IDTech Windows SDK Guide

for Kiosk III/IV

#80136501-001

Rev. A

Revision History

Revision	Description and Reason for Change	Date
A	Initial Release - Manual;User;MiniSmartII;SDK;Win	1/12/2016
A	Added Kiosk IV Reference	6/20/2018

Contents

Chapter 1

IDTech Windows SDK Reference Guide for Kiosk III/IV

IDTechSDK.dll is a Windows dynamic link library that will be provided by IDTech as the main interface between Windows Form (WinForms) the Kiosk III/IV and payment processing solutions.

The purpose of this document is to describe the requirements of the API as well as the interface definitions and requirements needed for a WinForms application wishing to deploy with the payment application.

Chapter 2

Important Security Notice

The Payment Card Industry Payment Application Data Security Standard (PCI PA-DSS) is comprised of fourteen requirements that support the Payment Card Industry Data Security Standard (PCI DSS). The PCI Security Standards Council (PCI SSC), which was founded by the major card brands in June 2005, set these requirements in order to protect cardholder payment information. The standards set by the council are enforced by the payment card companies who established the Council: American Express, Discover Financial Services, JCB International, MasterCard Worldwide, and Visa, Inc.

PCI PA-DSS is an evolution of Visas Payment Application Best Practices (PABP), which was based on the Visa Cardholder Information Security Program (CISP). In addition to Visa CISP, PCI DSS combines American Express Data Security Operating Policy (DSOP), Discover Networks Information Security and Compliance (DISC), and MasterCards Site Data Protection (SDP) into a single comprehensive set of security standards. The transition to PCI PA-DSS was announced in April 2008. In early October 2008, PCI PA-DSS Version 1.2 was released to align with the PCI DSS Version 1.2, which was released on October 1, 2008. On January 1, 2011, PCI PA-DSS Version 2.0 was released. This extends the PCI DSS Version 1.2, which was released on October 1, 2008 and is effective as of January 1, 2011.

2.1 Applicability

The PCI PA-DSS applies to any payment application that stores, processes, or transmits cardholder data as part of authorization or settlement, unless the application would fall under the merchants PCI DSS validation. It is important to note that PA-DSS validated payment applications alone do not guarantee PCI DSS compliance for the merchant. The validated payment application must be implemented in a PCI DSS compliant environment. If your application runs on Windows XP, you are required to turn off Windows XP System Restore Points.

2.2 What Does PA-DSS Mean to You?

The following table provides opening points to cover in any discussion with merchants on data storage.

	Data Element	Storage Permitted	Protection Required	PCI DSS Req. 3, 4
Cardholder Data	Primary Account Number	Yes	Yes	Yes
	Cardholder Name [1]	Yes	Yes [1]	No
	Service Code [1]	Yes	Yes [1]	No
	Expiration Date [1]	Yes	Yes [1]	No
Sensitive Authentication Data [2]	Full Magnetic Stripe Data [3]	No	N/A	N/A
	CAV2/CID/CVC2/CVV2	No	N/A	N/A
	PIN/PIN Block	No	N/A	N/A

[1] These data elements must be protected if stored in conjunction with the PAN. This protection should be per PCI DSS requirements for general protection of the cardholder environment. Additionally, other legislation (for example, related to consumer personal data protection, privacy, identity theft, or data security) may require specific protection of this data, or proper disclosure of a company's practices if consumer-related personal data is being collected during the course of business. PCI DSS, however, does not apply if PANs are not stored, processed, or transmitted.

[2] Do not store sensitive authentication data after authorization (even if encrypted).

[3] Full track data from the magnetic stripe, magnetic-stripe image on the chip, or elsewhere.

2.3 Third Party Applications

The end-to-end transaction process, beginning with entry into the third party application until the response from the payment engine is returned, must meet the same level of compliance. In order to claim the third party application is end-to-end compliant, the application would need to be submitted to a QSA for a full PA-DSS audit.

The end user and/or P.O.S. developer can integrate and be compliant in the processing portion of a payment transaction. A brief review (given below) of the PA-DSS environmental variables that impact the end user merchant can help the end user merchant obtain and/or maintain PA-DSS compliance. Environmental variables that could prevent passing an audit include without limitation issues involving a secure network connection(s), end user setup location security, users, logging and assigned rights. Remove all testing configurations, samples, and data prior to going into production on your application.

2.4 PA-DSS Guidelines

The following PA-DSS Guidelines are being provided by IDTech as a convenience to its customers. Customers should not rely on these PA-DSS Guidelines, but should instead always refer to the most recent PCI DSS Program Guide published by PCI SSC.

