MagTek Universal SDK

For MMS Devices
Programmer’s Manual (Java)

September 2021

Manual Part Number:
D998200385-10

REGISTERED TO ISO 9001:2015

Table 0.1 - Revisions

Rev Number	Date	Notes
10	September 15, 2020	Initial release

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.

Table of Contents

Table of Contents 5

1 Introduction 8

1.1 About the MagTek Sample Code 8

1.2 Nomenclature 8

1.3 SDK Contents 8

1.4 System Requirements 8

2 How to Set Up the MagTek Universal Project 9

2.1 How to Download and Set Up the MagTek Universal Java Project 9

2.2 How to Set Up the Java Library With the JRE/JVM 9

2.3 How to Modify Manifest 9

2.4 How to Sign JAR 10

2.5 How to Set Up the SDK in Eclipse 11

3 CoreAPI 12

3.1 getAPIVersion 12

3.2 getDevice 12

3.3 getDeviceList 12

3.4 getConnectionTypes 13

3.5 GetConnectionTypeString 13

3.6 GetConnectionTypeFromString 13

4 IDevice 14

4.1 cancelTransaction 14

4.2 getCapabilities 14

4.3 getConnectionInfo 14

4.4 getConnectionState 14

4.5 getDeviceControl 14

4.6 getDeviceConfiguration 14

4.7 getDeviceInfo 15

4.8 Name 15

4.9 requestPIN 15

4.10 requestSignature 16

4.11 sendAuthorization 16

4.12 sendSelection 16

4.13 startTransaction 16

4.14 subscribeAll 20

4.15 unsubscribeAll 20

5 IDeviceCapabilities 21

5.1 BatteryBackedClock 21

5.2 Display 21

5.3 MSRPowerSaver 21

5.4 PaymentMethods 21
5.5 PINPad 21
5.6 Signature 21
5.7 SRED 22
6 IDeviceControl 23
6.1 close 23
6.2 deviceReset 23
6.3 displayMessage 23
6.4 endSession 25
6.5 getInput 25
6.6 open 25
6.7 playSound 25
6.8 send 26
6.9 sendExtendedCommand 26
6.10 sendSync 26
6.11 setDateTime 27
6.12 setLatch 27
6.13 showImage 28
7 ConnectionInfo 29
7.1 getAddress 29
7.2 getConnectionType 29
7.3 getDeviceType 29
8 DeviceInfo 30
8.1 getModel 30
8.2 getName 30
8.3 getSerial 30
9 IDeviceConfiguration 31
9.1 getChallengeToken 31
9.2 getConfigInfo 31
9.3 getDeviceInfo 31
9.4 getFile 32
9.5 getKeyInfo 32
9.6 sendFile 32
9.7 sendImage 33
9.8 sendSecureFile 33
9.9 setConfigInfo 33
9.10 setDisplayImage 34
9.11 updateFirmware 34
9.12 updateKeyInfo 35
10 IEventSubscriber Delegates 36
10.1 OnEvent 36

11 IConfigurationCallback Delegates 37
11.1 OnCalculateMAC 37
11.2 OnProgress 37
11.3 OnResult 37
12 MTUSDK Enumerations 38
12.1 DeviceType 38
12.2 ConnectionType 39
12.3 ConnectionState 40
12.4 InfoType 40
12.5 PaymentMethod 40
12.6 StatusCode 41
12.7 TransactionStatus 41
12.8 EventType 42
12.9 UserEvent 42
Appendix A Status Codes 44
A.1 Library Status Codes 44
Appendix B API Walk Through 45
B.1 CoreAPI Walk Trough 45
B.2 IDevice Walk Through 46
B.2.1 Handling Events 47
B.3 IDeviceControl Walk Through 50
B.4 ConnectionInfo Walk Through 51
B.5 IDeviceCapabilities Walk Through 52
B.6 IDeviceConfiguration Walk Through 53
B.6.1 Handling Events 54
Appendix C EMV Transaction Flow 55
C.1 Flow Chart 55
C.2 Sample Flow Code 56
C.3 MSR Fallback Flow 62

1 Introduction

This document provides instructions for software developers who want to create Java software solutions that include MagTek devices connected to a Windows based host. This document is part of a larger library of documents designed to assist MagTek device implementers, which includes the following documents available from MagTek:

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

1.1 About the MagTek Sample Code

The sample code provides Java demonstration source code and a reusable MTUSDK library that provides developers of custom software solutions with an easy-to-use interface for MagTek devices. Developers can distribute the MTUSDK 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
MTUDK.jar	Java built SDK.
MTMMS.dll MTUSDK.dll	These are dependency components of MTUSDK for communicating with devices.

