynamo, Insert, DynaWave, iDynamo 6*** for the MTSCRA API.

```
MTSCRA CoreAPI.createSCRA();
```

Return Value:
Returns an `MTSCRA`.

3.6 getDeviceList

This function returns a list of IDevice. IDevice is the base for utilizing the MagTek Universal SDK interface.

```
List<IDevice> CoreAPI.getDeviceList();
List<IDevice> CoreAPI.getDeviceList(DeviceType deviceType);
```

```
List<IDevice> CoreAPI.getDeviceList(List<DeviceType> deviceTypes);
```

Parameter	Description
deviceType	An enum for the type of MagTek readers which the SDK will control.
deviceTypes	An enum list for the type of MagTek readers which the SDK will control.

Return Value:
Returns a list of IDevice.

3.7 getAPIVersion

This function returns the API version.

```
int CoreAPI.getAPIVersion();
```

Return Value:
Returns an integer representing the API version.

3.8 setMQTTBrokerInfo

This function sets the MQTT (Message Queuing Telemetry Transport) broker information. Call prior to device discovery.

```
void setMQTTBrokerInfo(
      string uri,
      string username = null,
      string password = null);
```

Parameter	Description
uri	URI included the port. Support URIs: • TCP: “test.mosquitto.org”, “test.mosquitto.org:1883”, “mqtt://test.mosquitto.org:1883”, “mqtt://broker.emqx.io:1883” • TCP (Authenticated): “test.mosquitto.org:1884”, “mqtt://test.mosquitto.org:1884” • TCP (Encrypted): “mqtts://test.mosquitto.org:8886”, “mqtts://broker.emqx.io:8883” • TCP (Encrypted & Authenticated): “mqtts://test.mosquitto.org:8885” • WebSocket: “ws://test.mosquitto.org:8080”, “ws://broker.emqx.io:8083” • WebSocket (Encrypted): “wss://test.mosquitto.org:8081” , “wss://broker.emqx.io:8084” • WebSocket (Authenticated): “ws://test.mosquitto.org:8090” • WebSocket, (Encrypted & Authenticated): “wss://test.mosquitto.org:8091”
username	Username
password	Password

Return Value: None

3.9 setMQTTClientID

This function sets the MQTT client ID to establish a connection.

```
void setMQTTClientID(string clientID);
```

Parameter	Description
clientID	Client ID. If not set, the default value is [HostName]-[RandomUUID]

Return Value: None

3.10 setMQTTClientCertificateInfo

This function sets the MQTT certificate information.

```
void setMQTTClientCertificateInfo(CertificateInfo certificateInfo);
```

Parameter	Description
certificateInfo	If client certificate is required when initiating a connection to the MQTT broker, this shall be used to establish the connection. If not set, the default value is NULL.

Return Value: None

3.11 setMQTTDeviceDiscoveryTimeout

This function sets the MQTTDeviceDiscoveryTimeout value.

```
void setMQTTDeviceDiscoveryTimeout(int timeout);
```

Parameter	Description
timeout	Time out in milliseconds. Default value = 5000 if not set. A call to getDeviceList() does not return the list until this time has expired.

Return Value: None

3.12 startMQTTDeviceStatusMonitoring

This function sets the MQTT callback interface instance.

After connecting to the MQTT broker, the MQTT Device Status Monitoring subscribes to "<MQTTSubscribeTopic>/#". The wildcard "/#" is automatically appended.

Example:
subscribe topic = "MagTek/Server/DynaFlexIIPED"
monitored subscription = "MagTek/Server/DynaFlexIIPED/#".

When a device status updates to “connected” or “disconnected”, OnConnected() and OnDisconnected() are invoked respectively.

This method can be called whether or not there is an active connection to an MQTT device.

```
void startMQTTDeviceStatusMonitoring(
      IMQTTDeviceStatusCallback callback);
```

Parameter	Description
callback	Name of a class or structure that implements the IMQTTDeviceStatusCallback interface events. Call after calling getDeviceList().

Return Value: None

3.13 setMQTTPublishTopic

This function sets the base value for MQTTPublishTopic.. When connected to the MQTT broker, the composed topic is in the format:
“<basetopic>/<DeviceID>/MMSMessage”.

Example:
Base topic = “MagTek/Device/DynaFlexIIPED/”
Full topic = “MagTek/Device/DynaFlexIIPED/B51E72D/MMSMessage”

```
void setMQTTPublishTopic(string topic);
```

Parameter	Description
topic	Base topic for which to publish messages. Full topic is composed by the SDK.

Return Value: None

3.14 setMQTTQos

This function sets the MQTTQoS value.

```
void setMQTTQos(int qos);
```

Parameter	Description
qos	The quality of service level for publishing messages. Range: 0 = At most once (default) 1 = At least once 2 = Exactly once

Return Value: None

3.15 setMQTTSubscribeTopic

This function sets the base value for MQTTSubscribeTopic. When connected to the device, the composed topic is in the format:
"<basetopic>/<DeviceID>/MMSMessage".

Example:
Base topic = "MagTek/Server/DynaFlexIIPED/"
Full topic = "MagTek/Server/DynaFlexIIPED/B51E72D/MMSMessage"

```
void setMQTTSubscribeTopic(string topic);
```

Parameter	Description
topic	Base topic for which to subscribe for messages. Full topic is composed by the SDK.

Return Value: None

3.16 stopMQTTDeviceStatusMonitoring

This function stops the MQTT device status monitoring process. If called when already running, the process shall be stopped immediately. This method can be called whether or not there is an active connection to an MQTT device.

```
void stopMQTTDeviceStatusMonitoring();
```

Return Value: None

3.17 setSystemStatusCallback

This function sets the SystemStatusCallback global value.

```
void setSystemStatusCallback(ISystemStatusCallback callback);
```

Parameter	Description
callback	Name of a class or structure that implements the ISystemStatusCallback interface event. Call before starting any MQTT communication. When there is an error, the interface's OnError() event is invoked.

Return Value: None

4 IDevice

Create an instance of the IDevice from CoreAPI.getDeviceList() or from CoreAPI.createDevice(). Then use the functions described in this chapter.

4.1 cancelTransaction

This function cancels a transaction. A transaction can only be cancelled before a card is presented.

```
bool IDevice.cancelTransaction();
```

Return Value:
Returns true if cancelled. Otherwise, returns false.

4.2 getCapabilities

This function retrieves the capabilities of the device.

```
IDeviceCapabilities IDevice.getCapabilities();
```

Return Value:
Returns **IDeviceCapabilities**

4.3 getConnectionInfo

This function retrieves the connection information of the device.

```
ConnectionInfo IDevice.getConnectionInfo();
```

Return Value:
Returns ConnectionInfo

4.4 getConnectionState

This function retrieves the connection state of the device.

```
ConnectionState IDevice.getConnectionState();
```

Return Value:
Returns ConnectionState

4.5 getDeviceConfiguration

This function allows the host to get an **IDeviceConfiguration** to configure the device.

```
IDeviceConfiguration IDevice.getDeviceConfiguration();
```

Return Value:
Returns **IDeviceConfiguration**.

4.6 getDeviceControl

This function retrieves the device control interface to the device.

```
IDeviceControl IDevice.getDeviceControl();
```

Return Value:
Returns **IDeviceControl**

4.7 getDeviceInfo

This function returns an information class of the device.

```
DeviceInfo IDevice.getDeviceInfo();
```

Return Value:
Returns **DeviceInfo .**

4.8 Name

This function returns the name of the device assigned from createDevice().

```
string IDevice.Name();
```

Return Value:
Returns the name of the device.

4.9 requestPAN

This function prompts the user to present their card and enter a PIN. A card is presented so that the device can retrieve the PAN, which is used for Format blocks requiring a PAN. The encrypted PIN block (EPB) will be returned in the event **OnEvent()**. The data byte array may be passed to builder function PANDataBuilder.GetPANData();

For DynaFlex devices, this function starts a PIN session on the first call and shall be called again to send the PIN status to the device for completing the PIN session.

```
bool IDevice.requestPAN(PANRequest panRequest, PINRequest pinRequest);
```

PANRequest:

Parameter	Type	Description
Timeout	byte	Wait time in seconds.

Parameter	Type	Description
PaymentMethods	List of PaymentMethod	List of the PaymentMethod enumeration. `MSR` = For magnetic stripe cards. `Contact` = For EMV chip cards. `Contactless` = For NFC contactless cards. `ManualEntry` = Manually entry, no card. When set, other payment methods must not be included. `Barcode` = For barcode. `BarcodeEncrypted` = For barcode with encrypted response. `AppleVAS` = For Apple VAS. `GoogleVAS` = For Apple VAS. `NFC` = For NFC tag.

`PINRequest:`

Parameter	Type	Description
Timeout	byte	Wait time in seconds.
PINMode	byte	PIN mode. `0x00` = Enter PIN `0x01` = Enter PIN Amount `0x02` = Reenter PIN Amount `0x03` = Reenter PIN `0x04` = Verify PIN For DynaFlex devices this is the User Interface Sequence: `0x01` = Present Card / Enter PIN (start session) `0x04` = Present Card / Enter PIN / Enter PIN Again (start session) On the second call to requestPAN(), send the PIN status: `0xFD` = Cancel PIN Session (end session) `0xFE` = PIN Entry Failed (end session) `0xFF` = PIN Entry Successful (end session)
MinLength	byte	Minimum length of accepted PIN (>= 4).
MaxLength	byte	Maximum length of accepted PIN (=< 12).
Tone	byte	Tone to play when prompting for the PIN. Usage: `0x00` = No sound `0x01` = One beep `0x02` = Two beeps

Parameter	Type	Description
Format	byte	ISO format for the PIN block. 0x00 = ISO Format 0 0x01 = ISO Format 1 0x03 = ISO Format 3 0x04 = ISO Format 4
PAN	String	First 12 digits of the Primary Account Number. Leave blank if not required by the ISO format for the PIN block.

Return Value:
Returns true if successful. Otherwise, returns false.

4.10 requestPIN

This function prompts the user to enter a PIN. The host must supply the PAN if the Format block selected requires a PAN. The encrypted PIN block (EPB) will be returned in the event **OnEvent()**. The data byte array may be passed to builder function PINDataBuilder.GetPINData();

For DynaFlex devices, this function starts a PIN session on the first call and shall be called again to send the PIN status to the device for completing the PIN session.

```
bool IDevice.requestPIN(PINRequest pinRequest);
```

PINRequest:

Parameter	Type	Description
Timeout	byte	Wait time in seconds.
PINMode	byte	PIN mode. 0x00 - Enter PIN 0x01 = Enter PIN Amount 0x02 = Reenter PIN Amount 0x03 = Reenter PIN 0x04 = Verify PIN For DynaFlex devices this is the User Interface Sequence: 0x00 = Enter PIN (start session) 0x02 = PIN Incorrect, Try Again (continue session) 0x03 = Enter PIN / Enter PIN Again (start session) 0x05 = Enter PIN Again (continue session) On the second call to requestPIN(), send the PIN status: 0xFD = Cancel PIN Session (end session) 0xFE = PIN Entry Failed (end session) 0xFF = PIN Entry Successful (end session)
MinLength	byte	Minimum length of accepted PIN (>= 4).
MaxLength	byte	Maximum length of accepted PIN (=< 12).

Parameter	Type	Description
Tone	byte	Tone to play when prompting for the PIN. 0x00 = No sound 0x01 = One beep 0x02 = Two beeps
Format	byte	ISO format for the PIN block. 0x00 = ISO Format 0 0x01 = ISO Format 1 0x03 = ISO Format 3 0x04 = ISO Format 4
PAN	String	First 12 digits of the Primary Account Number. Leave blank if not required by the ISO format for the PIN block.

Return Value:
Returns true if successful. Otherwise, returns false.

4.11 requestSignature

This function prompts the user to enter a signature. The response data will be returned in the event **OnEvent**.

```
bool IDevice.requestSignature(byte timeout);
```

Parameter	Description
timeout	Time in seconds for the operation to complete. 0x01 to 0xFF

Return Value:
Returns true of success. Otherwise, returns false.

4.12 sendAuthorization

This function sends the Authorization Response Code (ARPC) blob to the device. The response data will be returned in the event **OnEvent()**. See **15.23C.4 Sample Code** for how to process an EMV transaction.

```
bool IDevice.sendAuthorization(IData data);
```

Parameter	Description
data	Contains ARPC blob.

Return Value:
Returns true if successful. Otherwise, returns false.

4.13 sendClassicNFCCommand

This function sends a command to an NFC Mifare Classic Tag type 2. The NFC tag must first be activated by calling startTransaction() with NFC enabled.

```
bool IDevice.sendClassicNFCCommand(
```

```
IData data,
bool lastCommand,
bool encrypt);
```

Parameter	Description
data	Command to send to the NFC tag. For details of the command see NFC commands table below or ***D998200383 DynaFlex Family Programmer's Manual (COMMANDS) section NFC/Mifare Pass Through Commands***
lastCommand	Determines if this is the last NFC command to complete the operation. `true` = This is the last command. Device will provide a single beep after receiving a successful response from the NFC tag. To send subsequent commands, the NFC tag must be activated by calling startTransaction() with NFC enabled. `false` = Expect more commands (Default). Either set to true or false, if the NFC tag command fails, device will provide a double beep.
encrypt	Determines if data returned is to be encrypted. `true` = Encrypt data `false` = Do not encrypt data (Default)

Return Value:
Returns true if successful. Otherwise, returns false.

NFC Classic Commands

Command	Length	Field Value
Mifare Read	11	Byte 0 – 0x30 – Read Command Byte 1- Sector Number to Read Byte 2 – Start Block Number Byte 3 – End Block Number Byte 4 – Key Type, 0 = A, 1 = B Byte 5 to 10 = 6 Byte Key
Mifare Write	var	Byte 0 – 0xA0 – Write Command Byte 1- Sector Number to Write Byte 2 – Start Block Number Byte 3 – End Block Number Byte 4 – Key Type 0 = A, 1 = B Byte 5 to 10 = 6 Byte Key Byte 11 to x = Variable length Byte Data (16 bytes per block)

Command	Length	Field Value
Mifare Increment	14	Byte 0 – 0xC1 – Increment Command Byte 1 – Source Sector Number Byte 2- Source Block number Byte 3 – Key Type 0 = A, 1 = B Byte 4 to 9 = 6 Byte Key Byte 10 to 13 = 4 Byte Operand
Mifare Decrement	14	Byte 0 – 0xC0 – Decrement Command Byte 1 – Source Sector Number Byte 2- Source Block number Byte 3 – Key Type 0 = A, 1 = B Byte 4 to 9 = 6 Byte Key Byte 10 to 13 = 4 Byte Operand
Mifare Restore	10	Byte 0 – 0xC2 – Restore Command Byte 1 – Source Sector Number Byte 2 - Source Block number Byte 3 – Key Type 0 = A, 1 = B Byte 4 to 9 = 6 Byte Key
Mifare Transfer	10	Byte 0 – 0xB0 – Write the value from the Transfer Buffer into destination block number Byte 1 – Destination Sector Number Byte 2 - Destination Block number Byte 3 – Key Type 0 = A, 1 = B Byte 4 to 9 = 6 Byte Key

Response Data For NFC Mifare DESFire Light Tag Type 4

Tag	Len	Value / Description
81	var	Tag Response Code. Byte 0 = 0x00 = Success Byte 0 = 0x01 = I/O Failed Byte 0 = 0x02 Authentication Failed Byte 1 = 0x01 = Block that Failed (optional)

<table>
<tr><th>Tag</th><th>Len</th><th>Value / Description</th></tr>
<tr><td>82</td><td>var</td><td>Encryption Control Payload tag data.
• Data of R-APDU

If unencrypted:
<table>
<tr><th>Tag</th><th>Len</th><th>Value / Description</th></tr>
<tr><td>FC</td><td>var</td><td>NFC Data Container</td></tr>
<tr><td>/DF7A</td><td>var</td><td>NFC Data</td></tr>
</table>

If encrypted:
<table>
<tr><th>Tag</th><th>Len</th><th>Value / Description</th></tr>
<tr><td>/DFDF59</td><td>var</td><td>Encrypted Data Primitive to be decrypted.</td></tr>
<tr><td>/DFDF50</td><td>var</td><td>Encrypted Data KSN</td></tr>
<tr><td>/DFDF51</td><td>01</td><td>Encrypted Data Encryption Type</td></tr>
</table>
</td></tr>
</table>

Example Unencrypted Payload

```
81 0100 (Tag Respons Code)
82 82036D (Encryption Control)
   FC 820369 (NFC Data Container)
      DF7A 820364 (NFC Data)
           031391010F5504 7777772E6D616774656B2E636F6D FE00. . .
                          www.magtek.com
```

Example Encrypted Payload

```
81 0100 (Tag Response code)
82 820389 (Encryption Control)
   DFDF59 820370 (Encrypted Data Primitive)
          03679DC03B4CA607E3A7D2B52C8E9F1B5CD3D85E7368425. . .
   DFDF50 0A (Encrypted Data KSN)
          FFFF9876543210200047
   DFDF51 01 (Encrypted Data Type)
          80
```

4.14 sendDESFireNFCCommand

This function sends a command to an NFC Mifare DESFire Light Tag Type 4. The NFC tag must first be activated by calling startTransaction() with NFC enabled.

```
bool IDevice.sendDESFireNFCCommand(
      IData data,
      bool lastCommand,
      bool encrypt);
```

Parameter	Description
data	Command to send to the NFC tag. See DESFire Data Sheet (MF2DLHX0). Should follow ISO 7816-4 APDU format. For details of the command see ***D998200383 DynaFlex Family Programmer's Manual (COMMANDS) section NFC/Mifare Pass Through Commands***
lastCommand	Determines if this is the last NFC command to complete the operation. `true` = This is the last command. Device will provide a single beep after receiving a successful response from the NFC tag. To send subsequent commands, the NFC tag must be activated by calling startTransaction() with NFC enabled. `false` = Expect more commands (Default). Either set to true or false, if the NFC tag command fails, device will provide a double beep.
encrypt	Determines if data returned is to be encrypted. `true` = Encrypt data `false` = Do not encrypt data (Default)

Return Value:
Returns true if successful. Otherwise, returns false.