1. Sensitive Data Storage Guidelines

Do not retain full magnetic stripe, card validation code or value (CAV2, CID, CVC2, CVV2), or PIN block data. 1.1 Do not store sensitive authentication data after authorization (even if encrypted): Sensitive authentication data includes the data as cited in the following Requirements 1.1.1 through 1.1.3. PCI Data Security Standard Requirement 3.2

Note: By prohibiting storage of sensitive authentication data after authorization, the assumption is that the transaction has completed the authorization process and the customer has received the final transaction approval. After authorization has completed, this sensitive authentication data cannot be stored.

1.1.1 After authorization, do not store the full contents of any track from the magnetic stripe (located on the back

of a card, contained in a chip, or elsewhere). This data is alternatively called full track, track, track 1, track 2, and magnetic-stripe data.

In the normal course of business, the following data elements from the magnetic stripe may need to be retained:

- The accountholders name,
- Primary account number (PAN),
- Expiration date, and
- Service code
- To minimize risk, store only those data elements needed for business.

Note: See PCI DSS and PA-DSS Glossary of Terms, Abbreviations, and Acronyms for additional information. PCI Data Security Standard Requirement 3.2.1

1.1.2 After authorization, do not store the card-validation value or code (three-digit or four-digit number printed on the front or back of a payment card) used to verify card-not-present transactions. Note: See PCI DSS and P↩ A-DSS Glossary of Terms, Abbreviations, and Acronyms for additional information. PCI Data Security Standard Requirement 3.2.2

1.1.3 After authorization, do not store the personal identification number (PIN) or the encrypted PIN block.

Note: See PCI DSS and PA-DSS Glossary of Terms, Abbreviations, and Acronyms for additional information. PCI Data Security Standard Requirement 3.2.3

1.1.4 Securely delete any magnetic stripe data, card validation values or codes, and PINs or PIN block data stored by previous versions of the payment application, in accordance with industry-accepted standards for secure deletion, as defined, for example by the list of approved products maintained by the National Security Agency, or by other State or National standards or regulations. PCI Data Security Standard Requirement 3.2

Note: This requirement only applies if previous versions of the payment application stored sensitive authentication data.

1.1.5 Securely delete any sensitive authentication data (pre-authorization data) used for debugging or troubleshooting purposes from log files, debugging files, and other data sources received from customers, to ensure that magnetic stripe data, card validation codes or values, and PINs or PIN block data are not stored on software vendor systems. These data sources must be collected in limited amounts and only when necessary to resolve a problem, encrypted while stored, and deleted immediately after use. PCI Data Security Standard Requirement 3.2

2. Protect stored cardholder data

2.1 Software vendor must provide guidance to customers regarding purging of cardholder data after expiration of customer-defined retention period. PCI Data Security Standard Requirement 3.1

2.2 Mask PAN when displayed (the first six and last four digits are the maximum number of digits to be displayed).

Notes:

- This requirement does not apply to those employees and other parties with a legitimate business need to see full PAN;
- This requirement does not supersede stricter requirements in place for displays of cardholder datafor example, for point-of-sale (POS) receipts. PCI Data Security Standard Requirement 3.3

2.3 Render PAN, at a minimum, unreadable anywhere it is stored, (including data on portable digital media, backup media, and in logs) by using any of the following approaches:

- One-way hashes based on strong cryptography with associated key management processes and procedures
- Truncation

- Index tokens and pads (pads must be securely stored)
- Strong cryptography with associated key management processes and procedures. The MINIMUM account information that must be rendered unreadable is the PAN. PCI Data Security Standard Requirement 3.4

The PAN must be rendered unreadable anywhere it is stored, even outside the payment application. Note: Strong cryptography is defined in the PCI DSS and PA-DSS Glossary of Terms, Abbreviations, and Acronyms.

2.4 If disk encryption is used (rather than file- or column-level database encryption), logical access must be managed independently of native operating system access control mechanisms (for example, by not using local user account databases). Decryption keys must not be tied to user accounts. PCI Data Security Standard Requirement 3.4.2

2.5 Payment application must protect cryptographic keys used for encryption of cardholder data against disclosure and misuse. PCI Data Security Standard Requirement 3.5

2.6 Payment application must implement key management processes and procedures for cryptographic keys used for encryption of cardholder data. PCI Data Security Standard Requirement 3.6

2.7 Securely delete any cryptographic key material or cryptogram stored by previous versions of the payment application, in accordance with industry-accepted standards for secure deletion, as defined, for example the list of approved products maintained by the National Security Agency, or by other State or National standards or regulations. These are cryptographic keys used to encrypt or verify cardholder data. PCI Data Security Standard Requirement 3.6

Note: This requirement only applies if previous versions of the payment application used cryptographic key materials or cryptograms to encrypt cardholder data.