1.4 System Requirements

Tested operating systems:

- Windows 8.1
- Windows 10

Java 8 and above

2 How to Set Up the MagTek Universal Project

2.1 How to Download and Set Up the MagTek Universal Java Project

To set up the MT Universal Libraries, follow these steps:

1) Download the ***1000008300-Web.zip***, available from MagTek.com

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

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

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

2.3 How to Modify Manifest

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

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

The **Application-Library-Allowable-Codebase** attribute identifies the locations where your signed RIA is expected to be found. This attribute is used to determine what is listed in the Location field for the security prompt that is shown to users when the JAR file for your RIA is in a different location than the

JNLP file or HTML page that starts your RIA. If the files are not in the locations identified, the RIA is blocked. Set this attribute to the domains where the JAR file, JNLP file, and HTML page are located. To specify more than one domain, separate the domains by a space, for example:

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

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

In order to modify the Manifest file, please follow these steps.

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

2.4 How to Sign JAR

These instructions provide an overview of obtaining and using Sun Java signing and a digital certificate.

1) Make sure your machine has the latest Java JDK installed.
2) Generate a public/private key pair by entering the following command, specifying an alias for your keystore:
   ```
   keytool -genkey -keyalg rsa -alias MyCert
   ```
3) Generate a certificate signing request (CSR) by entering the following command:
   ```
   keytool -certreq -alias MyCert
   ```
   After prompting you to enter the password for your keystore, keytool will generate a CSR.
4) Save the certificate received from the Certificate provider as Certname.p7b.
5) Import your Digital Certificate by entering the following command:
   ```
   keytool -import -alias MyCert -file Certname.p7b
   ```
   In this string, keytool is requested to import the Digital ID “Certname.cer” into the keystore MyCert.
6) Bundle your applet into a Java Application Resource (JAR) file by entering the following command:
   ```
   jar cvf C:\mtusdk.jar
   ```
7) Sign your applet by using jarsigner to sign the JAR file, using the private key you saved in your keystore:
   ```
   jarsigner C:\mtusdk.jar MyCert
   ```
8) Verify the output of your signed JAR file by entering the following command:
   ```
   jarsigner -verify -verbose -certs C:\mtusdk.jar
   ```

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

2.5 How to Set Up the SDK in Eclipse

To set up the MT Universal Java Libraries, download and install package from MagTek.com.

1) On the master workstation, navigate to the root of the MagTek Universal SDK for Java. By default, it will be **...\Library**.
2) In Eclipse, click File → New → Java Project
3) Type the project name
4) In the library tab, click Add external jar, then point to the Lib folder of the root folder of MagTek Universal SDK for Java to add the MTUSDK.jar.

3 CoreAPI

Use the CoreAPI to create an **IDevice**. **IDevice** 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 getAPIVersion

This function returns the version of the MagTek Universal SDK.

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

Return Value:
Returns integer.

3.2 getDevice

This function returns an instance of a device.

```
IDevice CoreAPI.getDevice(
      DeviceType deviceType,
      ConnectionType connectionType,
      String deviceAddress);
```

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

deviceAddress	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`

Return Value:
Returns an `IDevice`.

3.3 getDeviceList

This function returns a list of **IDevice**.

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

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

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

Return Value:
Returns `List<IDevice>`.

3.4 getConnectionTypes

This function returns a list of **ConnectionType** supported for a particular device type.

```
List<ConnectionType> CoreAPI.getConnectionTypes(DeviceType
deviceType);
```

Parameter	Description
deviceType	Enumerated device type.

Return Value:
Returns a list of `List<ConnectionType>`.

3.5 GetConnectionTypeString

This function returns a string for a connection type.

```
String CoreAPI.GetConnectionTypeString(ConnectionType connectionType);
```

Parameter	Description
connectionType	Enumerated connections type.

Return Value:
Returns a string for a connection type.

3.6 GetConnectionTypeFromString

This function returns an enumerated connection type from a string.

```
ConnectionType CoreAPI.GetConnectionTypeFromString(String
connectionType);
```

Parameter	Description
connectionType	Enumerated connections type.

Return Value:
Returns a `ConnectionType`.

4 IDevice

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

4.1 cancelTransaction

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

```
boolean IDevice.cancelTransaction();
```

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

4.2 getCapabilities

This function retrieves the **IDeviceCapabilities** interface to 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 getDeviceControl

This function retrieves the device control interface to the device.

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

Return Value:
Returns **IDeviceControl**

4.6 getDeviceConfiguration

This function retrieves a device configuration interface to the device.

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

Return Value:
Returns **IDeviceConfiguration**

4.7 getDeviceInfo

This function returns an information class of the device.

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

Return Value:
Returns **DeviceInfo**

4.8 Name

This function retrieves the name of the device.

```
String IDevice.Name();
```

Return Value:
Returns a string containing the device name.

4.9 requestPIN

This function prompts the user to enter a PIN on the device. The response data will be returned in the event **OnEvent**. requestPin() is reserved for future use.