Response Data For NFC Mifare DESFire Light Tag Type 4

Tag	Len	Value / Description
81	02	Tag Response (SW1 SW2). See DESFire Data Sheet (MF2DLHX0). Should follow ISO 7816-4 APDU format. • SW1 and SW2 of R-APDU If card is not able to respond: • SW1 = 0x64, SW2 = 0x00

<table>
<tr><th>Tag</th><th>Len</th><th>Value / Description</th></tr>
<tr><td>82</td><td>var</td><td>Encryption Control Payload tag data
• Data of R-APDU

If unencrypted:
<table>
<tr><th>Tag</th><th>Len</th><th>Value / Description</th></tr>
<tr><td>FC</td><td>var</td><td>NFC Data Container</td></tr>
<tr><td>/DF7A</td><td>var</td><td>NFC Data</td></tr>
</table>

If encrypted:
<table>
<tr><th>Tag</th><th>Len</th><th>Value / Description</th></tr>
<tr><td>/DFDF59</td><td>var</td><td>Encrypted Data Primitive to be decrypted.</td></tr>
<tr><td>/DFDF50</td><td>var</td><td>Encrypted Data KSN</td></tr>
<tr><td>/DFDF51</td><td>01</td><td>Encrypted Data Encryption Type</td></tr>
</table>
</td></tr>
</table>

Example Unencrypted Payload

```
81 0100 (Tag Respons Code)
82 82036D (Encryption Control)
   FC 820369 (NFC Data Container)
      DF7A 820364 (NFC Data)
           031391010F5504 7777772E6D616774656B2E636F6D FE00. . .
                          www.magtek.com
```

Example Encrypted Payload

```
81 0100 (Tag Response code)
82 820389 (Encryption Control)
   DFDF59 820370 (Encrypted Data Primitive)
          03679DC03B4CA607E3A7D2B52C8E9F1B5CD3D85E7368425. . .
   DFDF50 0A (Encrypted Data KSN)
          FFFF9876543210200047
   DFDF51 01 (Encrypted Data Type)
          80
```

4.15 sendNFCCommand

This function sends a command to an NFC tag type 2. The NFC tag must first be activated by calling startTransaction() with NFC enabled.

```
bool IDevice.sendNFCCommand(
      IData data,
      bool lastCommand,
      bool encrypt);
```

Parameter	Description
data	Command to send to the NFC tag. • `Get Version` • `Read` • `Fast Read` • `Write` • `Compatibility Write` • `Read_Cnt` • `PWD_Auth` • `Read_Sig` For details of the command see the NFC commands table below or ***D998200383 DynaFlex Family Programmer's Manual (COMMANDS) section NFC/Mifare Pass Through Commands***
lastCommand	Determines if this is the last NFC command to complete the operation. `true` = This is the last command. Device will provide a single beep after receiving a successful response from the NFC tag. To send subsequent commands, the NFC tag must be activated by calling startTransaction() with NFC enabled. `false` = Expect more commands (Default). Either set to true or false, if the NFC tag command fails, device will provide a double beep.
encrypt	Determines if data returned is to be encrypted. `true` = Encrypt data `false` = Do not encrypt data (Default)

Return Value:
Returns true if successful. Otherwise, returns false.

NFC Commands

Command	Length	Field Value
Get Version	1	The GET_VERSION command is used to retrieve information on the NTAG family, the product version, storage size and other product data required to identify the specific NTAG21x. Byte 0 = 0x60

Command	Length	Field Value
Read	2	The READ command requires a start page address, and returns the 16 bytes of four NTAG21x pages. For example, if address is 03h then pages 03h, 04h, 05h, 06h are returned. Special conditions apply if the READ command address is near the end of the accessible memory area. The special conditions also apply if at least part of the addressed pages is within a password protected area. Byte 0 = 0x30 Byte 1 = Start Page Address
Fast Read	3	The FAST_READ command requires a start page address and an end page address and returns the all n*4 bytes of the addressed pages. For example, if the start address is 03h and the end address is 07h then pages 03h, 04h, 05h, 06h and 07h are returned. Byte 0 = 0x3A Byte 2 = Start Page Address Byte 3 = End Page Address
Write	6	The WRITE command requires a block address, and writes 4 bytes of data into the addressed NTAG21x page. Byte 0 = 0xA2 Byte 1 = Address to Write Byte 2 to 5 = 4 Bytes of Data to Write
Compatibility Write	18	The COMPATIBILITY_WRITE command is implemented to guarantee interoperability with the established MIFARE Classic PCD infrastructure, in case of coexistence of ticketing and NFC applications. Even though 16 bytes are transferred to NTAG21x, only the least significant 4 bytes (bytes 0 to 3) are written to the specified address. Set all the remaining bytes, 04h to 0Fh, to logic 00h. Byte 0 = 0xA0 Byte 1 = Address to Write Byte 2 to 17 = 16 Bytes of Data to Write (only least significant 4 bytes are written) Note: This command is sent in 2 steps, which the Firmware will handle (1) <CMD><Address to Write><CRCH><CRCL> (2) <16 Bytes of Data to Write><CRCH><CRCL>

Command	Length	Field Value
READ_CNT	2	The READ_CNT command is used to read out the current value of the NFC one-way counter of the NTAG213, NTAG215 and NTAG216. The command has a single argument specifying the counter number and returns the 24-bit counter value of the corresponding counter. If the NFC_CNT_PWD_PROT bit is set to 1b the counter is password protected and can only be read with the READ_CNT command after a previous valid password authentication Byte 0 = 0x39 Byte 1 = 0x02 (NFC Counter Address)
PWD_AUTH	5	A protected memory area can be accessed only after a successful password verification using the PWD_AUTH command. The AUTH0 configuration byte defines the protected area. It specifies the first page that the password mechanism protects. The level of protection can be configured using the PROT bit either for write protection or read/write protection. The PWD_AUTH command takes the password as parameter and, if successful, returns the password authentication acknowledge, PACK. By setting the AUTHLIM configuration bits to a value larger than 000b, the number of unsuccessful password verifications can be limited. Each unsuccessful authentication is then counted in a counter featuring anti-tearing support. After reaching the limit of unsuccessful attempts, the memory access specified in PROT, is no longer possible. Byte 0 = 0x1B Byte 1..4 = password (4 bytes)
READ_SIG	2	The READ_SIG command returns an IC specific, 32-byte ECC signature, to verify NXP Semiconductors as the silicon vendor. The signature is programmed at chip production and cannot be changed afterwards. Byte 0 = 0x3C Byte 1 = 0x00, RFU

Response Data For NFC Tag Type 2

<table>
<tr><th>Tag</th><th>Len</th><th>Value / Description</th></tr>
<tr><td>81</td><td>01</td><td>Tag Response Code
0x00 = Success
0x01 = Failed</td></tr>
<tr><td>82</td><td>var</td><td>Encryption Control Payload

If unencrypted:
<table>
<tr><th>Tag</th><th>Len</th><th>Value / Description</th></tr>
<tr><td>FC</td><td>var</td><td>NFC Data Container</td></tr>
<tr><td>/DF7A</td><td>var</td><td>NFC Data</td></tr>
</table>
If encrypted:
<table>
<tr><th>Tag</th><th>Len</th><th>Value / Description</th></tr>
<tr><td>/DFDF59</td><td>var</td><td>Encrypted Data Primitive to be decrypted.</td></tr>
<tr><td>/DFDF50</td><td>var</td><td>Encrypted Data KSN</td></tr>
<tr><td>/DFDF51</td><td>01</td><td>Encrypted Data Encryption Type</td></tr>
</table></td></tr>
</table>

Example Unencrypted Payload

```
81 0100 (Tag Respons Code)
82 82036D (Encryption Control)
   FC 820369 (NFC Data Container)
      DF7A 820364 (NFC Data)
           031391010F5504 7777772E6D616774656B2E636F6D FE00. . .
                          www.magtek.com
```

Example Encrypted Payload

```
81 0100 (Tag Response code)
82 820389 (Encryption Control)
   DFDF59 820370 (Encrypted Data Primitive)
          03679DC03B4CA607E3A7D2B52C8E9F1B5CD3D85E7368425. . .
   DFDF50 0A (Encrypted Data KSN)
          FFFF9876543210200047
   DFDF51 01 (Encrypted Data Type)
          80
```

4.16 sendPlusNFCCommand

This function sends a command to a Mifare Plus NFC tag. The NFC tag must first be activated by calling startTransaction() with NFC enabled.

```
bool IDevice.sendPlusNFCCommand(
      IData data,
      bool lastCommand,
      bool encrypt);
```

Parameter	Description
data	Command to send to the NFC tag. • `Get Version` • `Read` • `Fast Read` • `Write` • `Compatibility Write` • `Read_Cnt` • `PWD_Auth` • `Read_Sig` For details of the command see the NFC commands table below or ***D998200383 DynaFlex Family Programmer's Manual (COMMANDS) section NFC/Mifare Pass Through Commands***
lastCommand	Determines if this is the last NFC command to complete the operation. `true` = This is the last command. Device will provide a single beep after receiving a successful response from the NFC tag. To send subsequent commands, the NFC tag must be activated by calling startTransaction() with NFC enabled. `false` = Expect more commands (Default). Either set to true or false, if the NFC tag command fails, device will provide a double beep.
encrypt	Determines if data returned is to be encrypted. `true` = Encrypt data `false` = Do not encrypt data (Default)

Return Value:
Returns true if successful. Otherwise, returns false.

4.17 sendSelection

This function sends a user selection to the device. The response data will be returned in the event **OnEvent**.

```
bool IDevice.sendSelection(IData data);
```

Parameter for data	Description
Byte 0	Status of User Selection: `0x00` = User Selection Request completed, see Selection Result `0x01` = User Selection Request aborted, cancelled by user `0x02` = User Selection Request aborted, timeout

Parameter for data	Description
Byte 1	The menu item selected by the user. This is a single byte zero based binary value.

Return Value:
Returns true if successful. Otherwise, returns false.

4.18 startTransaction

This function starts a transaction. This function will automatically handle the opening and closing of a device. The transaction will be processed through multiple calls to the event **OnEvent()**. See **15.23C.4 Sample Code** for how to process an EMV transaction.

```
bool IDevice.startTransaction(
```

ITransaction `transaction);`

Parameter	Description
transaction	An interface that contains the parameters for the transaction.

Return Value:
Returns true if the transaction started successfully. Otherwise, returns false.

4.19 subscribeAll

This function allows the host to be notified of all events sent by the device.

```
bool IDevice.subscribeAll(IEventSubscriber eventCallback);
```

Parameter	Description
eventCallback	Name of a class or structure that implements the IEventSubscriber interface event.

Return Value:
Returns true if successful. Otherwise, returns false.

4.20 unsubscribeAll

This function allows the host to no longer receive any events sent by the device.

```
bool IDevice.unsubscribeAll(IEventSubscriber eventCallback);
```

Parameter	Description
eventCallback	Name of a class or structure that implements the **IEventSubscriber** Delegate interface event.

Return Value:
Returns true if successful. Otherwise, returns false.

5 IDeviceCapabilities

Create an instance of the **IDeviceCapabilities** using **IDevice.getCapabilities()**. Then use the functions described in this chapter.

5.1 AutoSignatureCapture

This property returns true if the device is capable of automatically capturing a signature during a transaction.

```
bool IDeviceCapabilities.AutoSignatureCapture();
```

Return Value:
Returns true if device is capable of automatically capturing a signature. Otherwise, returns false.

5.2 BatteryBackedClock

This property returns true if the device is equipped with a battery that preserves the internal clock when not powered by a host system or charging.

```
bool IDeviceCapabilities.BatteryBackedClock();
```

Return Value:
Returns true if device is equipped with a battery backed clock. Otherwise, returns false.

5.3 Display

This property returns true if the device is equipped with display.

```
bool IDeviceCapabilities.Display();
```

Return Value:
Returns true if device is equipped with a display. Otherwise, returns false.

5.4 MSRPowerSaver

This property returns true if the device has the option to disable or enable the magnetic stripe reader head (MSR). The MSR may be powered down while the device is idle to minimize power consumption.

```
bool IDeviceCapabilities.MSRPowerSaver();
```

Return Value:
Returns true if device supports MSR power saver. Otherwise, returns false.

5.5 PaymentMethods

This property returns an enumerated list of payment methods supported by the device.

```
List<PaymentMethod> IDeviceCapabilities.PaymentMethods();
```

Return Value:
Returns a list of PaymentMethod

5.6 PINPad

This property returns true if the device is equipped with a PIN Pad.

```
bool IDeviceCapabilities.PINPad();
```

Return Value:
Returns true if device is equipped with a PIN Pad. Otherwise, returns false.

5.7 Signature

This property returns true if the device is equipped with signature capture.

```
bool IDeviceCapabilities.Signature();
```

Return Value:
Returns true if device is equipped with signature capture. Otherwise, returns false.

5.8 SRED

This property returns true if the device supports Secure Reading and Exchange of Data.

```
bool IDeviceCapabilities.SRED();
```

Return Value:
Returns true if device supports SRED. Otherwise, returns false.

6 IDeviceControl

Create an instance of the **IDeviceControl** using **IDevice.getDeviceControl()**. Then use the function calls described in this chapter.

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

- **Asynchronous** functions return data in the event handlers in section **IEventSubscriber Delegate**.
- **Synchronous** functions return data in the return value. If the data is not available immediately, the call will block until the wait time has elapsed.

6.1 close

This function closes the connection to the device.

```
bool IDeviceControl.close();
```

Return Value:
Returns true if successful. Otherwise, returns false.

6.2 deviceReset

This function resets the device. This is equivalent to a power reset. After the reset, connection to the device will need to be re-established.

```
bool IDeviceControl.deviceReset();
```

Return Value:
Returns true if successful. Otherwise, returns false.

6.3 displayMessage

This function displays a predefined message on the device.

```
bool IDeviceControl.displayMessage(byte messageID, byte timeout);
```

Parameter	Description
messageID	Value representing the message. Examples: 0x00 - reserved, do not use. 0x01 - "AMOUNT" 0x02 - "AMOUNT OK?" 0x03 - "APPROVED" 0x04 - "CALL YOUR BANK" 0x05 - "CANCEL OR ENTER" 0x06 - "CARD ERROR" 0x07 - "DECLINED"

Parameter	Description
timeout	Timeout in seconds for the device to display the message. `0x00` = Infinite timeout. Device leaves the requested message on the display until the host initiates a change. `0x01 to 0xFF` = Timeout in seconds for the device to display the message.

Return Value:
Returns true if successful. Otherwise, returns false.

6.4 endSession

This function clears session data and returns the device to an idle state.

```
bool IDeviceControl.endSession();
```

Return Value:
Returns true if successful. Otherwise, returns false.

6.5 getInput

This function sends a request for user interaction input to the device. The response data will be returned in the event **OnEvent()**.

```
bool IDeviceControl.getInput();
```

Return Value:
Returns true if successful. Otherwise, returns false.

6.6 open

This function opens a connection to the device.

```
bool IDeviceControl.open();
```

Return Value:
Returns true if successful. Otherwise, returns false.

6.7 playSound

This function instructs the device to play a tone.

```
bool IDeviceControl.playSound(IData data);
```

Parameter	Description
data	Byte array or string data to send to the device.

Return Value:
Returns true if successful. Otherwise, returns false.

6.8 send

This function sends a command to the device. The response will be passed to the event **OnEvent()**.

```
bool IDeviceControl.send(IData data);
```

Parameter	Description
data	Byte array or string data to send to the device. Data must contain the full command as required by the device.

Return Value:
Returns true if successful. Otherwise, returns false.

6.9 sendExtendedCommand

This function sends an extended command to the device. The response will be passed to the event **OnEvent()**.

```
bool IDeviceControl.sendExtendedCommand(IData data);
```

Parameter	Description
data	Byte array or string data to send to the device. Data must contain the full command as required by the device.

Return Value:
Returns true if successful. Otherwise, returns false.

6.10 sendNFCPassThroughCommand

This function sends a raw APDU command from the host application to the NFC target while the device is operating in NFC Pass-Through mode. For NFC Type A or Type B using ISO 14443-4 Protocol.

Used for applications that require direct APDU-level communication, such as mDL, secure element transactions, or custom NFC protocols.

```
bool IDeviceControl.sendNFCPassThroughCommand(IData data);
```

Parameter	Description
data	APDU command data to be transmitted to the NFC target device.

Return Value:
Returns true if successful. Otherwise, returns false.

6.11 sendSync

This function sends a synchronous command to the device. The response from the device will be returned in IResult.

```
IResult IDeviceControl.sendSync(IData data);
```

Parameter	Description
data	Byte array or string data to send to the device.

Return Value:
Returns `IResult`.

```
public interface IResult
{
  StatusCode Status
  IData Data
}
```

6.12 setDateTime

This function sets the date and time for the device.

```
bool IDeviceControl.setDateTime(IData data);
```

Parameter	Description
data	Byte array or string data to send to the device.

Return Value:
Returns true if successful. Otherwise, returns false.

6.13 setLatch

This function send a command to lock or unlock the card latch. The host can choose to lock the card during EMV transactions to limit the possibility of the cardholder prematurely removing the card. The lock can also be enabled while the card is out of the system to block cardholders from inserting a card.

```
bool IDeviceControl.setLatch(bool enableLock);
```

Parameter	Description
enableLock	`false` = unlock the latch in the device. `true` = lock the latch in the device.

Return Value:
Returns true if successful. Otherwise, returns false.

6.14 setLEDStatus

This function controls the status of the device's LED indicators (Light Emitting Diode) to provide visual feedback NFC operation states. Applicable after entering NFC Pass-Through mode. The status is reset after exiting NFC Pass-Through mode.

```
bool IDeviceControl.setLEDStatus(
      byte led,
      byte status);
```

Parameter	Description
led	LED number: 0x00 = LED 1 (Power LED) 0x01 = LED 2 0x02 = LED 3 0x03 = LED 4 0xFF = All 4 LEDs Where the LEDs are numbered 1, 2, 3, 4 counting from the left.
status	LED Status: 0x00 = OFF 0x01 = GREEN ON 0x02 = RED ON

Return Value:
Returns true if successful. Otherwise, returns false.

6.15 setNFCPassThroughMode

This function commands the device to enter NFC Pass-Through Mode. This enables the host application to directly exchange APDU commands and responses with an NFC card or mobile device.

```
bool IDeviceControl.setNFCPassThroughMode(
      byte timeout,
      byte mode);
```

Parameter	Description
timeout	Duration to stay in Pass-Through mode. 0x00 = Indefinite 0x01 to 0xFF = 1 to 255 seconds
mode	Pass-Through mode. 0x00 = Stop Pass-Through Mode. Applicable only when already started. When stopped, the RF Antenna is turned off. 0x01 = Start Pass-Through Mode. This mode can only be used when the device is in Idle state.

Return Value:
Returns true if successful. Otherwise, returns false.

6.16 setNFCPollingMode

This function configures the NFC reader polling behavior used to detect nearby NFC targets such as cards, mobile devices, or mDL wallets. Applicable after entering the Pass-Through mode.

```
bool IDeviceControl.setNFCPollingMode(
      byte timeout,
      byte mode);
```

Parameter	Description
timeout	Duration to stay in Pass-Through mode. 0x00 = Indefinite 0x01 to 0xFF = 1 to 255 seconds
mode	Polling Mode. 0x00 = Stop Polling Mode. Applicable only when already started. 0x01 = Start Polling Mode

Return Value:
Returns true if successful. Otherwise, returns false.