3. Provide secure authentication features

3.1 The payment application must support and enforce unique user IDs and secure authentication for all administrative access and for all access to cardholder data. Secure authentication must be enforced to all accounts, generated or managed by the application by the completion of installation and for subsequent changes after the "out of the box" installation (defined at PCI DSS Requirements 8.1, 8.2, and 8.5.88.5.15) for all administrative access and for all access to cardholder data. PCI Data Security Standard Requirements 8.1, 8.2, and 8.5.88.5.15

Note: These password controls are not intended to apply to employees who only have access to one card number at a time to facilitate a single transaction. These controls are applicable for access by employees with administrative capabilities, for access to servers with cardholder data, and for access controlled by the payment application. This requirement applies to the payment application and all associated tools used to view or access cardholder data. 3.1.10 If a payment application session has been idle for more than 15 minutes, the application requires the user to re-authenticate. PCI Data Security Standard Requirement 8.5.15.

3.2 Software vendors must provide guidance to customers that all access to PCs, servers, and databases with payment applications must require a unique user ID and secure authentication. PCI Data Security Standard Requirements 8.1 and 8.2

3.3 Render payment application passwords unreadable during transmission and storage, using strong cryptography based on approved standards

Note: Strong cryptography is defined in PCI DSS and PA-DSS Glossary of Terms, Abbreviations, and Acronyms. PCI Data Security Standard Requirement 8.4

4. Log payment application activity

4.1 At the completion of the installation process, the out of the box default installation of the payment application must log all user access (especially users with administrative privileges), and be able to link all activities to individual users. PCI Data Security Standard Requirement 10.1

4.2 Payment application must implement an automated audit trail to track and monitor access. PCI Data Security Standard Requirements 10.2 and 10.3

5. Develop secure payment applications

5.1 Develop all payment applications in accordance with PCI DSS (for example, secure authentication and logging) and based on industry best practices and incorporate information security throughout the software development life cycle. These processes must include the following: PCI Data Security Standard Requirement 6.3

5.1.1 Live PANS are not used for testing or development. PCI Data Security Standard Requirement 6.4.4.

- Validation of all input (to prevent cross-site scripting, injection flaws, malicious file execution, etc.)
- Validation of proper error handling
- Validation of secure cryptographic storage
- Validation of secure communications
- Validation of proper role-based access control (RBAC)

5.1.2 Separate development/test, and production environments

5.1.3 Removal of test data and accounts before production systems become active development. PCI Data Security Standard Requirement 6.4.4

5.1.4 Review of payment application code prior to release to customers after any significant change, to identify any potential coding vulnerability. Removal of custom payment application accounts, user IDs, and passwords before payment applications are released to customers

Note: This requirement for code reviews applies to all payment application components (both internal and public-facing web applications), as part of the system development life cycle required by PA-DSS Requirement 5.1 and PCI DSS Requirement 6.3. Code reviews can be conducted by knowledgeable internal personnel or third parties.

5.2 Develop all web payment applications (internal and external, and including web administrative access to product) based on secure coding guidelines such as the Open Web Application Security Project Guide. Cover prevention of common coding vulnerabilities in software development processes, to include:

- Injection flaws, with particular emphasis on SQL injection, Cross-site scripting (XSS) OS Command Injection, LDAP and Xpath injection flaws, as well as other injection flaws.
- Buffer Overflow.
- Insecure cryptographic storage.
- Insecure communications.
- Improper error handling.
- All HIGH vulnerabilities as identified in the vulnerability identification process at PA-DSS Requirement 7.1.
- Cross-site scripting (XSS)
- Improper access control such as insecure direct object references, failure to restrict URL access and directory traversal.
- Cross-site request forgery (CSRF)

Note: The vulnerabilities listed in PA-DSS Requirements 5.2.1 through 5.2.9 and in PCI DSS at 6.5.1 through 6.5.9 were current in the OWASP guide when PCI DSS v1.2 / PCI DSS v2.0 (01/01/10) were published. However, if and when the OWASP guide is updated, the current version must be used for these requirements.

5.3 Software vendor must follow change control procedures for all product software configuration changes. PCI Data Security Standard Requirement 6.4. 5.The procedures must include the following:

- Documentation of impact
- Management sign-off by appropriate parties
- Testing functionality to verify the new change(s) does not adversely impact the security of the system. Remove all testing configurations, samples, and data before finalizing the product for production.

- Back-out or product de-installation procedures

5.4 The payment application must not use or require use of unnecessary and insecure services and protocols (for example, NetBIOS, file-sharing, Telnet, unencrypted FTP must be secured via SSH, S-FTP, SSL, IPSec and other technology to implement end to end security). PCI Data Security Standard Requirement 2.2.2

6. Protect wireless transmissions

6.1 For payment applications using wireless technology, the wireless technology must be implemented securely. Payment applications using wireless technology must facilitate use of industry best practices (for example, IE↩ EE 802.11i) to implement strong encryption for authentication and transmission. Controls must be in place to protect the implemented wireless network from unknown wireless access points and clients. This includes testing the end users wireless deployment on a quarterly basis to detect unauthorized access points within the system. Change wireless vendor defaults, including but not limited to default wireless encryption keys, passwords, and S↩ NMP community strings. Maintain a detailed updated hardware list. The end to end wireless implementation must be end to end secure. The use of WEP as a security control was prohibited as of 30 June 2010. PCI Data Security Standard Requirements 1.2.3, 2.1.1, 4.1.1, 6.2, 11.1a-e and 11.4a-c.