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

Parameter	Type	Description
Timeout	byte	Wait time in seconds.
PINMode	byte	PIN mode. Usage: `0x00` - Enter PIN `0x01` - Enter PIN Amount `0x02` - Reenter PIN Amount `0x03` - Reenter PIN `0x04` - Verify PIN
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
Format	byte	ISO format for the PIN block.
PAN	String	The left most 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 requestSignature

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

```
boolean IDevice.requestSignature();
```

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

4.11 sendAuthorization

This function sends the Authorization Response Code (ARPC) blob to the device. The response data will be returned in the event **OnEvent**.

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

Parameter	Description
data	Contains ARPC blob.

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

4.12 sendSelection

This function send a user selection to the device.

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

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

4.13 startTransaction

This function starts a transaction. The transaction will be processed through multiple calls to the event **OnEvent**.

```
boolean IDevice.startTransaction(ITransaction transaction);
```

ITransaction		
Parameter	Type/ Format	Description
Timeout	byte	Transaction timeout in seconds. Default is 60 seconds. Usage: `0 to 255` - Depending on the device, 0 means no timeout.
PaymentMethods	List of PaymentMethod	List of the PaymentMethod enumeration. Usage: `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.
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. Usage: `false` - Do not enable QuickChip mode. `true` - Enable QuickChip mode. Default.
EMVOnly	bool	Flag that determines whether or not to start an EMV transaction. Usage: `false` - Do not start transaction if the device does not support EMV. `true` - Only start transaction if the device supports EMV. Default.
PreventMSRSignatureForCardWithICC	bool	Flag that determines whether or not to prompt for a signature for magnetic stripe when the transaction is from a chip card. Usage: `false` – Do not prompt for signature. `true` – Allow the prompt for a signature if requested.
SuppressThankYouMessage	bool	By default, devices with a display signal the end of a transaction by briefly showing “THANK YOU,” then “WELCOME.” Usage: `false` – Do not suppress the thank you message. `true` – Suppress the thank you message.

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.
		
		Usage:
		`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”`
		`0x08 - “ENTER AMOUNT”`
		`0x09 - reserved, do not use.`
		`0x0A - reserved, do not use.`
		`0x0B - “INSERT CARD”`
		`0x0C – “NOT ACCEPTED”`
		`0x0D - reserved, do not use.`
		`0x0E - “PLEASE WAIT”`
		`0x0F - “PROCESSING ERROR”`
		`0x10 - “REMOVE CARD”`
		`0x11 - “USE CHIP READER”`
		`0x12 - “USE MAGSTRIPE”`
		`0x13 - “TRY AGAIN”`
		`0x14 - “WELCOME”`
		`0x15 - “PRESENT CARD”`
		`0x16 - “PROCESSING”`
		`0x17 - “CARD READ OK - REMOVE CARD”`
		`0x18 - “INSERT OR SWIPE CARD”`
		`0x19 - “PRESENT ONE CARD ONLY”`
		`0x1A - “APPROVED PLEASE SIGN”`
		`0x1B - “AUTHORIZING PLEASE WAIT”`
		`0x1C - “INSERT, SWIPE OR TRY ANOTHER CARD”`
		`0x1D - “PLEASE INSERT CARD”`
		`0x1E - Null prompt (empty screen)`
		`0x1F - reserved, do not use.`
		`0x20 - “SEE PHONE”`
		`0x21 - “PRESENT CARD AGAIN”`
		`0x22 - “INSERT/SWIPE/TRY OTHER CARD”`
		`0x23 - “TAP or SWIPE CARD”`
		`0x24 - “TAP or INSERT CARD”`
		`0x25 - “TAP, INSERT or SWIPE CARD”`
		`0x26 - “TAP CARD”`
		`0x27 - “TIMEOUT”`
		`0x28 - “TRANSACTION TERMINATED”`

ITransaction		
EMVResponseFormat	byte	The format of the EMV response. Usage: 0x00 – Legacy. Default. 0x01 – RFU
TransactionType	byte 1	EMV Tag 9C - The type of financial transaction, represented by the first two digits of the ISO 8583:1987 Processing Code. Examples: `0x00` – purchase. Default. `0x01` – cash advance `0x09` – purchase with cashback `0x20` – refund Supported transaction types are specified in the commands manual specific to the device.
Amount	String 12	EMV Tag 9F02 - Authorized amount of the transaction. Example: `"000000000123"` – $1.23 `"000000009999"` – $99.99
CashBack	String 12	EMV Tag 9F03 - Secondary amount associated with the transaction. Example: `"000000000123"` – $1.23 `"000000009999"` – $99.99
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 position from the right.
TransactionCategory	byte[] 1	EMV Tag 9F53 - The type of contactless transaction being performed. The byte array is null by default.
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.
MerchantID	byte[] 15	EMV Tag 9F16 - Used to uniquely identify a given merchant. The byte array is null by default.