6.17 showBarCode

This function sends a command to show a barcode on the device's display.

```
bool IDeviceControl.showBarCode(
      BarCodeRequest request
      byte timeout
      IData prompt);
```

Parameter	Description
request	BarCodeRequest object containing the barcode data to display.
timeout	Display Time. 0x00 = Indefinite 0x01 to 0xFF = 1 to 255 seconds
prompt	Text to display below the QR code. In Landscape orientation, the limit is approximately 30 characters. In Portrait orientation, the limit is approximately 22 characters.

BarCodeRequest:

Parameter	Type	Description
Type	BarCodeType	Enum to specify the type of barcode
Format	BarCodeFormat	Enum to specify the barcode format
Data	byte[]	Data to encode into a barcode
BlockColor	byte[]	Block color. Use RRGGBB format. 0x000000 = Black
BackgroundColor	byte[]	Background color. Use RRGGBB format. 0xFFFFFF = White
ErrorCorrection	byte	Error Correction 0x00 = Low (default) 0x01 = Medium 0x02 = Quartile 0x03 = High See **ISO/IEC 18004:2015**

Parameter	Type	Description
MaskPattern	byte	Mask Pattern 0x00 to 0x07 = Mask Pattern 0xFF = Device Select Optimal Mask Pattern (default) See **ISO/IEC 18004:2015**
MinVersion	byte	Minimum Version. Must be less than or equal to Maximum Version. 0x01 to 0x28 = Version 1 to Version 40 (0x01 is default) See **ISO/IEC 18004:2015**
MaxVersion	byte	Maximum Version. Must be greater than or equal to Minimum Version. 0x01 to 0x28 = Version 1 to Version 40 (0x28 is default) See **ISO/IEC 18004:2015**

Return Value:
Returns true if successful. Otherwise, returns false.

6.18 showImage

This function sends a command to immediately show an image on the device's display. The image must already be loaded into a slot.

```
bool IDeviceControl.showImage(byte imageID);
```

Parameter	Description
imageID	Usage: 0x01 = show the image at slot 1. 0x02 = show the image at slot 2. 0x03 = show the image at slot 3. 0x04 = show the image at slot 4.

Return Value:
Returns true if successful. Otherwise, returns false.

6.19 showImage

This function sends a command to immediately upload and show an image on the device's display.

```
bool IDeviceControl.showImage(
      ImageData data
      byte timeout);
```

Parameter	Description
data	ImageData object containing the image to display.
timeout	Display Time 0x00 = Indefinite 0x01 to 0xFF = 1 to 255 seconds

```
ImageData
```

Member	Type/ Format	Description
type	ImageType	Enum for image type. Usage: BITMAP = BMP file
data	byte[]	Image encoded data. Images must be BMP format, 160KB or smaller with no compression, maximum 320px by 240px, with color depth 16 color, 256 color, 16-bit color, or 24-bit color. Images smaller than the maximum size are centered on the display. Note images at full screen size must be 16-bit color or lower to meet the size requirement. For details see ***D998200383 DynaFlex Family Programmer's Manual (COMMANDS)***
backgroundColor	byte[3]	Background color in RRGGBB format. `0x000000` = Black `0xFFFFFF` = White

Return Value:
Returns true if successful. Otherwise, returns false.

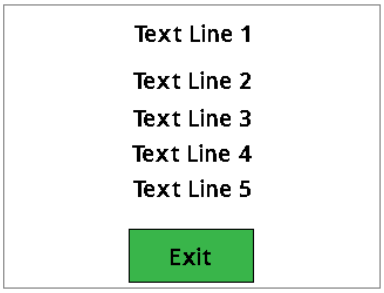
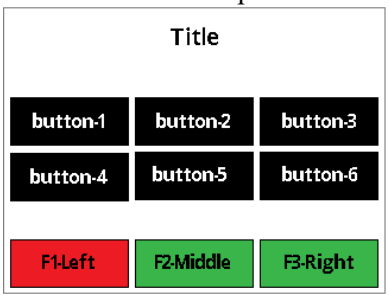
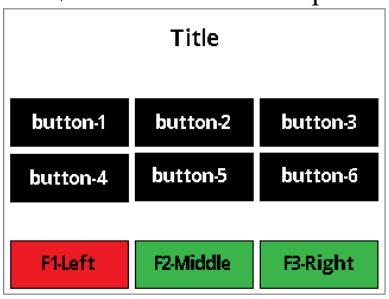
6.20 showUIPage

This function displays custom User Interface (UI) pages on devices that support a screen. It holds parameters for all UI page options of Text Lines, Text Buttons, Amount Buttons, Functional Buttons, and Image. Only one UI page option may be in effect at one time based on the option parameter. See the other subset functions for invoking a specific UI page.

When the UI page is active, the device waits for further action. The device notifies the host when a Button is pressed. Text Buttons and Functional Buttons report as the button number. Amount Buttons report the $ amount in BCD format. A press on a Text line does not produce a report.

```
bool showUIPage(
      byte timeout,
      byte option,
      byte[] titleStringID = null,
      string line1 = "",
      string line2 = "",
      string line3 = "",
      string line4 = "",
      string line5 = "",
      byte[] stringID1 = null,
      byte[] stringID2 = null,
      byte[] stringID3 = null,
      byte[] stringID4 = null,
      byte[] stringID5 = null,
      byte[] stringID6 = null,
      byte[] amount1 = null,
      byte[] amount2 = null,
      byte[] amount3 = null,
      byte[] amount4 = null,
      byte[] amount5 = null,
      byte[] amount6 = null,
      byte[] leftFButtonStringID = null,
      byte[] middleFButtonStringID = null,
      byte[] rightFButtonStringID = null,
      byte leftFButtonColor = 0,
      byte middleFButtonColor = 0,
      byte rightFButtonColor = 0,
      byte[] xPosition = null,
      byte[] yPosition = null,
      byte[] imageData = null);
```

Parameter	Description
timeout	Time in seconds to enable barcode reader. 0x00 = Infinite until the host initiates a change. 0x01 to 0xFF = RFU

Parameter	Description
option	UI page option. 0x00 = Up to 5 lines of text + 1 functional button Middle.  0x01 = Title + up to 6 text buttons + up to 3 functional buttons.  0x02 = Title + up to 6 $Amount buttons + up to 3 functional buttons.  0x03 = Title + custom image + 1 functional button Right.
titleStringID[]	String ID for the title. See list of display user interface strings. This list may vary according to the device configuration.
line1 … line5	Text string for the line. Terminate with null char. To disable a line, set to empty (Default).
stringID1[2] … stringID6[2]	Text string ID for buttons 1 to 6. 2-bytes in length. To disable a button, set to null (Default).

Parameter	Description

amount1[4] … amount6[4]	Value $ Amount in BCD (Binary Coded Decimal) array format for buttons 1 to 6. 4-bytes in length. To disable a button, set to null.
	
	Byte index:
	• [0-2] = dollar value
	• [3] = cents value
	
	BCD positioning:
	87 65 43 . 21
	0x00 0x00 0x00 0x00
	• 8 – hundred thousand
	• 7 – ten thousand
	• 6 – thousand
	• 5 – hundred
	• 4 – ten
	• 3 – one
	• 2 – 1/tenth
	• 1 – 1/hundredth
	
	Range:
	• Min value displayed is `$0.00`
	• Max value displayed is `$999,999.99`
	
	Example:
	`// $0.01`
	`amount1 = new Byte[] { 0x00, 0x00, 0x00, 0x01 };`
	
	`//$100,000.23`
	`amount1 = new Byte[] { 0x10, 0x00, 0x00, 0x23 };`

leftFButtonStringID[2]	String ID for the Left functional button. 2-bytes in length. To disable this button, set to null (Default).
middleFButtonStringID[2]	String ID for the Middle functional button. 2-bytes in length. To disable this button, set to null (Default).
rightFButtonStringID[2]	String ID for the Right functional button. 2-bytes in length. To disable this button, set to null (Default).
leftFButtonColor	Color of Left functional button. `0` = red, `1` = green, `2` = yellow
middleFButtonColor	Color of Middle functional button. `0` = red, `1` = green, `2` = yellow
rightFButtonColor	Color of Right functional button. `0` = red, `1` = green, `2` = yellow

Parameter	Description
xPosition[2]	X Position for image. 2-bytes in length. To display the image in the center of the loading image area, set to null (Default).
yPosition[2]	Y Position for image. 2-bytes in length. To display the image in the center of the loading image area, set to null (Default).
imageData[]	Bitmap data. Image encoded in full BMP file format as defined by Microsoft (e.g, starting with "BM"). If no image provided, set to null (Default).

Return Value:
Returns true if successful. Otherwise, returns false.

6.21 showUIPageWithAmountButtons

This function shows the Amount UI page.

```
bool showUIPageWithAmountButtons(
      byte timeout,
      byte[] titleStringID,
      string[] amountList,
      byte[] leftFButtonStringID,
      byte[] middleFButtonStringID,
      byte[] rightFButtonStringID,
      byte leftFButtonColor,
      byte middleFButtonColor,
      byte rightFButtonColor);

bool showUIPageWithAmountButtons(
      byte timeout,
      byte[] titleStringID,
      string amount1,
      string amount2,
      string amount3,
      string amount4,
      string amount5,
      string amount6,
      byte[] leftFButtonStringID,
      byte[] middleFButtonStringID,
      byte[] rightFButtonStringID,
      byte leftFButtonColor,
      byte middleFButtonColor,
      byte rightFButtonColor);

bool showUIPageWithAmountButtons(
      byte timeout,
      byte[] titleStringID,
      byte[] amount1,
      byte[] amount2,
```

```
byte[] amount3,
byte[] amount4,
byte[] amount5,
byte[] amount6,
byte[] leftFButtonStringID,
byte[] middleFButtonStringID,
byte[] rightFButtonStringID,
byte leftFButtonColor,
byte middleFButtonColor,
byte rightFButtonColor);
```

Parameter	Description
timeout	Time in seconds to enable barcode reader. 0x00 = Infinite until the host initiates a change. 0x01 to 0xFF = RFU
titleStringID[2]	String ID for the title. 2-bytes in length. See list of display user interface strings. This list may vary according to the device configuration.
amountList	List of $ Amount in BCD string format to display for each button.
leftFButtonStringID[2]	String ID for the Left functional button. 2-bytes in length. To disable this button, set to 0.
middleFButtonStringID[2]	String ID for the Middle functional button. 2-bytes in length. To disable this button, set to 0.
rightFButtonStringID[2]	String ID for the Right functional button. 2-bytes in length. To disable this button, set to 0.
leftFButtonColor	Color of Left functional button. 0 = red, 1 = green, 2 = yellow
middleFButtonColor	Color of Middle functional button. 0 = red, 1 = green, 2 = yellow
rightFButtonColor	Color of Right functional button. 0 = red, 1 = green, 2 = yellow
amount1 … amount6	Value $ Amount in BCD string format for buttons 1 to 6. 8-characters in length. To disable a button, set to NULL. Range: • Min value displayed is $0.00 • Max value displayed is $999,999.99 • Decimal needed to display cents • Comma not needed Example: // $0.01 string sAmount1 = "0.01"; // $100,000.23 string sAmount1 = "100000.23";

Parameter	Description

amount1[4] … amount6[4]	Value $ Amount in BCD (Binary Coded Decimal) array format for buttons 1 to 6. 4-bytes in length. To disable a button, set to NULL.
	
	Byte index:
	• [0-2] = dollar value
	• [3] = cents value
	
	BCD positioning:
	87 65 43 . 21
	0x00 0x00 0x00 0x00
	• 8 – hundred thousand
	• 7 – ten thousand
	• 6 – thousand
	• 5 – hundred
	• 4 – ten
	• 3 – one
	• 2 – 1/tenth
	• 1 – 1/hundredth
	
	Range:
	• Min value displayed is `$0.00`
	• Max value displayed is `$999,999.99`
	
	Example:
	`// $0.01`
	`amount1 = new Byte[] { 0x00, 0x00, 0x00, 0x01 };`
	
	`//$100,000.23`
	`amount1 = new Byte[] { 0x10, 0x00, 0x00, 0x23 };`

Return Value:
Returns true if successful. Otherwise, returns false.

6.22 showUIPageWithImage

This function shows the Image UI page.

```
bool showUIPageWithImage(
      byte timeout,
      byte[] titleStringID,
      byte[] rightFButtonStringID,
      byte[] xPosition,
      byte[] yPosition,
      byte[] imageData);
```

Parameter	Description
timeout	Time in seconds to enable barcode reader. 0x00 = Infinite until the host initiates a change. 0x01 to 0xFF = RFU
titleStringID[2]	String ID for the title. 2-bytes in length. See list of display user interface strings. This list may vary according to the device configuration.
rightFButtonStringID[2]	String ID for the Right functional button. 2-bytes in length. To disable this button, set to 0.
xPosition[2]	X Position for image. 2-bytes in length. To display the image in the center of the loading image area, set to null.
yPosition[2]	Y Position for image. 2-bytes in length. To display the image in the center of the loading image area, set to null.
imageData[]	Bitmap data. Image encoded in full BMP file format as defined by Microsoft (e.g, starting with "BM").

Return Value:
Returns true if successful. Otherwise, returns false.

6.23 showUIPageWithTextButtons

This function shows the Text buttons UI page.

```
bool showUIPageWithTextButtons(
      byte timeout,
      byte[] titleStringID,
      byte[] stringID1,
      byte[] stringID2,
      byte[] stringID3,
      byte[] stringID4,
      byte[] stringID5,
      byte[] stringID6,
      byte[] leftFButtonStringID,
      byte[] middleFButtonStringID,
      byte[] rightFButtonStringID,
      byte leftFButtonColor,
      byte middleFButtonColor,
      byte rightFButtonColor);
```

Parameter	Description
timeout	Time in seconds to enable barcode reader. 0x00 = Infinite until the host initiates a change. 0x01 to 0xFF = RFU

Parameter	Description
titleStringID[2]	String ID for the title. See list of display user interface strings. This list may vary according to the device configuration.
stringID1[2] to stringID6[2]	Text string ID for buttons 1 to 6. 2-bytes in length. To disable a button, set to null.
leftFButtonStringID[2]	String ID for the Left functional button. To disable this button, set to null.
middleFButtonStringID[2]	String ID for the Middle functional button. To disable this button, set to null.
rightFButtonStringID[2]	String ID for the Right functional button. To disable this button, set to null.
leftFButtonColor	Color of Left functional button. 0 = red, 1 = green, 2 = yellow
middleFButtonColor	Color of Middle functional button. 0 = red, 1 = green, 2 = yellow
rightFButtonColor	Color of Right functional button. 0 = red, 1 = green, 2 = yellow

Return Value:
Returns true if successful. Otherwise, returns false.

6.24 showUIPageWithTextLines

This function shows the Text lines UI page.

```
bool showUIPageWithTextLines(
      byte timeout,
      string[] lines,
      byte[] middleFButtonStringID);

bool showUIPageWithTextLines(
      byte timeout,
      string line1,
      string line2,
      string line3,
      string line4,
      string line5,
      byte[] middleFButtonStringID);
```

Parameter	Description
timeout	Time in seconds to enable barcode reader. 0x00 = Infinite until the host initiates a change. 0x01 to 0xFF = RFU
lines[]	List of Text strings for lines 1 to 5 in array format. Terminate with NULL char. To disable a line, set to null.

Parameter	Description
line 1 … line5	Text string for lines 1 to 5 in string format. Terminate with NULL char. To disable a line, set to empty.
middleFButtonStringID[2]	String ID for the Middle functional buttons. 2-byte in length. To disable this button, set to null.

Return Value:
Returns true if successful. Otherwise, returns false.

6.25 startBarCodeReader

This function sends a command to start the barcode reader.

```
bool IDeviceControl.startBarCodeReader(
      byte timeout,
      byte encryptionMode);
```

Parameter	Description
timeout	Time in seconds to enable barcode reader. `0x00` = Wait until a barcode is read or **stopBarCodeReader()** is called. `0x01 to 0xFF` = 1 to 255 seconds
encryptionMode	Encrypt payload `0x00` = do not encrypt barcode data `0x01` = encrypt barcode data.

Return Value:
Returns true if successful. Otherwise, returns false.

6.26 startCardEmulation

This function sends a command to start card emulation.

- Compliance: Conforms to ISO/IEC 14443 Type-A and NFC Forum Type 4 standards.
- Passive Operation mode: The NFC reader must generate the field to activate the simulated card.
- Read-Only: The emulated card is read-only and does not support writing.
- Supported Data Type: Supports the URI (URL) data type.

```
bool IDeviceControl.startCardEmulation(
      byte timeout,
      string data);
```

Parameter	Description
timeout	Time in seconds. 0x00 = No timeout or until stopNFCCardEmulation() is called 0x01 to 0xFF = 1 to 255 seconds
data	NFC data to emulate. Limit is <= 254 characters.

Return Value:
Returns true if successful. Otherwise, returns false.

6.27 startPersonalInfoEntry

This function sends a command to prompt a cardholder for customer information. The information captured is later returned in OnEvent() EventType.TouchscreenPersonalInfoEntry.

```
bool IDeviceControl.startPersonalInfoEntry(
      CaptureType captureType,
      bool encrypt);
```

Parameter	Description
captureType	Enum for the type of information to capture.
encrypt	Encrypt payload `false` = do not encrypt data `true` = encrypt data.

Return Value:
Returns true if successful. Otherwise, returns false.

6.28 stopBarCodeReader

This function sends a command to stop the barcode reader. This is applicable only when the timeout value for startBarCodeReader() was set to 0x00.

```
bool IDeviceControl.stopBarCodeReader();
```

Return Value:
Returns true if successful. Otherwise, returns false.

6.29 stopCardEmulation

This function sends a command to stop card emulation.

```
bool IDeviceControl.stopCardEmulation();
```

Return Value:
Returns true if successful. Otherwise, returns false.

6.30 stopPersonalInfoEntry

This function stops the Personal Info Entry process. If called during info entry, the process will be canceled immediately.

```
bool IDeviceControl.stopPersonalInfoEntry();
```

Return Value:
Returns true if successful. Otherwise, returns false.

7 ConnectionInfo

Create an instance of the ConnectionInfo using **getConnectionInfo()**. Then use the function calls described in this chapter.

7.1 getAddress

This function returns the address of the device.

```
string ConnectionInfo.getAddress();
```

Return Value:
Returns the address of the device.

7.2 getConnectionType

This function returns the type of connection Interface for the device.

```
ConnectionType ConnectionInfo.getConnectionType();
```

Return Value:
Returns the **ConnectionType**

7.3 getDeviceType

This function returns the type for the device.

```
DeviceType ConnectionInfo.getDeviceType();
```

Return Value:
Returns the **DeviceType**

7.4 getCertificateInfo

This function returns the type for the device.

```
CertificateInfo ConnectionInfo.getCertificateInfo();
```

Return Value:
Returns the **DeviceType**

8 DeviceInfo

Create an instance of the **DeviceInfo** from **IDevice.getDeviceInfo()**. Then use the function calls described in this chapter.

8.1 getModel

This function returns the model name of the device.

```
string DeviceInfo.getModel();
```

Return Value:
Returns the model name of the device.

8.2 getName

This function returns the name of the device.

```
string DeviceInfo.getName();
```

Return Value:
Returns the name of the device.

8.3 getSerial

This function returns the serial number of the device.

```
string DeviceInfo.getSerial();
```

Return Value:
Returns the serial number of the device.

9 IDeviceConfiguration

Create an instance of the **IDeviceConfiguration** using **getDeviceConfiguration()**. Then use the function calls described in this chapter.

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

- **Asynchronous** functions return data in the event handlers in section **IEventSubscriber Delegate**.
- **Synchronous** functions return data in the return value. If the data is not available immediately, the call will block until the wait time has elapsed.

9.1 deleteFile

This function deletes a file from the device.

```
Int IDeviceConfiguration.deleteFile(byte[] fileID);
```

Parameter	Description
fileID	Byte array for the file ID. For DynaFlex, use a 4-byte file id. 0x02000000 = Custom idle page image 1 0x02000001 = Custom idle page image 2 0x02000002 = Custom idle page image 3 0x02000003 = Custom idle page image 4

Return Value:
Returns 0 if the asynchronous update operation started. Otherwise, returns a non 0 value.

9.2 getChallengeToken

This function retrieves a challenge token from the device. A challenge token consists of a random nonce or timestamp. A challenge token must be used within the time allowed by the device (generally 5 minutes) of being issued. Only one token can be active at a time. Attempts to use a token for requests other than the one specified will cause the token to be revoked/erased.

```
byte[] IDeviceConfiguration.getChallengeToken(byte[] data);
```

Parameter	Description
data	Byte array containing the request ID to be protected.

Return Value:
Returns a byte array containing the challenge token.

9.3 getConfigInfo

This function retrieves device configuration information. For an example, see appendix **15.23B.6 IDeviceConfiguration Walk Through**.

```
byte[]IDeviceConfiguration.getConfigInfo(
      byte configType,
      byte[] data);
```

Parameter	Description
configType	Type of configuration. For DynaFlex, this is the first number of the Property OID.
data	Configuration data to be sent to the device. For DynaFlex, this is the remainder of the constructed OID. For constructing the OID see ***D998200383 DynaFlex Family Programmer's Manual (COMMANDS)***

Return Value:
Returns an array of bytes containing the configuration information.

9.4 getDeviceInfo

This function retrieves device specific information.

```
string IDeviceConfiguration.getDeviceInfo(InfoType infoType);
```

Parameter	Description
infoType	Enumerated information type.

Return Value:
Returns a string value device information.

9.5 getFile

This function sets device configuration information.

```
int IDeviceConfiguration.getFile(
      byte[] fileID,
      IConfigurationCallback callback);
```

Parameter	Description
fileID	Byte array for the file ID. For DynaFlex, use a 4-byte file id.
callback	Name of a class or structure that implements the **IConfigurationCallback** Delegates events.

Return Value:
Returns 0 if the asynchronous configuration operation started. Otherwise, returns a non 0 value.

9.6 getKeyInfo

This function retrieves key information.

```
byte[]IDeviceConfiguration.getKeyInfo(
      byte keyType,
      byte[] data);
```

Parameter	Description
keyType	Type of key. For DynaFlex, use 0.
data	Key data to be sent to the device. For DynaFlex, this is the 2-byte key slot number.

Return Value:
Returns an array of bytes containing the key information.

9.7 sendFile

This function sends a file to the device.

```
int IDeviceConfiguration.sendFile(
      byte[] fileID,
      byte[] data,
      IConfigurationCallback callback);
```

Parameter	Description
fileID	Byte array for the file ID. For DynaFlex, use a 4-byte file id.
data	File contents to be sent to the device.

Parameter	Description
callback	Name of a class or structure that implements the **IConfigurationCallback** Delegates events.

Return Value:
Returns 0 if the asynchronous update operation started. Otherwise, returns a non 0 value.

9.8 sendImage

This function sends an image to the device.

```
int IDeviceConfiguration.sendImage(
      byte imageID,
      byte[] data,
      IConfigurationCallback callback);
```

Parameter	Description
imageID	Value for the image ID. For DynaFlex, use: `1, 2, 3, or 4`
data	Image encoded data. Images must be BMP format, 160KB or smaller with no compression, maximum 320px by 240px, with color depth 16 color, 256 color, 16-bit color, or 24-bit color. Images smaller than the maximum size are centered on the display. Note images at full screen size must be 16-bit color or lower to meet the size requirement. For details see ***D998200383 DynaFlex Family Programmer's Manual (COMMANDS)***
callback	Name of a class or structure that implements the **IConfigurationCallback** Delegates events.

Return Value:
Returns 0 if the asynchronous update operation started. Otherwise, returns a non 0 value.

9.9 sendSecureFile

This function sends a file to the device using a secure command structure.

```
int IDeviceConfiguration.sendFile(
      byte[] fileID,
      byte[] data,
      IConfigurationCallback callback);
```

Parameter	Description
fileID	Byte array for the file ID. For DynaFlex, use a 4-byte file id.
data	File contents to be sent to the device.

Parameter	Description
callback	Name of a class or structure that implements the **IConfigurationCallback** Delegates events.

Return Value:
Returns 0 if the asynchronous update operation started. Otherwise, returns a non 0 value.

9.10 setConfigInfo

This function sets device configuration information. For an example, see appendix **15.23B.6 IDeviceConfiguration Walk Through**.

```
int IDeviceConfiguration.setConfigInfo(
      byte configType,
      byte[] data,
      IConfigurationCallback callback);
```

Parameter	Description
configType	Type of configuration. For DynaFlex, this is the first number of the Property OID.
data	Configuration data to be sent to the device. For DynaFlex, this is the remainder of the constructed OID and value. For constructing the OID see ***D998200383 DynaFlex Family Programmer's Manual (COMMANDS)***
callback	Name of a class or structure that implements the **IConfigurationCallback** Delegates events.

Return Value:
Returns 0 if the asynchronous configuration operation started. Otherwise, returns a non 0 value.

9.11 setDisplayImage

This function sets which image is to be displayed when the device is idle. The image is displayed after a device reset.

```
int IDeviceConfiguration.setDisplayImage(byte imageID);
```

Parameter	Description
imageID	Value for the image ID. For DynaFlex, use: `0, 1, 2, 3, or 4` `0 =` restore the idle image to the "Welcome" screen.

Return Value:
Returns 0 if the asynchronous configuration operation started. Otherwise, returns a non 0 value.

9.12 updateFirmware

This function updates the device firmware.

```
int IDeviceConfiguration.updateFirmware(
```

```
      ushort firmwareType,
      byte[] data,
      IConfigurationCallback callback);
```

Parameter	Description
firmwareType	Type of firmware. For DynaFlex, use: 1 – Main App 2 – Wireless App
data	Firmware image to be sent to the device.
callback	Name of a class or structure that implements the **IConfigurationCallback** Delegates events.

Return Value:
Returns 0 if the asynchronous update operation started. Otherwise, returns a non 0 value.

9.13 updateKeyInfo

This function updates key information in the device.

```
int IDeviceConfiguration.updateKeyInfo(
      byte keyType,
      byte[] data,
      IConfigurationCallback callback);
```

Parameter	Description
keyType	Type of key.
data	Key data to be sent to the device.
callback	Name of a class or structure that implements the **IConfigurationCallback** Delegates events.

Return Value:
Returns 0 if the asynchronous update operation started. Otherwise, returns a non 0 value.

10 Classes

These classes are equipped with helper classes named Builders. Builders can parse the raw data byte array of an OnEvent() into a format required by a builder's class.

10.1 BarCodeData

These constructors initialize a BarCodeData object. Use BarCodeDataBuilder.GetBarCodeData with data from the BarCodeData event to return a BarcodeData.

```
BarCodeData BarCodeDataBuilder.GetBarCodeData(
      DeviceType deviceType,
      byte[] dataBytes
      );

new BarCodeData(
      byte[] Data,
      boolean Encrypted,
      byte EncryptionType,
      byte[] KSN
      );
```

Member	Description
Data	Returns the data payload.
Encrypted	Returns the encryption status. false = data is not encrypted true = data is encrypted
EncryptionType	Returns the encryption type.
KSN	Returns the Key Serial Number.

Return Value:
Returns an instance of BarCodeData.

10.2 CertificateInfo

CertificateInfo is used for the connection to a device requiring client credentials. See the following for details on installing a certificate chain: ***D998200550 DynaFlex II PED Using Wireless LAN Guide***.

CertificateInfo		
Member	Type/ Format	Description
format getFormat()	string	Certificate data format. "PKCS12" – for .p12 file. "PFX" – for .pfx file.
data getData()	byte[]	Certificate data

CertificateInfo		
password getPassword()	string	Password to access the certificate data.

Example of using CertificateInfo.

```
string format = "PKCS12";
byte[] data = mSessionManager.getDataFromURI(mCertificateURI);
string password = "password";

CertificateInfo certificateInfo = new CertificateInfo(
      format,
      data,
      password);

IDevice device = CoreAPI.createDevice(
      context,
      DeviceType.MMS,
      ConnectionType.WEBSOCKET,
      "",
      "DynaFlex",
      "",
      "",
      certificateInfo));
```

10.3 ConnectionStateBuilder

This class returns the connection state of the device when supplied the data object of the OnEvent() ConnectionState event.

```
string ConnectionStateBuilder.GetString(ConnectionState value);

ConnectionState ConnectionStateBuilder.GetValue(string data);
```

Member	Description
CONNECTED	Returns string of connected.
CONNECTING	Returns string of connecting.
DISCONNECTED	Returns string of disconnected.
DISCONNECTING	Returns string of disconnecting.
ERROR	Returns string of error.
GetString()	Returns string of the ConnectionState enum.
GetValue()	Returns ConnectionState enum from the event data string value.

Return Value:
Returns an instance of ConnectionStateBuilder.

10.4 DirectoryEntry

DirectoryEntry is similar to an InputRequest for Application during a transaction. Other fields of information are included besides Application Label. During a transaction, the device selects the PPSE. The directory entries are the PPSE response starting from the BF0C tag. The number of directory entries correspond to the number of applications on the card.

This takes the data from the EnhancedInputRequest event and returns an enhanced application selection list. This class extends InputRequest.

Code paths to receiving DirectoryEntry:

```
EnhancedInputRequest Event > EnhancedInputRequest Object >

Object.EnhancedSelectionList() > List<DirectoryEntry>
```

These constructors initialize a DirectoryEntry object.

```
new DirectoryEntry();
new DirectoryEntry(
      String Aid,
      String Label,
      byte Priority,
      byte[] ProprietaryData,
      byte KernelIdentifier,
      byte[] IssuerIN,
      byte[] IssuerINE,
      byte[] IssuerCountryCodeAlpha2,
      byte[] IssuerCountryCodeAlpha3,
      byte[] CardProductDetails
      );
```

Parameter	Type	Description
Aid	String	Application Identifier. Tag 4F Max 16
Label	String	Application Label. Tag 50 Max 16
Priority	byte	Application Priority Indicator. Tag 87
ProprietaryData	byte[]	Application Selection Registered Proprietary Data. Tag 9F0A
KernelIdentifier	byte	Kernel Identifier. Tag 9F2A
IssuerIN	byte[3]	Issuer Identification Number. Tag 42
IssuerINE	byte[4]	Issuer Identification Number Extended. Tag 9F0C
IssuerCountryCodeAlpha2	byte[2]	Issuer Country Code (alpha2 format). Tag 5F55
IssuerCountryCodeAlpha3	byte[3]	Issuer Country Code (alpha3 format). Tag 5F56
CardProductDetails	byte[2]	Card Product Details. Tag 9F7D

Return Value:
Returns an instance of DirectoryEntry.

10.5 EnhancedInputRequest

This supplies a directory entry list for application selection after a card has been presented to the device during a transaction. To be implemented, Application Selection Behavior property 1.2.1.1.1.2 must be set to 0x03 – Enhanced Prompt Cardholder.

The data byte array from the EnhancedInputRequest event is parsed and returned as an enhanced application selection list (directory entry). This list is the PPSE response starting from tag BF0C. EnhancedInputRequest class extends InputRequest.

```
new EnhancedInputRequest();
new EnhancedInputRequest(byte[] data);
```

EnhancedInputRequest		
Member	**Return**	**Description**
EnhancedSelectionList()	List<DirectoryEntry>	Returns a list of enhanced application selection.
setEnhancedSelectionList()	List<DirectoryEntry>	Sets a list of enhanced application selection.
Title()	String	Title to display.
Type()	int	Selection type.
Timeout()	long	Timeout in seconds to make selection.

10.6 InputRequest

This is used for displaying messages prompted by the card during a transaction. The data byte array from the InputRequest event is parsed and returned as an InputRequest.

```
new InputRequest();
new InputRequest(byte[] data);
```

InputRequest		
Member	**Type/ Format**	**Description**
Type	byte	Input type. 0x00 = INPUT_TYPE_APPLICATION 0x01 = INPUT_TYPE_LANGUAGE
Timeout	byte	Timeout in seconds
Title	string	Title
SelectionList	List<string>	List of selections for Application and Language depending on the Type.

Static		
Member	**Return**	**Description**
INPUT_TYPE_APPLICATION	byte	Selection type is for Application.

Static		
INPUT_TYPE_LANGUAGE	byte	Selection type is for Language.
INPUT_STATUS_COMPLETED	byte	To send the status of completed for sendSelection().
INPUT_STATUS_CANCELLED	byte	To send the status of canceled for sendSelection().
INPUT_STATUS_TIMED_OUT	byte	To send the status of timed out for sendSelection().

Return:
Returns and instance of InputRequest.

10.7 IData

IData is used for the payload of events and passing data to functions. When assigning the member StringValue, the member ByteArray is automatically assigned. Same is true vice versa. In this way either a string or an array can be accessed without need of data conversion.

Use the BaseData() function to assign an instance of IData.

IData		
Member	Type/ Format	Description
StringValue	string	String value
ByteArray	byte[]	Byte array

Example of using IData.

```
// String usage
IData data1 = new BaseData("3030");

IData data2 = new BaseData("");
data2.StringValue = "3030";

// Array usage
IData data3 = new BaseData(new byte[] {0x30, 0x30});

IData data4 = new BaseData("");
data4.ByteArray = new byte[] {0x30, 0x30};
```

10.8 ITransaction

This is the interface used as the parameter for **startTransaction**(). For an example, see the sample code in **IDevice Walk Through**.

ITransaction		
Member	**Type/ Format**	**Description**
PaymentMethods	List of PaymentMethod	List of the PaymentMethod enumeration. `MSR` = For magnetic stripe cards. `Contact` = For EMV chip cards. `Contactless` = For NFC contactless cards. `ManualEntry` = Manually entry, no card. When set, other payment methods must not be included. `Barcode` = For barcode. `BarcodeEncrypted` = For barcode with encrypted response. `AppleVAS` = For Apple VAS. `GoogleVAS` = For Apple VAS. `NFC` = For NFC tag.
QuickChip	bool	In QuickChip mode, the device does not prompt for an amount. Device sends an ARQC request to the host. Device automatically populates the ARPC response data with EMV Tag 8A set to "Z3". Card holder is prompted to remove the card. Transaction result is later determined by the processor and not by the card. `false` - Do not enable QuickChip mode. `true` - Enable QuickChip mode. Default.
Timeout	byte	Transaction timeout in seconds. Default is 60 seconds. `0 to 255` - Depending on the device, 0 means no timeout.
Amount	string Max 13	EMV Tag 9F02 - Authorized amount of the transaction. Example: "`1.23`" = $1.23 "`10`" = $10.00
CashBack	string Max 13	EMV Tag 9F03 - Secondary amount associated with the transaction. Example: "`1.23`" = $1.23 "`10`" = $10.00

ITransaction		
CurrenyCode	byte[] 2	EMV Tag 5F2A - Currency code of the transaction according to ISO 4217. The byte array is null by default. Example: `0x0840` = US Dollar `0x0978` = Euro `0x0826` = UK Pound
CurrencyExponent	byte[] 1	EMV Tag 5F36 - The decimal point position from the right of the transaction amount. The byte array is null by default. Example: `0x02` = decimal point at 2 positions from the right.
EMVOnly	bool	Flag that determines whether or not to start a transaction in EMV mode. This takes effect for devices which support both MSR and EMV mode (not limited to eDynamo, tDynamo, and DynaPro Family). This has no effect on non EMV devices. `false` = Do not start transaction in EMV mode. `true` = Only start transaction in EMV mode. Default.
PreventMSRSignatureForCardWithICC	bool	Flag that forces the device to skip signature capture during an MSR-only transaction if the card's service code indicates it is a chip card. `false` = Allow the prompt for a signature if requested. `true` = Do not prompt for signature.
SuppressThankYouMessage	bool	By default, devices with a display signal the end of a transaction by briefly showing "THANK YOU," then "WELCOME." `false` = Do not suppress the thank you message. `true` = Suppress the thank you message.
DisplayAmountForQuickChip	bool	Display Amount for Quick Chip Transaction Flow. `false` = Do not display Amount when QuickChip mode is true. Default. `true` = Display Amount when QuickChip mode is true.

ITransaction		
OverrideFinalTransactionMessage	byte	By default, devices with a display signal the end of a transaction by returning to the idle page and showing "WELCOME." This parameter directs the device to show a message based on the Message ID from the command **displayMessage()**. This option completely overrides the device's idle page behavior until the next transaction, power cycle, or other similar state change. Examples: 0x00 - reserved, do not use. 0x01 - "AMOUNT" 0x02 - "AMOUNT OK?" 0x03 - "APPROVED" 0x04 - "CALL YOUR BANK" 0x05 - "CANCEL OR ENTER" 0x06 - "CARD ERROR" 0x07 - "DECLINED"
EMVResponseFormat	byte	The format of the EMV response. 0x00 = Legacy. Default. 0x01 = RFU
MerchantCategory	byte[] 2	EMV Tag 9F15 - The type of business being done by the merchant, represented according to ISO 18245. The byte array is null by default.
MerchantCustomData	byte[] 20	EMV Tag 9F7C – Proprietary merchant data that may be requested. The byte array is null by default.
MerchantID	byte[] 15	EMV Tag 9F16 - Used to uniquely identify a given merchant. The byte array is null by default.
TransactionCategory	byte[] 1	EMV Tag 9F53 - The type of contactless transaction being performed. The byte array is null by default.
TransactionType	byte 1	EMV Tag 9C - The type of financial transaction, represented by the first two digits of the ISO 8583:1987 Processing Code. 0x00 = purchase. Default. 0x01 = cash advance 0x09 = purchase with cashback 0x20 = refund Supported transaction types can found in the commands programmers manual specific to the device.
AppleVASMode	VASMode	An enumeration for the Apple VAS Mode.
AppleVASProtocol	VASProtocol	An enumeration for the Apple VAS Protocol.