ITransaction		
MerchantCustomData	byte[] 20	EMV Tag 9F7C – Proprietary merchant data that may be requested. The byte array is null by default.

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

4.14 subscribeAll

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

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

Parameter	Description
eventCallback	Name of a class that implements the **IEventSubscriber Delegates** event.

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

4.15 unsubscribeAll

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

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

Parameter	Description
eventCallback	Name of a class that implements the **IEventSubscriber Delegates** event.

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

5 IDeviceCapabilities

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

5.1 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.

```
boolean IDeviceCapabilities.BatteryBackedClock();
```

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

5.2 Display

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

```
boolean IDeviceCapabilities.Display();
```

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

5.3 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.

```
boolean IDeviceCapabilities.MSRPowerSaver();
```

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

5.4 PaymentMethods

This property returns a list of **PaymentMethods** supported by the device.

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

Return Value:
Returns a list of payment method enumerations.

5.5 PINPad

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

```
boolean IDeviceCapabilities.PINPad();
```

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

5.6 Signature

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

```
boolean IDeviceCapabilities.Signature();
```

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

5.7 SRED

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

```
boolean IDeviceCapabilities.SRED();
```

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

6 IDeviceControl

Create an instance of the **IDeviceControl**, then use the function calls described in this chapter.

6.1 close

This function closes the connection to the device.

```
boolean 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.

```
boolean IDeviceControl.deviceReset();
```

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

6.3 displayMessage

This function sets a show message on the device's display.

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

Parameter	Description

messageID	Value for the message ID.
	
	Usage:
	`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"`
	`0x08 - "ENTER AMOUNT"`
	`0x09 - reserved, do not use.`
	`0x0A - reserved, do not use.`
	`0x0B - "INSERT CARD"`
	`0x0C – "NOT ACCEPTED"`
	`0x0D - reserved, do not use.`
	`0x0E - "PLEASE WAIT"`
	`0x0F - "PROCESSING ERROR"`
	`0x10 - "REMOVE CARD"`
	`0x11 - "USE CHIP READER"`
	`0x12 - "USE MAGSTRIPE"`
	`0x13 - "TRY AGAIN"`
	`0x14 - "WELCOME"`
	`0x15 - "PRESENT CARD"`
	`0x16 - "PROCESSING"`
	`0x17 - "CARD READ OK - REMOVE CARD"`
	`0x18 - "INSERT OR SWIPE CARD"`
	`0x19 - "PRESENT ONE CARD ONLY"`
	`0x1A - "APPROVED PLEASE SIGN"`
	`0x1B - "AUTHORIZING PLEASE WAIT"`
	`0x1C - "INSERT, SWIPE OR TRY ANOTHER CARD"`
	`0x1D - "PLEASE INSERT CARD"`
	`0x1E - Null prompt (empty screen)`
	`0x1F - reserved, do not use.`
	`0x20 - "SEE PHONE"`
	`0x21 - "PRESENT CARD AGAIN"`
	`0x22 - "INSERT/SWIPE/TRY OTHER CARD"`
	`0x23 - "TAP or SWIPE CARD"`
	`0x24 - "TAP or INSERT CARD"`
	`0x25 - "TAP, INSERT or SWIPE CARD"`
	`0x26 - "TAP CARD"`
	`0x27 - "TIMEOUT"`
	`0x28 - "TRANSACTION TERMINATED"`

Parameter	Description
timeout	Timeout in seconds for the device to display the message. Usage: `0x00` - Infinite timeout. Device leaves the requested message on the display until the host initiates a change. All other values - 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.

```
boolean IDeviceControl.endSession();
```

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

6.5 getInput

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

```
boolean IDeviceControl.getInput(IData data);
```

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

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

6.6 open

This function opens a connection to the device.

```
boolean IDeviceControl.open();
```

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

6.7 playSound

This function instructs the device to play a tone.

```
boolean 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 returned to the event **OnEvent**.

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

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

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

6.9 sendExtendedCommand

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

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

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

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

6.10 sendSync

This function sends a synchronous command to the device. The response from the device will not be returned to the event **OnEvent**. The response is returned as IResult which contains the StatusCode and IData.

```
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;
}

public class Result implements IResult
{
    private StatusCode mStatus;
    private IData mData = null;

    public Result(StatusCode status)
    {
        mStatus = status;
        mData = null;
    }

    public Result(StatusCode status, IData data)
    {
        mStatus = status;
        mData = null;

        if (data != null)
            mData = data.Clone();
    }

    public StatusCode Status()
    {
        return mStatus;
    }

    public IData Data()
    {
        return mData;
    }
}


//Example Usage:
IResult result = deviceControl.sendSync(data);
Log("result = " + result.StatusCode());
Log("result = " + result.Data().StringValue());
```

6.11 setDateTime

This function sets the date and time for the device.

```
boolean 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.

6.12 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.

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

Parameter	Description
enableLock	Usage: `false` – unlock the latch in the device. `true` – lock the latch in the device.

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

6.13 showImage

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

```
boolean 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.