ITransaction		
ECP2FrameData	byte[] 3-18	Apple ECP2.0 frame from Byte 2 to Byte N. See Apple ECP2.0 specification. • For example, if ECP2.0 VASUP-A frame is 6A02C3020003FFFF then Byte 2-N is 0xC3020003FFFF. • By default, ECP2.0 frame (Byte 2 -N) is 0xC3020003FFFF
CustomNFCTransactionMode	List of NFCTransactionMode	List of NFC transaction modes.
CustomNFCReadDataMode	NFCDataMode	Custom NFC data mode.
ManualEntryType	byte	User interface sequence. 0x00 = Card Number, Expiration Date, Security Code 0x01 = Name on Card, Card Number, Expiration Date, Security Code (Reserved for Future Use) 0x02 = Qwick Code, Last 4 digits of Card Number, Security Code (Reserved for future use)
ManualEntryFormat	byte	Card number valid format. 0x00 = PAN min 8, max 21 digits
ManualEntrySound	byte	Beeper feedback. 0x00 = On keypress sound disabled 0x01 = On keypress sound enabled
TipMode	byte	Tip mode. 0x00 = Disable Tip Mode 0x01 = Show Tip GUI immediately using % 0x02 = Show Tip GUI immediately using $ 0x11 = Enable Read Channel(s), with +Tip Button using % 0x12 = Enable Read Channel(s), with +Tip Button, using $
Tip1DisplayMode	byte	Display mode for Tip button 1. 0x00 = % or $ 0x01 = Display Custom 0x02 = Display NO TIP 0x03 = Disabled
Tip2DisplayMode	byte	Display mode for Tip button 2. 0x00 = % or $ 0x01 = Display Custom 0x02 = Display NO TIP 0x03 = Disabled

ITransaction		
Tip3DisplayMode	byte	Display mode for Tip button 3. 0x00 = % or $ 0x01 = Display Custom 0x02 = Display NO TIP 0x03 = Disabled
Tip4DisplayMode	byte	Display mode for Tip button 4. 0x00 = % or $ 0x01 = Display Custom 0x02 = Display NO TIP 0x03 = Disabled
Tip5DisplayMode	byte	Display mode for Tip button 5. 0x00 = % or $ 0x01 = Display Custom 0x02 = Display NO TIP 0x03 = Disabled
Tip6DisplayMode	byte	Display mode for Tip button 6. 0x00 = % or $ 0x01 = Display Custom 0x02 = Display NO TIP 0x03 = Disabled
Tip1Value	string Max 7	Display value for Tip button 1. "1.00" = $1.00 (Display Mode is $) "10.0001" = 10.0001% (Display Mode is %)
Tip2Value	string Max 7	Display value for Tip button 2. "1.00" = $1.00 (Display Mode is $) "10.0001" = 10.0001% (Display Mode is %)
Tip3Value	string Max 7	Display value for Tip button 3. "1.00" = $1.00 (Display Mode is $) "10.0001" = 10.0001% (Display Mode is %)
Tip4Value	string Max 7	Display value for Tip button 4. "1.00" = $1.00 (Display Mode is $) "10.0001" = 10.0001% (Display Mode is %)
Tip5Value	string Max 7	Display value for Tip button 5. "1.00" = $1.00 (Display Mode is $) "10.0001" = 10.0001% (Display Mode is %)
Tip6Value	string Max 7	Display value for Tip button 6. "1.00" = $1.00 (Display Mode is $) "10.0001" = 10.0001% (Display Mode is %)
TaxAmount	string Max 7	Tax amount. "1.00" = $1.00

ITransaction		
FunctionalButtonRight Option	byte[] 2	String ID to use for the Right functional button. The byte array is null by default. When user presses this button, device sends a notification to the host to indicate the present card functional button Right is pressed. Device then waits for the next command from the host. While waiting, the screen shows "PLEASE WAIT". `NULL` = Disable. `0x0000 to 0x00FF` = String ID

10.9 NFCData

These constructors initialize an NFCData object. Use NFCDataBuilder.GetNFCData with data byte array from the NFCData event to return an NFCData.

```
NFCData NFCDataBuilder.GetNFCData(
DeviceType deviceType,
byte[] dataBytes);

new NFCData(
      byte[] Data,
      boolean Encrypted,
      byte EncryptionType,
      byte[] KSN
      );
```

Member	Description
Data()	Returns the data payload.
Encrypted()	Returns the encryption status. false = data is not encrypted true = data is encrypted
EncryptionType()	Returns the encryption type.
KSN()	Returns the Key Serial Number.

Return Value:
Returns an instance of NFCData.

10.10 NFCRAPDUData

These constructors initialize an NFCRAPDUData object. Use NFCDataBuilder.GetNFCRAPDUData with data byte array from the NFCRAPDUResponse event to return an NFCRAPDUData.

```
NFCRAPDUData NFCRAPDU(
      byte[] response,
      byte[] data,
      boolean encrypted,
      byte encryptionType,
      byte[] ksn
```

```
      );

NFCRAPDUData NFCDataBuilder.GetNFCRAPDUData(
      DeviceType deviceType,
      byte[] dataBytes,
      );
```

Parameter	Description
response	Response data.
data	The data payload.
encrypted	Encryption status. false = data is not encrypted true = data is encrypted
encryptionType	The encryption type.
ksn	The Key Serial Number.
deviceType	Device type.
dataBytes	Data bytes from the NFCAPDUResponse event.

Return Value:
Returns an instance of NFCRAPDUData.

10.11 NFCEventBuilder

This class assist in parsing NFCEvent data.

Static		
Member	Return	Description
GetDetail(string data)	string	Returns a string containing NFC detail.
GetEventValue(string data)	NFCEvent	Returns the NFCEvent enumeration.
GetString(NFCEvent value)	string	Returns a string representation of NFCEvent enumeration.

Static		
Member	Value	Description
NFC_MIFARE_ULTRALIGHT	String	"nfc_mifare_ultralight"
MIFARE_CLASSIC_1K	String	"mifare_classic_1k"
MIFARE_CLASSIC_4K	String	"mifare_classic_4k"
MIFARE_DESFIRE	String	“mifare_desfire”
TAG_REMOVED	String	"tag_removed"
FAILED	String	"failed"

Static		
IO_FAILED	String	"io_failed"
AUTHENTICATION_FAILED	String	"authentication_failed"

10.12 PersonalInfoEntry

These constructors initialize a PersonalInfoEntry object.

- Call startPersonalInfoEntry()
- At OnEvent():TouchscreenPersonalInfoEntry build the object.

```
PersonalInfoEntry PersonalInfoEntry.GetPersonalInfoEntry(
DeviceType deviceType,
byte[] dataBytes);

new PersonalInfoEntry(
      byte[] Data,
      CaptureType DataType,
      bool Encrypted,
      byte EncryptionType,
      byte[] KSN
      );
```

Member	Description
deviceType	Type of device.
dataBytes	Data to pass in from OnEvent().
Data()	Returns the data payload.
DataType()	Returns the capture type.
Encrypted()	Returns the encryption status. false = data is not encrypted true = data is encrypted
EncryptionType()	Returns the encryption type.
KSN()	Returns the Key Serial Number.

Return Value:
Returns an instance of PersonalInfoEntry.

10.13 PersonalInfoEntryBuilder

This class assist in parsing PersonalInfoEntry data.

- Call startPersonalInfoEntry()
- At OnEvent():TouchscreenPersonalInfoEntry build the object.

```
PersonalInfoEntry GetPersonalInfoEntry(
DeviceType deviceType,
byte[] dataBytes);
```

Member	Description
deviceType	Type of device.
dataBytes	Data to pass in from OnEvent().

Return Value:
Returns an instance of PersonalInfoEntry.

11 IEventSubscriber Delegates

This interface invokes a callback function to receive data and/or a detailed response. To register for the event(s), call the **subscribeAll()** function with the name of a class that implements the **IEventSubscriber** Delegate interface.

11.1 OnEvent

This is the handler for multiple event types.

```
public void OnEvent(
      EventType eventType,
      IData data);
```

Parameter	Description
eventType	An enumeration for EventType. This indicates the event triggered by the device.
data	Contains the data for the event. The payload is dependent on the event type.

Return Value: None

Event implementation

The OnEvent method must be included and parameterized correctly as shown below to conform to the interface.

In this example, a separate class implements IEventSubscriber.

```
class OnEventClass : IEventSubscriber
{
    public void OnEvent(EventType eventType, IData data)
    {
        // Event handler
    }
}

OnEventClass eventCallBack = new OnEventClass();
device.subscribeAll(eventCallBack);
```

In this example, the main window implements IEventSubscriber. The keyword “this” is used to pass in the name of the current class MainWindow.

```
public partial class MainWindow : Window, IEventSubscriber
{
    public void OnEvent(EventType eventType, IData data)
    {
        // Event handler
    }

    private void connectDevice()
    {
        device.subscribeAll(this);
    }
}
```

Event Data Parsing
Classes can be initialized by passing in the data byte array.

Example:

```
public void OnEvent(EventType eventType, IData data)
    {
     // Barcode data
     BarCodeData barcodeData =
BarCodeDataBuilder.GetBarCodeData(DeviceType.MMS, data.ByteArray);

     // Input request
     InputRequest ir = new InputRequest(data.ByteArray);

     // NFC data
     NFCData nfcData = NFCDataBuilder.GetNFCData(DeviceType.MMS,
data.ByteArray);

     // Enhanced input request
     EnhancedInputRequest eir = new
EnhancedInputRequest(data.ByteArray);
     List<DirectoryEntry> deList = eir.EnhancedSelectionList;

    }
```

12 IConfigurationCallback Delegates

This interface invokes callback functions to receive data and/or a detailed response. To register for the event(s), call the **subscribeAll()** function with the name of a class that implements the **IConfigurationCallback** Delegates interface.

12.1 OnCalculateMAC

This event is called when certain asynchronous **IDeviceConfiguration** operations need to have a MAC included with the request.

```
IResult OnCalculateMAC(
      byte macType,
      byte[] data);
```

Parameter	Description
macType	Type of Mac algorithm. For DynaFlex, use 0.
data	Contains the data of the payload to MAC.

Return Value:
Returns an IResult that contains the calculated MAC.

12.2 OnProgress

This event is called to update the host on the progress of an asynchronous IDeviceConfiguration operation.

```
public void OnProgres(int progress);
```

Parameter	Description
progress	The progress of the configuration operation. Range: 0 - 100

Return Value: None

12.3 OnResult

This event is called to update the host when an asynchronous IDeviceConfiguration operation is completed.

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

Parameter	Description
status	An enumerated **Library Status Codes**.
data	Contains the data for the event.

Return Value: None

C# Example:

```
class OnConfigCallbackClass : IConfigurationCallback
{
    public IResult OnCalculateMAC(byte macType, byte[] data)
    {
        // Event handler
    }

    public void OnProgress(int progress)
    {
        // Event handler
    }

    public void OnResult(StatusCode status, byte[] data)
    {
        // Event handler
    }

}

OnConfigCallbackClass configCallBack = new OnConfigCallbackClass();
```

13 IMQTTDeviceStatusCallback Delegates

This interface invokes callback functions to receive data and/or a detailed response. To register for the event(s), call the setMQTTDeviceStatusMonitoring() function with the name of a class that implements the IMQTTDeviceStatusCallback interface.

13.1 OnConnected

This event is called when a device is connected to the MQTT broker.

```
void OnConnected(string deviceAddress);
```

Parameter	Description
deviceAddress	Device address.

Return Value: None

13.2 OnDisconnected

This event is called when a device is disconnected from the MQTT broker.

```
void OnDisconnected(string deviceAddress);
```

Parameter	Description
deviceAddress	Device address.

Return Value: None

14 ISystemStatusCallback Delegates

This interface invokes callback functions to receive data and/or a detailed response. To register for the event(s), call the setSystemStatusCallback() function with the name of a class that implements the ISystemStatusCallback interface.

14.1 OnError

This event is called when an error occurs with the connection with the MQTT broker.

```
void OnError(
      ErrorType error,
      string details);
```

Parameter	Description
error	Type of error enumeration.
details	Details of the error.

Return Value: None

15 Enumerations

15.1 BarCodeFormat

This enum refers to the type of barcodes to display.

Enum	Description
BLOB	Data is binary format
COMMAND	Data is a command in binary format
BLOB_BASE64	Data is Base64 encoded format
COMMAND_BASE64	Data is a command in Base64 format

15.2 BarCodeType

This enum refers to the type of barcodes to display.

Enum	Description
QRCODE	QR code

15.3 CaptureType

This enum refers to the type of customer information to capture.

Enum	Description
Unavailable	Capture unavailable
PhoneNumber	Capture the phone number
SocialSecurityNumber	Capture the social security number
ZipCode	Capture the zip code
EmployeeID	Capture Employee ID number
BirthDate	Capture birth date in USA format

15.4 ConnectionState

This enum refers to the readiness of the SDK to communicate with the device. This is not the physical attachment to a host system.

Enum	Description
Unknown	Device is in an unknown connection state.
Disconnected	Device is disconnected.
Connecting	Device is in the process of connecting. The next state is to be Connected.
Error	There was an error either connecting or disconnecting the device.
Connected	Device is connected and ready for transacting.

Enum	Description
`Disconnecting`	Device is in the process of disconnecting. The next state will be `Disconnected`.

15.5 ConnectionType

This enum refers to the communication interface type of MagTek reader which the SDK will control.

Enum	Description
`USB`	Universal Serial Bus supported devices: • eDynamo • mDynamo • Dynamag • DynaMax • tDynamo • kDynamo • cDynamo • iDynamo 6 • DynaPro • DynaPro Go • DynaPro Mini • DynaFlex • DynaFlex Pro • DynaFlex II PED • DynaProx • DynaFlex II Go
`BLUETOOTH_LE`	Bluetooth Low Energy devices: • DynaMax • DynaFlex II Go
`BLUETOOTH_LE_EMV`	Bluetooth Low Energy with EMV supported devices: • eDynamo
`BLUETOOTH_LE_EMVT`	Bluetooth Low Energy with EMV supported devices: • tDynamo
`TCP`	Transmission Control Protocol supported devices: • DynaPro
`TCP_TLS`	Transmission Control Protocol with Transport Layer Security supported devices: • DynaPro Go
`TCP_TLS_TRUST`	Transmission Control Protocol with Transport Layer Security supported devices: • DynaPro Go
`WEBSOCKET`	WebSocket supported devices: • DynaFlex Pro • DynaFlex II PED

Enum	Description
WEBSOCKET_TRUST	WebSocket supported devices. This will establish a TLS connection to device without requirement for name match. • DynaFlex Pro • DynaFlex II PED
MQTT	MQTT • DynaFlex II PED
SERIAL	UART supported devices
AUDIO	Audio devices: • iDynamo 6
VIRTUAL	Virtual devices

15.6 DataEntryType

This enum is reserved for future use.

Enum	Description
PIN	Request Personal Identification Number
Signature	Request Signature
SSN	Request Social security number
ZIPCODE	Request Zip code
BirthDate	Request Birth date
ActivationCode	Request Activation code

15.7 DeviceEvent

This enum refers to a change in the device status.

Enum	Description
None	No event to report.
DeviceResetOccured	A device reset had occurred.
DeviceResetWillOccur	A device reset will occur soon. Host application may uses this as a warning to take appropriate actions.
DeviceBondingFailure	Host fails to pair with device.
DeviceTemperatureLow	The device's temperature falls below the temperature set in Low Temperature Notification Level.
DeviceTemperatureHigh	The device's temperature rises above the temperature set in High Temperature Notification Level.
DeviceBatteryLow	The battery charge reaches 15 percent. If a device is powered on with a charge that is already 15 percent or below, this notification is sent shortly after power up.

Enum	Description
DeviceBatteryLowPowerDown	This notification is sent one minute before it automatically powers down the device. This occurs when the battery charge has reached 0 percent. Shutdown can be prevented by connecting the device to a USB power source.

15.8 DeviceFeature

This enum refers to a featured supported by the device.

Enum	Description
None	No feature
SignatureCapture	Supports signature capture
PINEntry	Supports PIN entry
PANEntry	Supports PAN entry
ShowBarCode	Supports display of a barcode
ScanBarCode	Supports scanning a barcode
DisplayMessage	Supports display of messages
Buzzer	Has a buzzer
CardEmulation	Supports NFC card emulation
PersonalInfoEntry	Supports personal info entry

15.9 DeviceType

This enum refers to the type of MagTek reader which the SDK will control.

Enum	Description
SCRA	Secure Reader Authenticator devices. List includes but not limited to: • eDynamo • mDynamo • Dynamag • DynaMax • tDynamo • kDynamo • cDynamo • iDynamo 6

Enum	Description
PPSCRA	PIN Pad Secure Reader Authenticator devices. List includes but not limited to: • DynaPro • DynaPro Go • DynaPro Mini
CMF	Common Message Structure devices. List includes but not limited to: • oDynamo
MMS	MMS class devices. (MagTek Message Scheme) List includes but not limited to: • DynaFlex • DynaFlex Pro • DynaProx

15.10 ErrorType

This enum refers to the type of connection error.

Enum	Description
Unknown	Device is in an unknown connection state.
TimedOut	Device has timed out.
BluetoothOff	Bluetooth is off.
BluetoothUnauthorized	Bluetooth not paired.
NetworkOff	Network is off.
NetworkUnreachable	Network is unreachable.
SecurityRejected	Security rejected.
SecurityFailed	Security failed.
ConnectionFailed	Connection failed.

15.11 EventType

This enum refers to the type of event triggered by the device.

Enum	Description
ConnectionState	There was a change in the connection state of the device.
DeviceResponse	Device has responded to a command.
DeviceExtendedResponse	Device has responded to an extended command.
DeviceNotification	Device has sent a notification.

Enum	Description
DeviceTransferCancelled	Device has cancelled data transfer.
CardData	Device has sent magnetic stripe data from a card swipe.
TransactionStatus	There was a change in transaction status.
DisplayMessage	Device has a message to display for the user.
ClearDisplay	Device has sent a notification to clear the display of messages for the user.
InputRequest	Device is requesting input from the user.
EnhancedInputRequest	Device is requesting enhanced input from the user.
AuthorizationRequest	Device has sent the Authorization Request Cryptogram and associated block of EMV tags for a transaction. This block is meant to be sent to the transaction processor.
TransactionResult	Device has sent the result of the transaction.
PINBlock	Device has sent the PINBlock after the user has entered a PIN on the device.
Signature	Device has sent data which represents a signature from a user.
DeviceDataFile	Device has sent a data file.
OperationStatus	Device has sent an operation status of a command.
DeviceEvent	Device has sent change of device state.
UserEvent	Device has sent a notification related to user interaction with the device.
PINData	Device has sent data related to a PIN.
PANData	Device has sent data related to a PAN.
FeatureStatus	Device has sent status of a feature.
BarCodeData	Device has sent barcode data.
NFCEvent	Device has sent NFC event.
NFCData	Device has sent NFC data.
NFCResponse	Device has sent response to an NFC command.
NFCAPDUResponse	Device has sent response to an NFC APDU command for Mifare DESFire Tag.
NFCCardData	Device has sent NFC card data.
NFCDeviceEngagementData	Device has sent data after NFC engagement.

Enum	Description
NFCPassThroughData	Device has sent NFC Pass-Through data.
NFCPassThroughResponse	Device has sent a response to NFC Pass-Through exchange.
GPOResponse	Device has GPO response.
TouchscreenSignatureCapture	Device has sent response to signature capture.
TouchscreenFunctionalButtonSelected	Device has sent a functional button selected.
TouchscreenTextStringButtonSelected	Device has sent a text string button selected.
TouchscreenAmountButtonSelected	Device has sent an amount button selected.
TouchscreenPresentCardFunctionalButtonSelected	Device has sent a present card functional button selected.
TouchscreenPersonalInfoEntry	Device has sent a personal info entry.

15.12 FeatureStatus

This enum refers to the status of a specific feature reported from **DeviceFeature**.

Enum	Description
NoStatus	No change in status
Success	Success
Failed	Failed
TimedOut	Timed out
Cancelled	Cancelled
Error	Error
HardwareNA	Featured hardware not applicable for a status

15.13 ImageType

This enum refers to the type of image.

Enum	Description
BITMAP	BMP file

15.14 InfoType

This enum refers to the type of specific information to retrieve from the device.

Enum	Description
DeviceSerialNumber	Device serial number.
FirmwareVersion	Firmware version of the device.
DeviceCapabilities	Capabilities of the device delimited by a comma.

Enum	Description
Boot1Version	Boot 1 firmware version of the device.
Boot0Version	Boot 0 firmware version of the device.
FirmwareHash	Firmware hash comprised of part numbers, versions, and timestamps.
TamperStatus	Tamper status of the device. 0x00 = Not Tampered 0x01 = Tampered
OperationStatus	Operation status of the device. 0x01 = Offline 0x02 = Online
OfflineDetail	Details of why the device is offline. • Bit 0 = Tamper problem present • Bit 1 = Master Key problem present • Bit 2 = Keys and Certificates problem present • Bit 3 = Real Time Clock problem present • Bit 4 = Random Number Generator problem present • Bit 5 = Cryptography Engine problem present • Bit 6 = Magnetic Stripe Reader Hardware problem present • Bit 7 = Reserved
FirmwareVersionWLAN	Firmware version for the WLAN module of the device.
FirmwareVersionBLE	Firmware version for the Bluetooth LE module of the device.
BatteryLevel	Battery level of the device.

15.15 NFCDataMode

This enum refers to NFC data modes.

Enum	Description
ASCII	ASCII data mode
Binary	Binary data mode

15.16 NFCEvent

This enum refers to NFC events.

Enum	Description
None	No event
NFCMifareUltralight	Mifare Ultralight
MifareClassic1K	Mifare Classic 1K
MifareClassic4K	Mifare Classic 4K
MifareDESFireLight	Mifare DESFire Light
MifareMini	Mifare Mini

Enum	Description
MifarePlusEV1	Mifare Plus EV1
MifarePlusEV2	Mifare Plus EV2
MifarePlusSE	Mifare Plus SE
MifarePlusX	Mifare Plus X
MifareDESFireEV1	Mifare DESFire EV1
MifareDESFireEV2	Mifare DESFire EV2
MifareDESFireEV3	Mifare DESFire EV3
ISO14443TypeA	ISO 14443 Type A
ISO14443TypeB	ISO 14443 Type B
NFCCardTypeNotSupported	NFC Card Type Not Supported
mDL	Mobile Driver license
mDLDeviceEngagementSuccess	mDL Device Engagement Success
mDLDeviceEngagementFailed	mDL Device Engagement Failed
TagRemoved	Tag removed
Failed	Command failed
IOFailed	IO failed
AuthenticationFailed	Authentication failed

15.17 NFCTransactionMode

This enum refers to NFC transaction modes.

Enum	Description
MifareClassic	Mifare Classic
MifareDESFire	Mifare DESFire
AppleWalletMobileDESFire	Apple Wallet mobile DESFire
Mifare2GoMobileDESFire	Mifare 2 Go Mobile DESFire

15.18 OperationStatus

This enum refers to the operation status of the device.

Enum	Description
NoStatus	No update for the operation.
Started	Device has started an operation.
Warning	Device has sent a warning about the operation.

Enum	Description
Failed	Device has failed an operation.
Done	Device has completed an operation.

15.19 PaymentMethod

This enum refers to which card type the device will perform a transaction.

Enum	Description
MSR	For magnetic stripe cards.
Contact	For EMV chip cards.
Contactless	For NFC contactless cards.
ManualEntry	For user to manually enter transaction data without any card access.
Barcode	For barcode.
BarcodeEncrypted	For encrypted barcode response.
AppleVAS	For Apple VAS.
NFC	For NFC tag.
GoogleVAS	For Google Wallet Smart Tap VAS.

15.20 TransactionStatus

This enum refers to the status of the transaction.

Enum	Description
NoStatus	Set before the start of a transaction and before a card is presented to the device.
NoTransaction	No transaction in progress.
CardSwiped	A card was swiped into the device.
CardInserted	A card was inserted into the device.
CardRemoved	A card was removed from the device.
CardDetected	A card was detected by the device.
CardCollision	A card collision was detected by the device.
TimedOut	The transaction was not completed before a timeout period.
HostCancelled	The host software sent a cancel.
TransactionCancelled	The transaction was cancelled.
TransactionInProgress	The transactions is in progress.
TransactionError	There is an error during the transaction.
TransactionApproved	The transactions is approved.

Enum	Description
TransactionDeclined	The transactions is declined.
TransactionCompleted	The transaction is completed.
TransactionFailed	The transaction failed.
TransactionNotAccepted	The transaction was not accepted by the device.
SignatureCaptureRequested	A signature capture is requested by the device.
TechnicalFallback	Due to technical reasons, the chip transaction cannot be completed by the reader.
QuickChipDeferred	Device has sent a “Z3” response code to the chip card.
DataEntered	Data has been entered on the device for a manual card entry transaction.
TryAnotherInterface	Due to removal of the chip card or error with contactless card, the transaction cannot be completed by the reader.
BarcodeRead	A barcode is read.
VASError	Apple VAS error occurred.

15.21 UserEvent

This enum refers to the type of user event reported by the device. These events relate to user interaction.

Enum	Description
None	No events yet to occur.
ContactlessCardPresented	Contactless card has been presented.
ContactlessCardRemoved	Contactless card has been removed.
CardSeated	Card is seated into the chip station.
CardUnseated	Card was removed from the chip station.
CardSwiped	Magnetic stripe card was swiped.
TouchPresented	Touch screen sensor press detected.
TouchRemoved	Touch screen sensor release detected.
BarcodeRead	Barcode detected.
NFCMifareUltralightPresented	Mifare Ultralight presented.
MifareClassic1KPresented	Mifare Classic 1K presented.
MifareClassic4KPresented	Mifare Classic 4K presented.
MifareDESFireLightPresented	Mifare DESFire Light presented.
MifareMiniPresented	Mifare Mini presented.
MifarePlusEV1Presented	Mifare Plus EV1 presented.
MifarePlusEV2Presented	Mifare Plus EV2 presented.

Enum	Description
MifarePlusSEPresented	Mifare Plus SE presented.
MifarePlusXPresented	Mifare Plus X presented.
MifareDESFireEV1Presented	Mifare DESFire EV1 presented.
MifareDESFireEV2Presented	Mifare DESFire EV2 presented.
MifareDESFireEV3Presented	Mifare DESFire EV3 presented.
mDLPresented	Mobile Driver license presented.
NFCMifareUltralightRemoved	Mifare Ultralight removed.
MifareClassic1KRemoved	Mifare Classic 1K removed.
MifareClassic4KRemoved	Mifare Classic 4K removed.
MifareDESFireLightRemoved	Mifare DESFire Light removed.
MifareMiniRemoved	Mifare Mini removed.
MifarePlusEV1Removed	Mifare Plus EV1 removed.
MifarePlusEV2Removed	Mifare Plus EV2 removed.
MifarePlusSERemoved	Mifare Plus SE removed.
MifarePlusXRemoved	Mifare Plus X removed.
MifareDESFireEV1Removed	Mifare DESFire EV1 removed.
MifareDESFireEV2Removed	Mifare DESFire EV2 removed.
MifareDESFireEV3Removed	Mifare DESFire EV3 removed.
mDLRemoved	Mobile Driver license removed.

15.22 VASMode

This enum refers to the Apple VAS and Google Wallet VAS mode. This controls how the VAS data is returned in the transaction ARQC. For details on Apple VAS data structure returned in a transaction see ***D998200383 DynaFlex Family Programmer's Manual (COMMANDS)***.

Enum	Description
Single	The device reads only VAS data from a tapped smartphone, or reads EMV payment data from a tapped card. When the device sends ARQC to conclude the transaction, it only includes either EMV payment data in container FC for cards, or includes VAS data in container FE for smartphones.
Dual	The device reads both VAS data and EMV payment data from a tapped smartphone, or reads EMV payment data from a tapped card. When device sends ARQC to the host to conclude the transaction, it includes EMV payment data in container FC and includes VAS data, if available, in container FE.

Enum	Description
VASOnly	The device reads only VAS data from a tapped smartphone, and does not read data from a tapped card. If the tapped smartphone does not support VAS, the device does not detect or read from the smartphone. When the device send ARQC to conclude the transaction, it includes VAS data in container FE and does not include EMV payment data in container FC.
PaymentOnly	The device operates the same as EMV mode. It reads only EMV payment data from a tapped smartphone or a tapped card. When the device sends ARQC to conclude the transaction, it includes EMV payment data in container FC and does not include VAS data in container FE.
ECP2	The host can configure the ECP2 frame. See Apple ECP2.0 specification.

15.23 VASProtocol

This enum refers to the Apple VAS protocol. For details on Apple VAS data structure returned in a transaction see ***D998200383 DynaFlex Family Programmer's Manual (COMMANDS)***.

Enum	Description
URL	URL VAS protocol
Full	Full VAS protocol

Appendix A Status Codes

A.1 Library Status Codes

```
public enum StatusCode
{
    SUCCESS = 0,
    TIMEOUT = 1,
    ERROR = 2,
    UNAVAILABLE = 3
}
```

Enum	Description
SUCCESS	The operation completed successfully.
TIMEOUT	The operation timed out.
ERROR	Error attempting the operation.
UNAVAILABLE	Status currently unavailable.

A.2 Supported Function List

The following table shows which MTUSDK API function is supported for a device.

Function / Product	DynaFlex	DynaFlex Pro	DynaFlex II	DynaFlex II PED	DynaProx	DynaFlex II Go
CoreAPI						
createCMSDevice						
createDevice	*	*	*	*	*	*
createMMSDevice	*	*	*	*	*	*
createPPSCRA						
createSCRA						
getDeviceList	*	*	*	*	*	*
getAPIVersion	*	*	*	*	*	*
setMQTTBrokerInfo				*		
setMQTTClientID				*		
setMQTTClientCertificateInfo				*		
setMQTTDeviceDiscoveryTimeout				*		
startMQTTDeviceStatusMonitory				*		
setMQTTPublishTopic				*		
setMQTTQos				*		
setMQTTSubscribeTopic				*		
stopMQTTDeviceStatusMonitoring				*		
setSystemStatusCallback				*		
IDevice						
cancelTransaction	*	*	*	*	*	*
getCapabilities	*	*	*	*	*	*
getConnectionInfo	*	*	*	*	*	*
getConnectionState	*	*	*	*	*	*
getDeviceConfiguration	*	*	*	*	*	*
getDeviceInfo	*	*	*	*	*	*

Function / Product	DynaFlex	DynaFlex Pro	DynaFlex II	DynaFlex II PED	DynaProx	DynaFlex II Go
Name	*	*	*	*	*	*
requestPAN				*	*	
requestPIN				*		
requestSignature		*		*		
sendAuthorization	*	*	*	*	*	*
sendClassicNFCCommand			*	*	*	*
sendNFCCommand			*	*	*	*
sendSelection	*	*	*	*	*	*
startTransaction	*	*	*	*	*	*
subscribeAll	*	*	*	*	*	*
unsubscribeAll	*	*	*	*	*	*
IDeviceCapabilities						
AutoSignatureCapture	*	*	*	*	*	*
BatteryBackedClock	*	*	*	*	*	*
Display	*	*	*	*	*	*
MSRPowerSaver	*	*	*	*	*	*
PaymentMethods	*	*	*	*	*	*
PINPad	*	*	*	*	*	*
Signature	*	*	*	*	*	*
SRED	*	*	*	*	*	*
IDeviceControl						
close	*	*	*	*	*	*
deviceReset	*	*	*	*	*	*
displayMessage		*		*		
endSession	*	*	*	*	*	*
getInput						
open	*	*	*	*	*	*
playSound						

Function / Product	DynaFlex	DynaFlex Pro	DynaFlex II	DynaFlex II PED	DynaProx	DynaFlex II Go
send	*	*	*	*	*	*
sendExtendedCommand						
sendSync	*	*	*	*	*	*
setDateTime						
setLatch						
showImage		*		*		
showBarCode		*		*		
startBarCodeReader	*	*	*	*	*	*
stopBarCodeReader	*	*	*	*	*	*
ConnectionInfo						
getAddress	*	*	*	*	*	*
getConnectionType	*	*	*	*	*	*
getDeviceType	*	*	*	*	*	*
getCertificationInfo	*	*	*	*	*	*
DeviceInfo						
getModel	*	*	*	*	*	*
getName	*	*	*	*	*	*
getSerial	*	*	*	*	*	*
IDeviceConfiguration						
getChallengeToken	*	*	*	*	*	*
getConfigInfo	*	*	*	*	*	*
getDeviceInfo	*	*	*	*	*	*
getFile	*	*	*	*	*	*
getKeyInfo	*	*	*	*	*	*
sendFile	*	*	*	*	*	*
sendImage		*		*		
sendSecureFile	*	*	*	*	*	*

Function / Product	DynaFlex	DynaFlex Pro	DynaFlex II	DynaFlex II PED	DynaProx	DynaFlex II Go
setConfigInfo	*	*	*	*	*	*
setDisplayImage		*		*		
updateFirmware	*	*	*	*	*	*
updateKeyInfo	*	*	*	*	*	*

Appendix B API Walk Through

B.1 CoreAPI Walk Trough

The following walks through how to create instances of a device.

- CoreAPI.createDevice → IDevice
- CoreAPI.getDeviceList → List<IDevice>
- CoreAPI.createPPSCRA → MTPPSCRA

These examples demonstrate methods for creating an IDevice to be used in the MagTek Universal SDK. This also shows how to establish a device specific API, which is not used with the MagTek Universal SDK.

Here, a single of IDevice is established.

```
// Access MMS with Universal SDK using createDevice()

IDevice mtmms = CoreAPI.createDevice(
      DeviceType.MMS,
      ConnectionType.USB,
      "",
      "",
      "DynaFlex",
      "");
mtmms.requestSignature(0xff);
```

Here, a list of IDevice is established. The first device is accessed at index 0.

```
// Acess MMS with Universal SDK using getDeviceList()

List<IDevice> mtmms = CoreAPI.getDeviceList(DeviceType.MMS);
mtmms[0].requestSignature(0xff);
```

Here the CoreAPI creates an instance of MTPPSCRA. The MTPPSCRANET is referenced and its library file is included in the application development project.

```
Using MTUSDKNET;
Using MTPPSCRANET;


// Directly access PPSCRA with MTPPSCRANET by createPPSCRA()

MTPPSCRANET.MTPPSCRA mtppscra = CoreAPI.createPPSCRA();
mtppscra.openDevice("");
mtppscra.requestSignature(30, 2, 0);
```

B.2 IDevice Walk Through

The following walks through how to make use of IDevice.

.

- Implement device events within the class to receive events.
- CoreAPI → IDevice.
- IDevice → subscribeAll().
- IDevice → other functions.
- IDevice → startTransaction().

Example

```
Using MTUSDKNET;
.

// Extend the main window to receive events.
public class MainWindow : IEventSubscriber, IConfigurationCallback
{

// Establish a device from CoreAPI.
List<IDevice> deviceList = CoreAPI.getDeviceList();
IDevice device = deviceList[0];

/* For a list of a single device type.
DeviceType deviceType = DeviceType.MMS;
List<IDevice> deviceList = CoreAPI.getDeviceList(deviceType);
IDevice device = deviceList[0];
*/

/* For a list of multiple device types.
List<DeviceType> deviceTypes = null;
deviceTypes.Add(DeviceType.MMS);
deviceTypes.Add(DeviceType.CMS);
List<IDevice> deviceList = CoreAPI.getDeviceList(deviceTypes);
IDevice device = deviceList[0];
*/

/* Suscribe to events sent from the device.
   These would be but not limited to: card inserted, card removed,
   connection state...

   Set MainWindow to receive the events. */
bool return = device.unsubscribeAll(this);
bool return = device.subscribeAll(this);

/* To handle events from some other class.
EventsVector eventsVector = new EventsVector()
bool return = device.unsubscribeAll(eventsVector);
bool return = device.subscribeAll(eventsVector);
*/
```

```
// Assign parameters for the transaction.
ITransaction transaction = new Transaction();
transaction.Amount = "1.00";
transaction.CashBack = "0.00";
transaction.EMVOnly = true;
transaction.PaymentMethods = new List<PaymentMethod>();
transaction.PaymentMethods.Add(PaymentMethod.MSR);
transaction.PaymentMethods.Add(PaymentMethod.Contact);
transaction.PaymentMethods.Add(PaymentMethod.Contactless);
transaction.QuickChip = false;

// Start transaction.
bool result = device.startTransaction(transaction);
```

B.2.1 Handling Events

Application Main window may extent the **IEventSubscriber** Delegate, or events can be interfaced by a separate class. This example uses a separate class and demonstrates how to parse for the various event types.