7 ConnectionInfo

Create an instance of the **ConnectionInfo**, then use the function calls described in this chapter.

7.1 getAddress

This function returns 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**

8 DeviceInfo

Create an instance of the **DeviceInfo** from **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 address of the device.

8.2 getName

This function returns the name of the device.

```
String DeviceInfo.getName();
```

Return Value:
Returns the address of the device.

8.3 getSerial

This function returns the serial number of the device.

```
String DeviceInfo.getSerial();
```

Return Value:
Returns the address 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 Delegates**.
- **Synchronous** functions return data in the return value. If the data is not available immediately, the call will block until a wait time has elapsed.

9.1 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.

9.2 getConfigInfo

This function retrieves device configuration information. For an example, see appendix **12.9B.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.

9.3 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.

9.4 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.

9.5 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.

9.6 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.
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.

9.7 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	File contents 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.

9.8 sendSecureFile

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

```
int IDeviceConfiguration.sendSecureFile(
      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.
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 setConfigInfo

This function sets device configuration information. For an example, see appendix **12.9B.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.

9.10 setDisplayImage

This function sets which image is to be displayed when the device is idle. The device requires a reset for the setting to take effect.

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

Parameter	Description
imageID	Value for the image ID. For DynaFlex, use: `0, 1, 2, 3, or 4` Use `0` to set the display image back to the "Welcome" screen.

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

9.11 updateFirmware

This function updates the device firmware. Progress of the update is reported in the event **OnProgress**.

```
int IDeviceConfiguration.updateFirmware(
      int firmwareType,
      byte[] data,
      IConfigurationCallback callback);
```

Parameter	Description
firmwareType	Type of firmware. For DynaFlex, use: `1` – Main 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.

9.12 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 IEventSubscriber Delegates

MagTeK Universal SDK will invoke the callback function in this chapter to provide the requested data and/or a detailed response. To delegate the event, call the **subscribeAll** function with the name of a class that implements the **IEventSubscriber Delegates** interface.

10.1 OnEvent

OnEvent handles most event types. The `eventType` parameter defines which event is triggered.

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

Parameter	Description
eventType	An enumeration indicating the event triggered by the device.
data	Contains the data for the event.

Return Value: None

Java Example:

```
@Override
public void OnEvent(EventType eventType, IData data)
{
    // Event handler
}
```

11 IConfigurationCallback Delegates

MagTek Universal SDK will invoke the callback function in this chapter to provide the requested data and/or a detailed response. These events will be called in a class that implements the **IConfigurationCallback Delegates** interface.

11.1 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.

11.2 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

11.3 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

12 MTUSDK Enumerations

12.1 DeviceType

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

Enum	Description
`SCRA`	Reserved for future use. Secure Reader Authenticator devices. • eDynamo • mDynamo • Dynamag • DynaMax • tDynamo • kDynamo • cDynamo • iDynamo 6
`PPSCRA`	Reserved for future use. PIN Pad Secure Reader Authenticator devices. • DynaPro • DynaPro Go • DynaPro Mini
`CMS`	Reserved for future use. Common Message Structure devices. • oDynamo
`MMS`	MMS class devices. (MagTek Message Scheme) • DynaFlex • DynaFlex Pro

12.2 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: • DynaFlex • DynaFlex Pro
`BLUETOOTH_LE`	Reserved for future use. Bluetooth Low Energy devices: • DynaMax
`BLUETOOTH_LE_EMV`	Reserved for future use. Bluetooth Low Energy with EMV supported devices: • eDynamo
`BLUETOOTH_LE_EMVT`	Reserved for future use. Bluetooth Low Energy with EMV supported devices: • tDynamo
`TCP`	Reserved for future use. Transmission Control Protocol supported devices: • DynaPro
`TCP_TLS`	Reserved for future use. Transmission Control Protocol with Transport Layer Security supported devices: • DynaPro Go
`TCP_TLS_TRUST`	Reserved for future use. Transmission Control Protocol with Transport Layer Security supported devices: • DynaPro Go
`WEBSOCKET`	Reserved for future use. WebSocket supported devices: • DynaFlex Pro
`SERIAL`	Reserved for future use. UART supported devices
`AIDL`	Reserved for future use. AIDL devices: • DynaGlass
`VIRTUAL`	Reserved for future use. Virtual devices

12.3 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.
`Disconnecting`	Device is in the process of disconnecting. The next state will is to be `Disconnected`.

12.4 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.
`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

12.5 PaymentMethod

This enum refers to card type for which 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.

12.6 StatusCode

This enum refers to returned status code of a function call.

Enum	Description
SUCCESS	Operation completed successfully.
TIMEOUT	Operation timed out before completion.
ERROR	Error in the operation.
UNAVAILABLE	Status currently unavailable.

12.7 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 is 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.
HostCanceled	The host software sent a cancel.
TransactionCanceled	The transaction was canceled.
TransactionInProgress	The transactions is in progress.
TransactionError	There is an error during the transaction.
TransactionApproved	The transactions is approved.
TransactionDeclined	The transactions is declined.
TransactionCompleted	The transaction is completed.

Enum	Description
`TransactionFailed`	The transaction failed.
`TransactionNotAccepted`	The transaction was not accepted by the device.
`TechnicalFallback`	Due to technical reasons, the chip transaction cannot be completed by the reader.
`SignatureCaptureRequested`	A signature capture is requested by the device.

12.8 EventType

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

Enum	Description
`Invalid`	
`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 is sending a notification.
`CardData`	Device has sent magnetic stripe data from a card swipe.
`TransactionStatus`	There was a change in transaction status.
`DisplayMessage`	Device has data to be displayed to the user.
`InputRequest`	Device is requesting 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 is sending 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 data a file.
`OperationStatus`	Device has sent an operation status of a command.

12.9 UserEvent

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

Enum	Description
`None`	No events yet to occur.
`ContactlessCardPresented`	Contactless card has been presented.

Enum	Description
`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 press detected.
`TouchRemoved`	Touch screen release detected.

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.

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>

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 **IDevice** is established.

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

IDevice mtmms = CoreAPI.getDevice(
      DeviceType.MMS,
      ConnectionType.USB,
      "",);
mtmms.requestSignature();
```

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.get(0).requestSignature();
```

B.2 IDevice Walk Through

The following walks through how to make use of **IDeviceError! Reference source not found.**.

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

Example

```
import comp.magtek.mtusdk;
.

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

/* For a list of a single device type.
DeviceType deviceType = DeviceType.MMS;
List<IDevice> deviceList = CoreAPI.getDeviceList(deviceType);
IDevice device = deviceList.get(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.get(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. */
boolean return = device.unsubscribeAll(this);
boolean return = device.subscribeAll(this);

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

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

Transaction transaction = new Transaction();
transaction.setAmount("1.00");
transaction.setCashBack("0.00");
transaction.setEMVOnly(true);
transaction.setPaymentMethods(paymentMethod);
transaction.setQuickChip(false);

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

B.2.1 Handling Events

Application Main window may extent the Error! Reference source not found. or can be extended by a eparate 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 implements IEventSubscriber
{
  @Override
  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.
if (data != NULL)
{
  message =
TransactionStatusBuilder.GetStatusDetail(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 processorARPC;
procARPC.byteArray = sendForAuthorization(data.ByteArray);

// Send authorization to device when not in QuickChip mode.
if (transaction.QuickChip == false)
{
  device.sendAuthorization(procARPC.ByteArray);
}

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.PINBlock:

break;
```

```
case EventType.Signature:

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.get(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.get(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 Error! Reference source not found..

- 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.get(0);


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


// Retrieve device capabilities.
boolean 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.get(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 = 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 custruction and instead
is passed as the configType.
*/
data = new byte[] {0xE2,0x08,0xE1,0x06,0xE1,0x04,0xE1,0x02,0xC1,0x00
};
byte[] response = devConfig.getConfigInfo(0x01, 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 custruction and instead
is passed as the configType.

*/
data = new byte[]
{0xE2,0x09,0xE1,0x07,0xE1,0x05,0xE1,0x03,0xC1,0x01,0x00 };
result = devConfig.getConfigInfo(0x01, 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