Example

```
// A class to handle events.
public class EventsVector : IEventSubscriber
{
  public void OnEvent(EventType eventType, IData data)
  {
    switch (eventType)
    {
```

Various events are separately shown below.

```
case EventType.ConnectionState:
// Parse for the ConnectionState
ConnectionState value =
ConnectionStateBuilder.GetValue(data.StringValue);

break;
```

```
case EventType.DeviceResponse:

break;
```

```
case EventType.DeviceExtendedResponse:

break;
```

```
case EventType.DeviceNotification:
```

```
break;
```

```
case EventType.CardData:

break;
```

```
case EventType.TransactionStatus:
// Parse for the transaction status code and detail.
TransactionStatus status =
TransactionStatusBuilder.GetStatusCode(data.StringValue);

string statusDetail =
TransactionStatusBuilder.GetStatusDetail(data.StringValue);

break;
```

```
case EventType.DisplayMessage:

string message;

// Get the message.
message = data.StringValue;

break;
```

```
case EventType.InputRequest:

break;
```

```
case EventType.AuthorizationRequest:

// Forward ARQC to processor.
/* data[0..1] - ARQC length
   data[2..n] - remainder contains the ARQC TLV object
*/

IData ARQC = new BaseData(data.ByteArray);

// App function to send the request to the processor.
string ARPC = sendARQCToProcessorForApproval(ARQC.ByteArray);

// Send authorization to device when not in QuickChip mode.
IData ARPCTLV = new BaseData("");
ARPCTLV.StringValue = "FF7413DFDF250742363243413546FA067004” + ARPC;

if (transaction.QuickChip == false)
{
  device.sendAuthorization(ARPCTLV);
}
```

```
break;
```

```
case EventType.TransactionResult:

/* data[0]    - Signature Required
   data[1..2] - Batch Data length
   data[3..n] - remainder contains the Batch Data TLV object
*/

// Parse the TLV from data[].
// Abstract Approval status from TLV tag "DFDF1A".
// Abstract Signature Required status from TLV tag at data[0].

break;
```

```
case EventType.PINData:

break;
```

```
case EventType.Signature:

break;
```

```
case EventType.TransactionStatus:

/*
Transaction status enumeration is build from
the TransactionStatusBuilder.

*/
TransactionStatus status =
TransactionStatusBuilder.GetStatusCode(data.StringValue);

if (status == TransactionStatus.CardSwiped)
{
  //
}
if (status == TransactionStatus.CardInserted)
{
  //
}

if (status == TransactionStatus.TransactionApproved)
{
  //
}

break;
```

B.3 IDeviceControl Walk Through

The following walks through how to make use of **IDeviceControl**.

- IDevice → IDeviceControl.
- IDeviceControl → open().
- IDeviceControl → other functions.
- IDeviceControl → close().

Example

```
// Establish a device from CoreAPI.
List<IDevice> deviceList = CoreAPI.getDeviceList();
IDevice device = deviceList[0];

// Establish a deviceControl from device.
IDeviceControl deviceControl = device.getDeviceControl();

// Open the device, then use the IDeviceControl functions.
deviceControl.open();

. . .

// Close the device.
deviceControl.close();
```

B.4 ConnectionInfo Walk Through

The following walks through how to make use of ConnectionInfo.

- IDevice → ConnectionInfo.
- ConnectionInfo → getAddress()
- ConnectionInfo → getConnectionType()
- ConnectionInfo → getDeviceType()

Example

```
// Establish a device from CoreAPI.
List<IDevice> deviceList = CoreAPI.getDeviceList();
IDevice device = deviceList[0];

// Establish a ConnectionInfo from device.
ConnectionInfo connectionInfo = device.getConnectionInfo();

// Retrieve address, connectionType, and deviceType.
string address = connectionInfo.getAddress();
ConnectionType connectionType = connectionInfo.getConnectionType();
DeviceType deviceType = connectionInfo.getDeviceType();
```

B.5 IDeviceCapabilities Walk Through

The following walks through how to make use of **IDeviceCapabilities**.

- IDevice → IDeviceCapabilities.
- IDeviceCapabilities → BatteryBackedClock() to check if date/time should be set.
- IDeviceCapabilities → PaymentMethods() to check card types supported.
- IDeviceCapabilities → other functions.

```
// Establish a device from CoreAPI.
List<IDevice> deviceList = CoreAPI.getDeviceList();
IDevice device = deviceList[0];


// Establish a IDeviceCapabilities from device.
IDeviceCapabilities capabilities = device.getCapabilities();


// Retrieve device capabilities.
bool batteryBackedClock = capabilities.BatteryBackedClock();
if (batteryBackedClock)
{
  // Call IDeviceControl.setDateTime().
}

// Retrieve supported card payment methods.
List<PaymentMethod> paymentMethods = capabilities.PaymentMethods();

. . .
```

B.6 IDeviceConfiguration Walk Through

The following walks through how to make use of **IDeviceConfiguration**.

- IDevice → getDeviceConfiguration().
- IDeviceConfiguration → updateFirmware().
- IDeviceConfiguration → getConfiguration().
- IDeviceConfiguration → setConfiguration().
- IDeviceConfiguration → other functions.

Example

```
// Establish a device from CoreAPI.
List<IDevice> deviceList = CoreAPI.getDeviceList();
IDevice device = deviceList[0];

IDeviceConfiguration devConfig = device.getDeviceConfiguration();

IDeviceControl devControl = device.getDeviceControl();
devControl.open();

/* To handle events from some other class.
ConfigCallBacks configCallBacks = new ConfigCallBacks();
*/

// Update firmware.
byte[] data = System.IO.File.ReadAllBytes("filepath");
int return = devConfig.updateFirmware(0x01, data, this);


/* Get configuration.
   Device-Driven Fallback OID = 1.2.1.1.1.1
               contructed OID = E2 08 E1 06 E1 04 E1 02 C1 00
   Note: first digit of OID is ommited in the construction and instead
is passed in the configType.
*/
byte configType = 0x01;
data = new byte[] {0xE2,0x08,0xE1,0x06,0xE1,0x04,0xE1,0x02,0xC1,0x00
};
byte[] response = devConfig.getConfigInfo(configType, data);


/* Set configuation.
   Device-Driven Fallback OID is 1.2.1.1.1.1
      Disabled contructed OID = E2 09 E1 07 E1 05 E1 03 C1 01 00
       Enabled contructed OID = E2 09 E1 07 E1 05 E1 03 C1 01 01
   Note: first digit of OID is ommited in the construction and instead
is passed in the configType.

*/
data = new byte[]
{0xE2,0x09,0xE1,0x07,0xE1,0x05,0xE1,0x03,0xC1,0x01,0x00 };
```

```
result = devConfig.getConfigInfo(configType, data);
```

B.6.1 Handling Events

Application Main window may extent the **IConfigurationCallback** Delegates or can be extended by a separate class. This example uses a separate class and demonstrates how to parse for the various events.

Example

```
// A class to handle configuration callback events.
public class ConfigCallbacks : MTUSDKNET.IConfigurationCallback
{
  public void OnProgress(int progress)
  {
      /* Handle progress.
         Progress is complete when progress = 100 */
  }
```

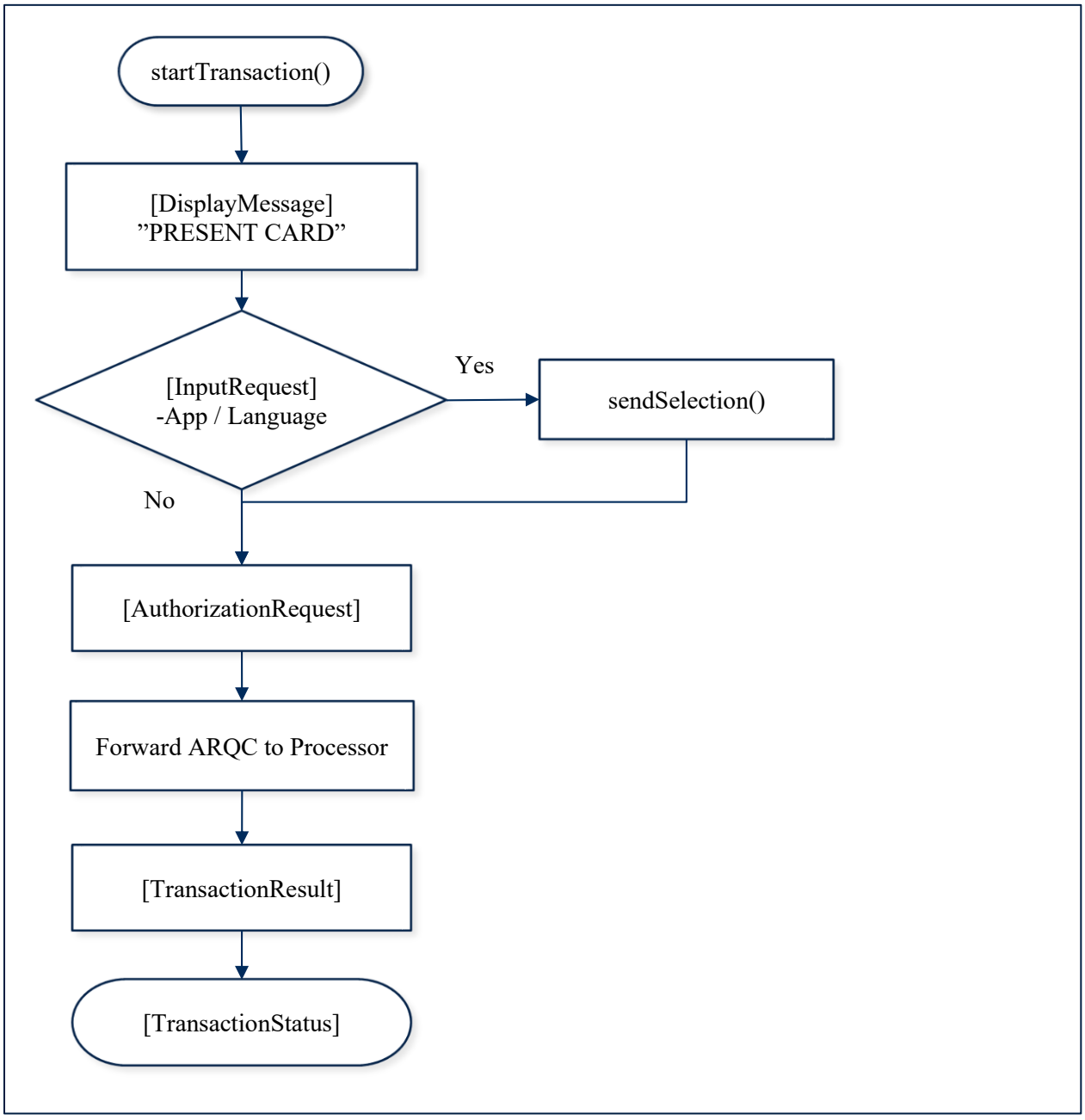
```
  public void OnResult(StatusCode status, byte[] data)
  {
     /* Handle result.
        A configuration process is complete when
        status = StatusCode.Success */
  }
```

```
  public IResult OnCalculateMAC(byte macType, byte[] data)
  {
    IResult result = new Result(StatusCode.UNAVAILABLE);
    byte[] macBytes = null;

    DeviceType deviceType =
     device.getConnectionInfo().getDeviceType();

    switch (deviceType)
    {
      case DeviceType.MMS:
        macBytes = getDynaFlexMAC(macType, data);
        break;
    }

    if (macBytes != null)
    {
      result = new Result(StatusCode.SUCCESS);
      result.Data = new BaseData(macBytes);
    }

    return result;
  }
}
```

Appendix C EMV Transaction Flow

This section demonstrates transaction flow.

C.1 Flow Chart - QuickChip



C.2 Sample Code - QuickChip

The following breaks out the EMV flow chart into code. When enabling QuickChip mode, host does not send the ARPC to the device to complete the transaction. Events are shown separately and in the order received.

```
// Assign parameters.
List<PaymentMethod> paymentMethod = new List<PaymentMethod>();
paymentMethod.Add(PaymentMethod.MSR);
paymentMethod.Add(PaymentMethod.Contact);
paymentMethod.Add(PaymentMethod.Contactless);

transaction = new Transaction();
transaction.Amount = "1.00";
transaction.CashBack = "0.00";
transaction.EMVOnly = true;
transaction.PaymentMethods = paymentMethod;
transaction.QuickChip = true;  // QuickChip mode enabled.

// Start transaction.
bool result = device.startTransaction(transaction);
```

```
public void OnEvent(EventType eventType, IData data)
{
  string message
  switch (eventType);
    {
      case EventType.DisplayMessage:
        // Get the message.
        message = data.StringValue;
  }
}
```

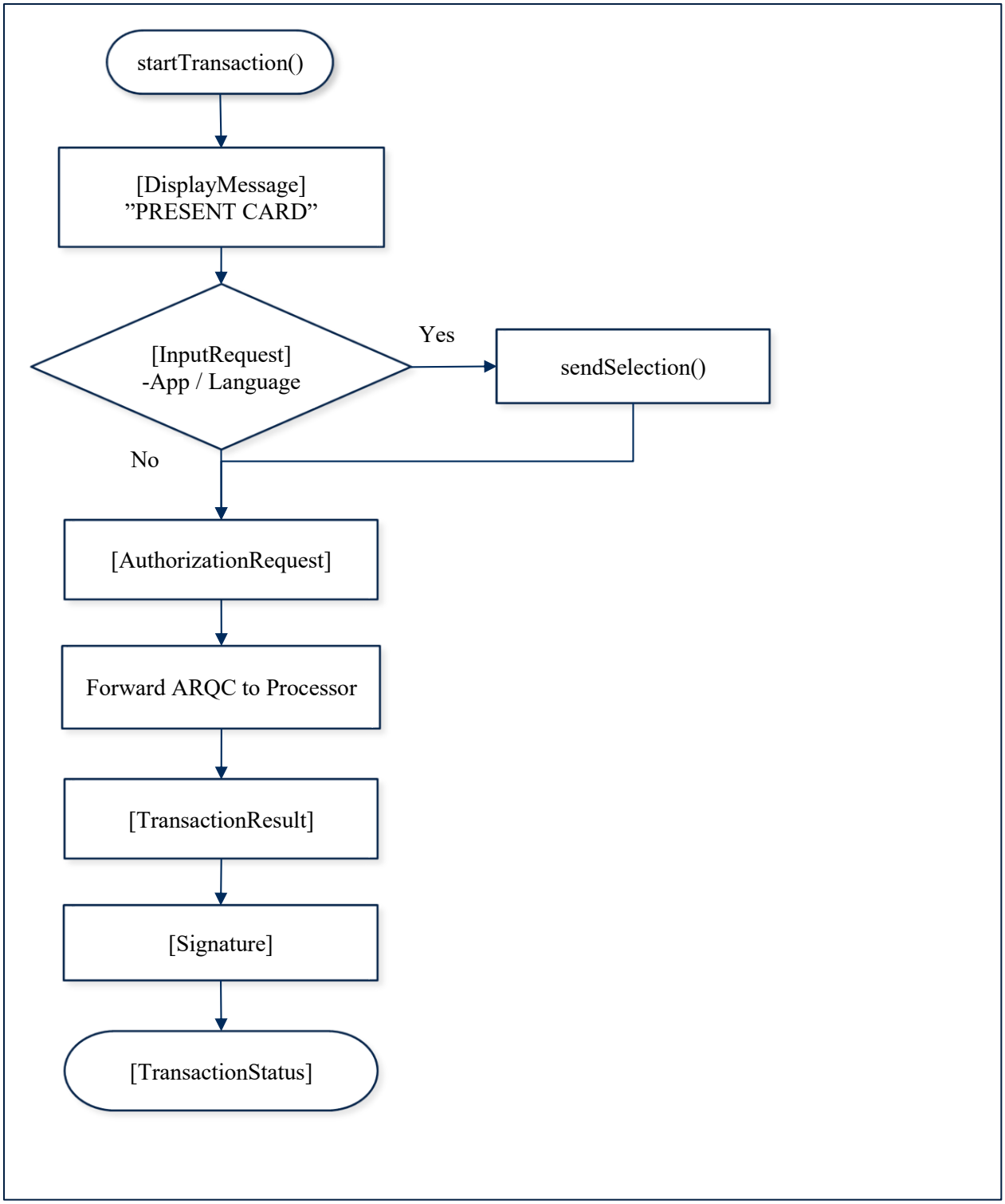
```
public void OnEvent(EventType eventType, IData data)
{
  string message;
  switch (eventType);
    {
      case EventType.InputRequest:
        // Get the message.
        message = data.StringValue;

        // display/retrieve user selection.

        // set status and selection result.
        IData selectionData = new BaseData(new Byte[] {status,
selection});
        device.sendSelection(selectionData);
    }
}
```

```
public void OnEvent(EventType eventType, IData data)
{
  byte[] ARQC = null;
  switch (eventType);
    {
      case EventType.AuthorizationRequest:
        // Forward ARQC to processor.
        /* data[0..1] – ARQC length
           data[2..n] – remainder contains the ARQC TLV object */
    }
}
```

```
public void OnEvent(EventType eventType, IData data)
{
  string message;
  switch (eventType);
    {
      case EventType.DisplayMessage:
        // Display approval message.
        message = data.StringValue;

        // A data size of 0 is an instruction to clear the display.
        if (data.StringValue.Length == 0)
        {
          // Clear the UI display.
        }
    }
}
```

```
public void OnEvent(EventType eventType, IData data)
{
  string message;
  switch (eventType);
    {
      case EventType.TransactionResult:
         /* data[0]    – Signature Required
            data[1..2] – Batch Data length
            data[3..n] – remainder contains the Batch Data TLV object
         */

        // Parse the TLV from data[].
        // Abstract Approval status from TLV tag “DFDF1A”.
        // Abstract Signature Required status from TLV tag data[0].
    }
}
```

C.3 Flow Chart – Signature Capture



C.4 Sample Code – Signature Capture

The following breaks out the EMV flow chart into code. Events are shown separately and in the order received.

```
// Assign parameters.
List<PaymentMethod> paymentMethod = new List<PaymentMethod>();
paymentMethod.Add(PaymentMethod.MSR);
paymentMethod.Add(PaymentMethod.Contact);
paymentMethod.Add(PaymentMethod.Contactless);

transaction = new Transaction();
transaction.Amount = "1.00";
transaction.CashBack = "0.00";
transaction.EMVOnly = true;
transaction.PaymentMethods = paymentMethod;
transaction.QuickChip = true;

// Start transaction.
bool result = device.startTransaction(transaction);
```

```
public void OnEvent(EventType eventType, IData data)
{
  string message;
  switch (eventType);
    {
      case EventType.DisplayMessage:

        // Get the message.
        message = data.StringValue;
  }
}
```

```
public void OnEvent(EventType eventType, IData data)
{
  string message;
  switch (eventType);
    {
      case EventType.InputRequest:
        // Get the message.
        message = data.StringValue;

        // display/retrieve user selection.

        // set status and selection result.
        IData selectionData = new BaseData(new Byte[] {status,
selection});
        device.sendSelection(selectionData);
    }
}
```

```
public void OnEvent(EventType eventType, IData data)
{
  byte[] ARQC = null;
  switch (eventType);
    {
      case EventType.AuthorizationRequest:
        // #4a
        // Forward ARQC to processor.
        /* data[0..1] – ARQC length
           data[2..n] – remainder contains the ARQC TLV object */
    }
}
```