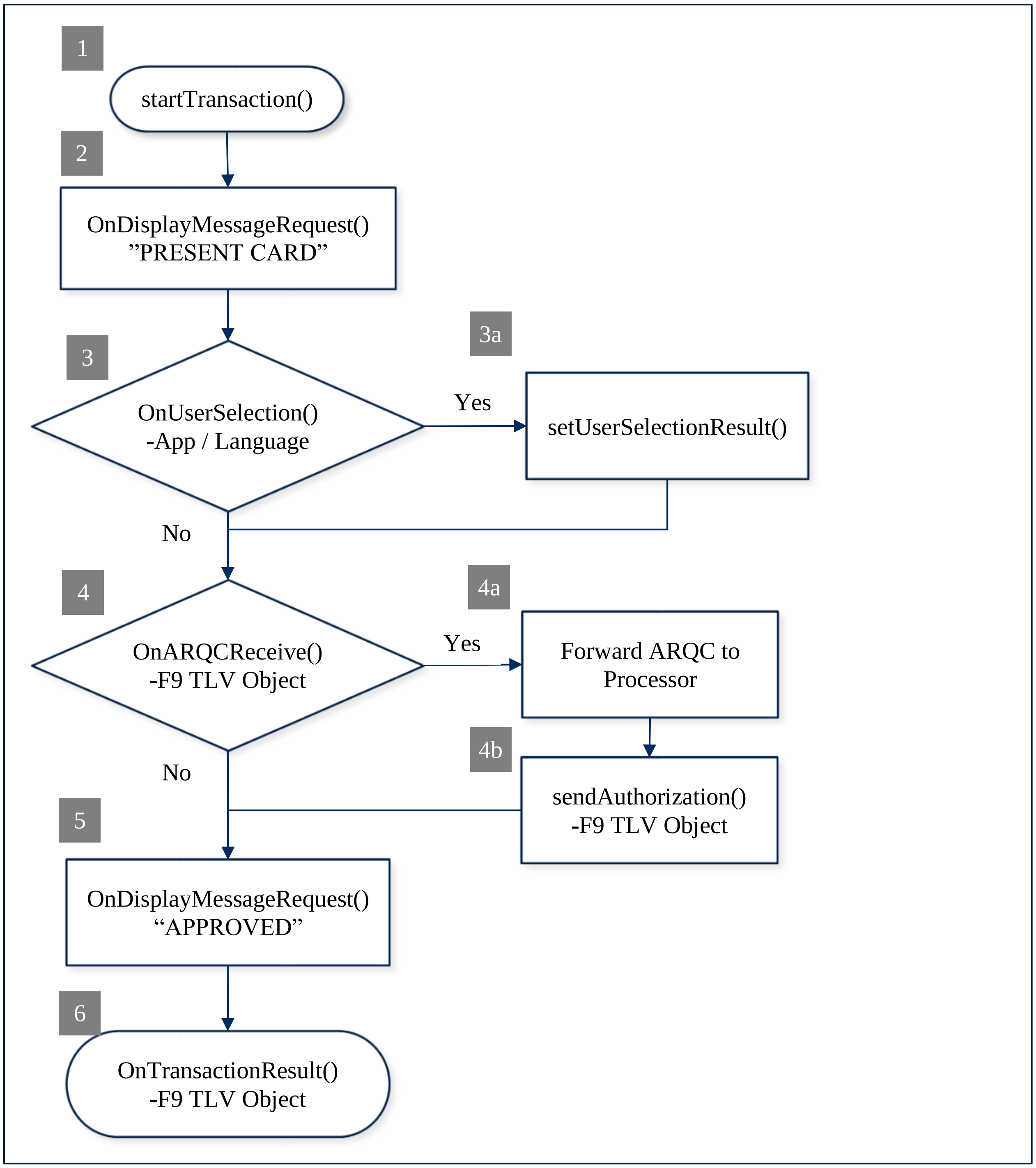
```
// A class to handle configuration callback events.
public class ConfigCallbacks : MTUSDK.IConfigurationCallback
{

  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;
    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, new BaseData(macBytes));
    }

    return result;
  }

}
```

Appendix C EMV Transaction Flow

C.1 Flow Chart



C.2 Sample Flow Code

The following breaks out the EMV flow chart into code.

```
// #1

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

Transaction transaction = new Transaction();
transaction.setAmount("1.00");
transaction.setCashBack("0.00");
transaction.setEMVOnly(true);
transaction.setPaymentMethods(paymentMethod);
transaction.setQuickChip(false);

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

```
// #2

@Override
public void OnEvent(EventType eventType, IData data)
{
  String message;

  switch (eventType);
    {
      case EventType.DisplayMessage:

        // Get the message.
        if (data != NULL)
        {
          message = data.StringValue();
        }

        break;
  }
}
```

```
// #3

@Override
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 IData;
        selectionData[0] = status;
        selectionData[1] = selection;
        device.sendSelection(data);

        break;
    }
}
```

```
// #4

@Override
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 */

        IData processorARPC;
        procARPC.byteArray = sendForAuthorization(data.ByteArray());


        // #4b
        // Send authorization to device when not in QuickChip mode.
        if (transaction.QuickChip == false)
        {
          device.sendAuthorization(procARPC.ByteArray());
        }

        break;
    }
}
```

```
// #5

@Override
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.
        }

        break;
    }
}
```

```
// #6

@Override
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].
        .
        break;
    }
}
```

C.3 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.

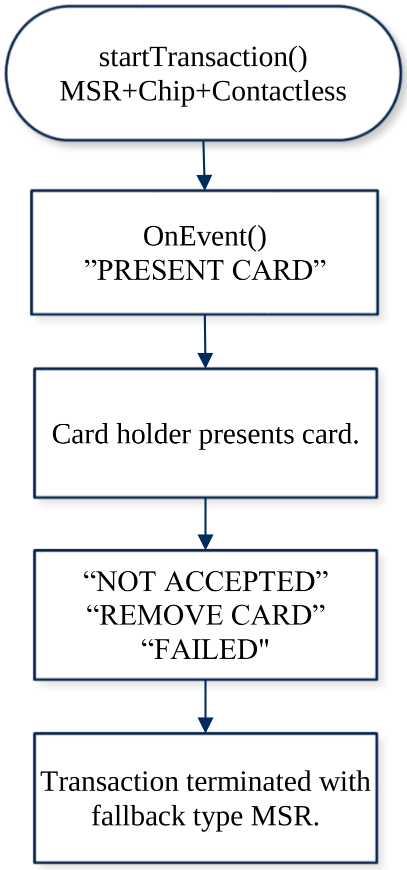
Pre-requisites:

- 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.

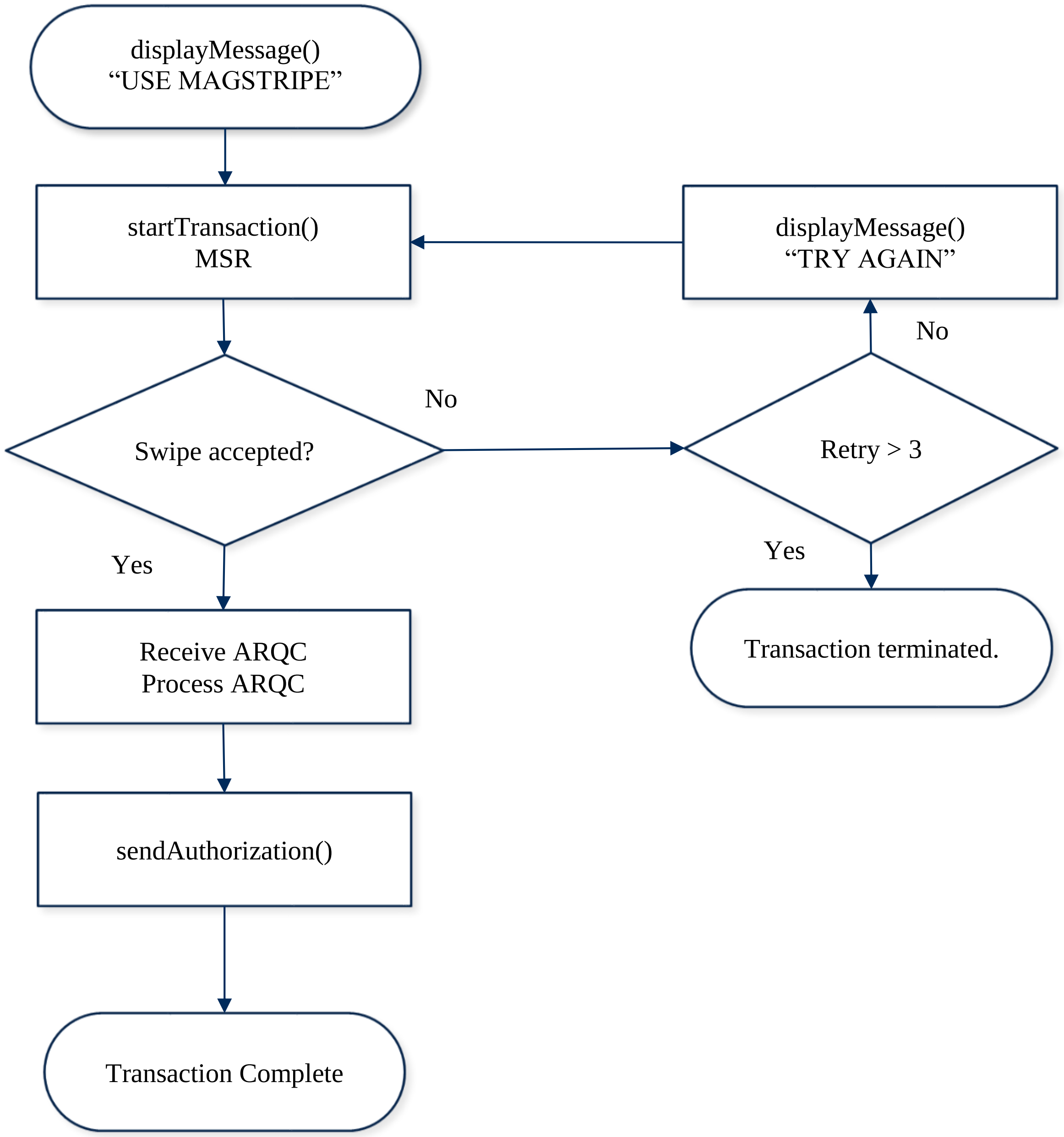
Scheme:

- →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 canceled 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:


displayMessage()
“USE MAGSTRIPE”
startTransaction()
MSR
displayMessage()
“TRY AGAIN”
No
No
Swipe accepted?
Retry > 3
Yes
Yes
Receive ARQC
Process ARQC
Transaction terminated.
sendAuthorization()
Transaction Complete