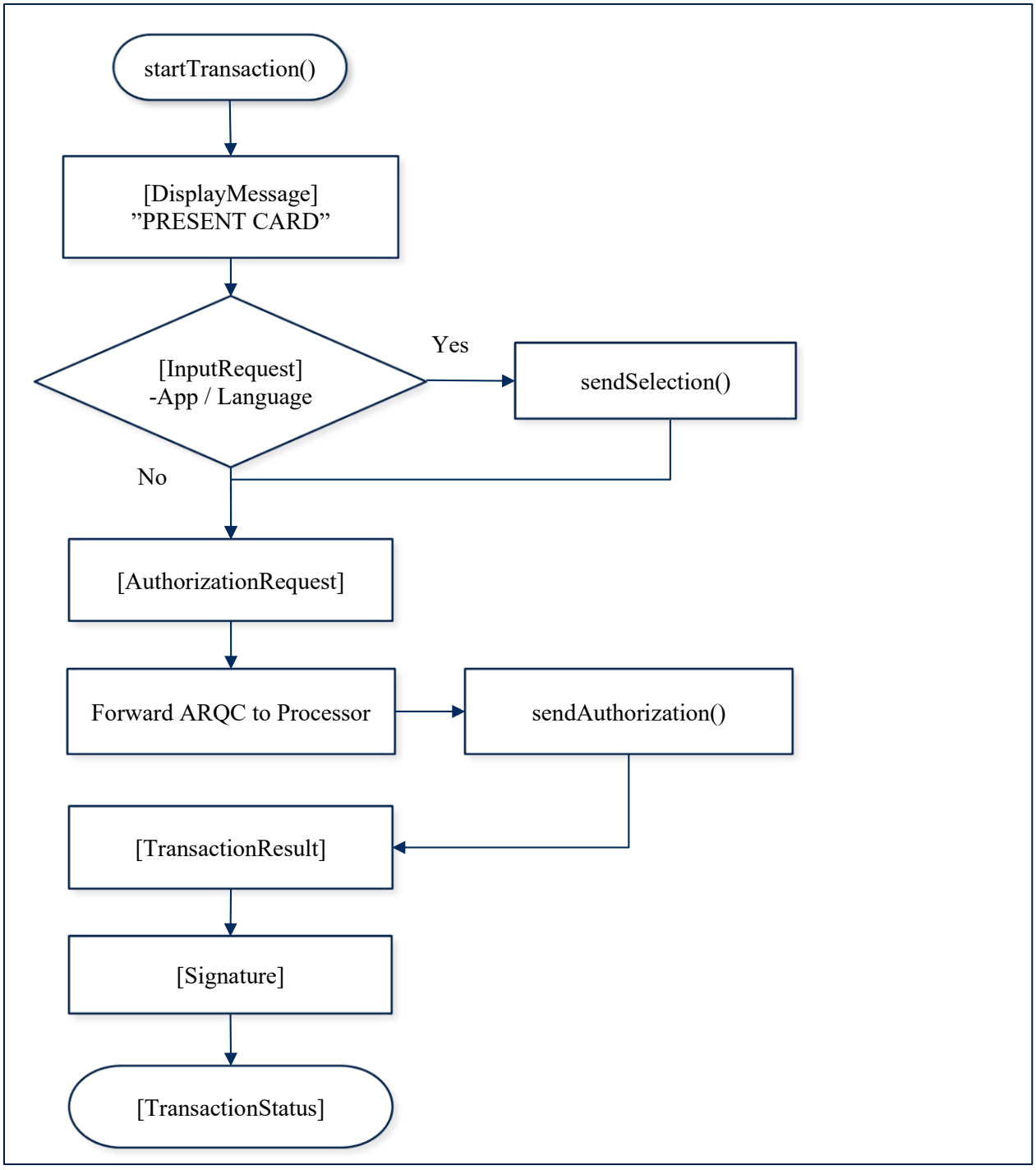
```
public void OnEvent(EventType eventType, IData data)
{
  string message;
  switch (eventType);
    {
      case EventType.DisplayMessage:
        // Display approval message.
        message = data.StringValue;

        // A data size of 0 is an instruction to clear the display.
        if (data.StringValue.Length == 0)
        {
          // Clear the UI display.
        }
    }
}
```

```
public void OnEvent(EventType eventType, IData data)
{
  string message;
  switch (eventType);
    {
      case EventType.TransactionResult:
         /* data[0]    – Signature Required
            data[1..2] – Batch Data length
            data[3..n] – remainder contains the Batch Data TLV object
         */

        // Parse the TLV from data[].
        // Abstract Approval status from TLV tag "DFDF1A".
        // Abstract Signature Required status from TLV tag data[0].
    }
}
```

```
public void OnEvent(EventType eventType, IData data)
{
```

```
  String signature;
  switch (eventType);
    {
      case EventType.Signature:
         signature = data.StringValue;
    }
}
```

C.5 Flow Chart – With ARPC



C.6 Sample Code – With ARPC

The following breaks out the EMV flow chart into code. When disabling QuickChip mode, host must send the ARPC to the device to complete the transaction. Events are shown separately and in the order received.

```
// Assign parameters.
List<PaymentMethod> paymentMethod = new List<PaymentMethod>();
paymentMethod.Add(PaymentMethod.MSR);
paymentMethod.Add(PaymentMethod.Contact);
paymentMethod.Add(PaymentMethod.Contactless);

transaction = new Transaction();
transaction.Amount = "1.00";
transaction.CashBack = "0.00";
transaction.EMVOnly = true;
transaction.PaymentMethods = paymentMethod;
transaction.QuickChip = false; //QuickChip mode disabled.

// Start transaction.
bool result = device.startTransaction(transaction);
```

```
public void OnEvent(EventType eventType, IData data)
{
  string message;
  switch (eventType);
    {
      case EventType.DisplayMessage:

        // Get the message.
        message = data.StringValue;
  }
}
```

```
public void OnEvent(EventType eventType, IData data)
{
  string message;
  switch (eventType);
    {
      case EventType.InputRequest:
        // Get the message.
        message = data.StringValue;

        // display/retrieve user selection.

        // set status and selection result.
        IData selectionData = new BaseData(new Byte[] {status,
selection});
        device.sendSelection(selectionData);
    }
}
```

```
public void OnEvent(EventType eventType, IData data)
{
  byte[] ARQC = null;
  switch (eventType);
    {
      case EventType.AuthorizationRequest:
        // Forward the ARQC to the processor.
        /* data[0..1] – ARQC length
           data[2..n] – remainder contains the ARQC TLV object */

        ARQC.ByteArray = data.ByteArray;
        // App function to send the request to the processor.
        ARPC = sendARQCToProcessorForApproval(ARQC.ByteArray);

    }
}
```

After the ARPC is returned from the processor, it is constructed into a TLV container and then sent to the device. The ARPC for approved (00) is set in ASCII 3030.

The optional tags 91, 71, and 72 (Issuer Authentication Data, Issuer Script Template 1, and Issuer Script Template 2) are not included in this example.

See the construction of the ARPCTLV in the table below.

```
String ARPC = "8A3030";
IData ARPCTLV = new BaseData("");
ARPCTLV.StringValue = "FF7413DFDF250742363243413546FA067004" + ARPC;

device.sendAuthorization(ARPCTLV);
```

ARPC TLV object for sendAuthorization().

Tag	Len	Value / Description	Typ	Req	Default
FF74	var	Container for non-MAC ARPC	T	R	
/DFDF25	var	Device Serial Number (IFD Serial Number)	B	R	
/FA	var	Container for generic data	T	R	
//70	var	Container for ARPC	T	R	
///8A	02	Authorization Response Code • 0x3030 = Approved • 0x3031 = Issuer Referral • 0x3035 = Declined • 0x3132 = Switch Interface • 0x3133 = Request Online PIN	AN	R	
///91	var	Issuer Authentication Data As defined in ***EMV Integrated Circuit Card Specifications for Payment Systems 4.3***	B	O	

Tag	Len	Value / Description	Typ	Req	Default
///71	var	Issuer Script Template 1 As defined in ***EMV Integrated Circuit Card Specifications for Payment Systems 4.3***. The host may include as many instances of this parameter as needed, up to a maximum length of 128 bytes including Tags and Lengths.	B	O	
///72	var	Issuer Script Template 2 As defined in ***EMV Integrated Circuit Card Specifications for Payment Systems 4.3***. The host may include as many instances of this parameter as needed, up to a maximum length of 128 bytes including Tags and Lengths.	B	O	

```
public void OnEvent(EventType eventType, IData data)
{
  string message;
  switch (eventType);
    {
      case EventType.DisplayMessage:
        // Display approval message.
        message = data.StringValue;

        // A data size of 0 is an instruction to clear the display.
        if (data.StringValue.Length == 0)
        {
          // Clear the UI display.
        }
    }
}
```

```
public void OnEvent(EventType eventType, IData data)
{
  string message;
  switch (eventType);
    {
      case EventType.TransactionResult:
         /* data[0]    - Signature Required
            data[1..2] - Batch Data length
            data[3..n] - remainder contains the Batch Data TLV object
         */

        // Parse the TLV from data[].
        // Abstract Approval status from TLV tag "DFDF1A".
        // Abstract Signature Required status from TLV tag data[0].
    }
}
```

```
public void OnEvent(EventType eventType, IData data)
{
  String signature;
  switch (eventType);
    {
      case EventType.Signature:
         signature = data.StringValue;
    }
}
```

C.7 MSR Fallback Flow

The use case for an MSR fallback is when communication with the chip results in a terminated transaction and the `TransactionStatus` is reported as `MSRFallback`.

The host application will re-attempt the transaction. To invoke this use case, here are the following pre-requisites.

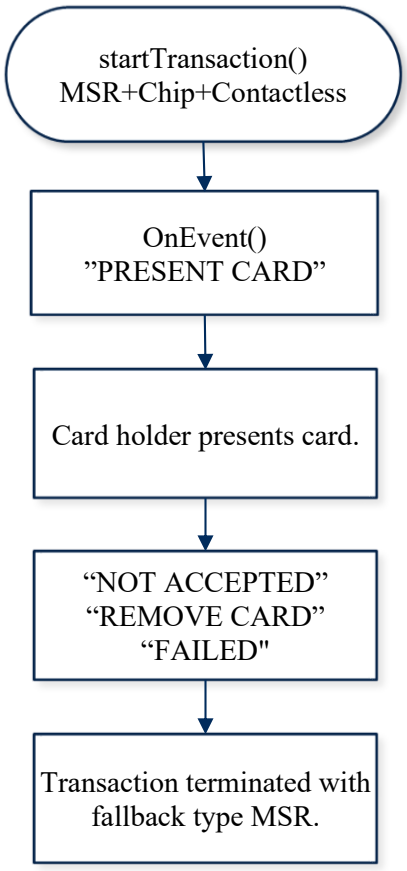
Pre-requisites:

- Device already configured for Device-Driven Fallback = Disabled.
- A card to cause the fallback. Example but not limited to a card with no applications programmed or a card with an application not configured on the device.

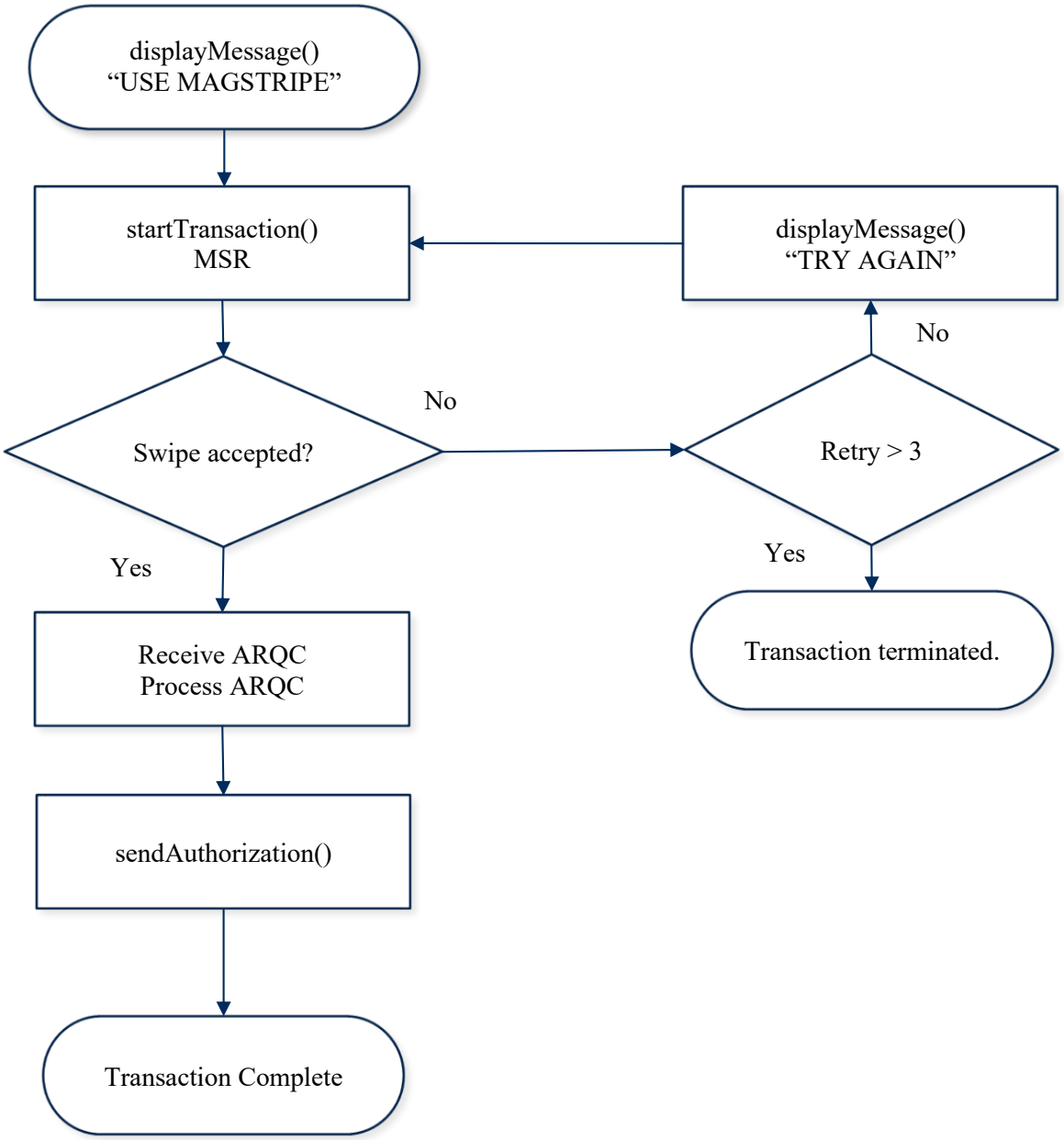
Scheme:

- →Host begins an initial transaction with `PaymentMethod` set to `MSR+Chip+Contactless`.
- ←Device responds with fail and with status of `MSRFallback`.
- →Host displays a message to use magstripe.
- →Host starts a transaction with `PaymentMethod` set to `MSR`.
- ←Device may respond with transaction cancelled card read error.
- →Host displays a message each time the transaction fails until successful or until Host decides to end the transaction.
- ←Device sends the transaction result.

Begin initial transaction:



Continue with Fallback transaction:



Appendix D Apple VAS

These instructions are for preparing Apple VAS (value-added service) transactions.

D.1 Merchant ID and URL Slots

- Set the Apple VAS Merchant ID and URL property for each slot 1 to 6 using IDeviceControl → `setConfigInfo()`.

D.2 POS Capabilities

- Set the POS capabilities property using IDeviceControl → `setConfigInfo()`.

D.3 Start Transaction

- Set the PaymentMethods to include `PaymentMethod.AppleVAS`.
- Set AppleVASMode to: `VASMode.Single`, `VASMode.Dual`, or `VASMode.VASOnly`.
- Set AppleVASProtocol to: `VASProtocol.Full` or `VASProtocol.URL`.

D.4 Transaction Response

- Data from an Apple VAS (9F27 and 9F2A) is returned in separate Apple VAS slot containers.

Tag	Length	Value /Description
FE	var	VAS Data Container
//FF01	var	Apple VAS Container Slot 1 Container
///9F27	var	VAS Data. Up to 128 bytes.
///9F2A	var	Mobile Token. Up to 36 bytes.
//FF02	var	Apple VAS Container Slot 2 Container
///9F27	var	VAS Data. Up to 128 bytes.
///9F2A	var	Mobile Token. Up to 36 bytes.
//FF03	var	Apple VAS Container Slot 3 Container
///9F27	var	VAS Data. Up to 128 bytes.
///9F2A	var	Mobile Token. Up to 36 bytes.
//FF04	var	Apple VAS Container Slot 4 Container

Tag	Length	Value /Description
///9F27	var	VAS Data. Up to 128 bytes.
///9F2A	var	Mobile Token. Up to 36 bytes.
//FF05	var	Apple VAS Container Slot 5 Container
///9F27	var	VAS Data. Up to 128 bytes.
///9F2A	var	Mobile Token. Up to 36 bytes.
//FF06	var	Apple VAS Container Slot 6 Container
///9F27	var	VAS Data. Up to 128 bytes.
///9F2A	var	Mobile Token. Up to 36 bytes.

Appendix E Google Wallet Smart Tap VAS

These instructions are for preparing Google Wallet Smart Tap VAS (value-added service) transactions.

E.1 Mobile Device

- Configure the mobile device for Google Wallet Smart Tap Pass.

E.2 Load Key

- Load the LTPK protection key (Long Term Private Key) into the MagTek device.
- Upload a Public Key to the Google Pay & Wallet Console for the issuer account associate with the Google Wallet Pass.

E.3 Collector ID Slots

- Set the Google Smart Tap Collector ID property for each slot 1 to 6 using IDeviceControl → `setConfigInfo()`.

E.4 POS Capabilities

- Set the POS capabilities property using IDeviceControl → `setConfigInfo()`.

E.5 Start Transaction

- Set the PaymentMethods to include `PaymentMethod.GoogleVAS`.
- Set AppleVASMode to: `VASMode.Single`, `VASMode.Dual`, or `VASMode.VASOnly`.

E.6 Transaction Response

- Data from a Google Wallet Smart Tap (DF7B) is returned in separate Collector ID slot containers associated with the Google Wallet Pass.

Tag	Length	Value /Description
FF41	var	Google Smart Tap Container
//FF01	var	Collector ID Slot 1 Container
///DF7B	var	Service Response NDEF Record
//FF02	var	Collector ID Slot 2 Container
///DF7B	var	Service Response NDEF Record
//FF03	var	Collector ID Slot 3 Container

Tag	Length	Value /Description
///DF7B	var	Service Response NDEF Record
//FF04	var	Collector ID Slot 4 Container
///DF7B	var	Service Response NDEF Record
//FF05	var	Collector ID Slot 5 Container
///DF7B	var	Service Response NDEF Record
//FF06	var	Collector ID Slot 6 Container
///DF7B	var	Service Response NDEF Record