7. Test payment applications to address vulnerabilities

7.1 Software vendors must establish a process to identify newly discovered security vulnerabilities (for example, subscribe to alert services freely available on the Internet) and to test their payment applications for vulnerabilities. Any underlying software or systems that are provided with or required by the payment application (for example, web servers, third-party libraries and programs) must be included in this process. Remove all test configurations, samples, and data after testing and before promoting the changes to production. PCI Data Security Standard Requirement 6.2

7.2 Software vendors must establish a process for timely development and deployment of security patches and upgrades, which includes delivery of updates and patches in a secure manner with a known chain-of-trust, and maintenance of the integrity of patch and update code during delivery and deployment.

8. Facilitate secure network implementation

8.1 The payment application must be able to be implemented into a secure network environment. Application must not interfere with use of devices, applications, or configurations required for PCI DSS compliance (for example, payment application cannot interfere with anti-virus protection, firewall configurations, or any other device, application, or configuration required for PCI DSS compliance). PCI Data Security Standard Requirements 1, 3, 4, 5, and 6.

9. Cardholder data must never be stored on a server connected to the Internet

9.1 The payment application must be developed such that the database server and web server are not required to be on the same server, nor is the database server required to be in the DMZ with the web server. PCI Data Security Standard Requirement 1.3.7

10. Facilitate secure remote software updates

10.1 If payment application updates are delivered securely via remote access into customers systems, software vendors must tell customers to turn on remote-access technologies only when needed for downloads from vendor

and to turn off immediately after download completes. Alternatively, if delivered via VPN or other high-speed connection, software vendors must advise customers to properly configure a firewall or a personal firewall product to secure authentication using a two factor authentication mechanism. PCI Data Security Standard Requirement 8.3

10.2 If payment application may be accessed remotely, remote access to the payment application must be authenticated using a two factor authentication mechanism. PCI Data Security Standard Requirement 8.3

10.3 Any remote access into the payment application must be done securely. If vendors, resellers/integrators, or customers can access customers payment applications remotely, the remote access must be implemented securely. PCI Data Security Standard Requirements 1, 8.3 and 12.3.9

11. Encrypt sensitive traffic over public networks

11.1 If the payment application sends, or facilitates sending, cardholder data over public networks, the payment application must support use of strong cryptography and security protocols such as SSL/TLS and Internet protocol security (IPSEC) to safeguard sensitive cardholder data during transmission over open, public networks. Examples of open, public networks that are in scope of the PCI DSS are: The Internet Wireless technologies Global System for Mobile Communications (GSM) General Packet Radio Service (GPRS) PCI Data Security Standard Requirement 4.1

11.2 The payment application must never send unencrypted PANs by end-user messaging technologies (for example, e-mail, instant messaging, and chat). PCI Data Security Standard Requirement 4.2

12. Encrypt all non-console administrative access

12.1 Instruct customers to encrypt all non-console administrative access using technologies such as SSH, VPN, or SSL/TLS for web-based management and other non-console administrative access. Telnet or remote login must never be used for administrative access. PCI Data Security Standard Requirement 2.3

13. Maintain instructional documentation and training programs for customers, resellers, and integrators

13.1 Develop, maintain, and disseminate a PA-DSS Implementation Guide(s) for customers, resellers, and integrators that accomplishes the following:

- Addresses all requirements in this document wherever the PA-DSS Implementation Guide is referenced.
- Includes a review at least annually and updates to keep the documentation current with all major and minor software changes as well as with changes to the requirements in this document.

13.2 Develop and implement training and communication programs to ensure payment application resellers and integrators know how to implement the payment application and related systems and networks according to the PA-DSS Implementation Guide and in a PCI DSS-compliant manner.

- Update the training materials on an annual basis and whenever new payment application versions are released.

2.5 More Information

IDTech Systems, Inc. highly recommends that merchants contact the card association(s) or their processing company and find out exactly what they mandate and/or recommend. Doing so may help merchants protect themselves from fines and fraud.

For more information related to security, visit:

- http://www.pcisecuritystandards.org
- http://www.visa.com/cisp
- http://www.sans.org/resources
- http://www.microsoft.com/security/default.asp
- https://sdp.mastercardintl.com/
- http://www.americanexpress.com/merchantspecs

CAPN questions: capninfocenter@aexp.com

Chapter 3

Sending Direct Commands

The main purpose of IDTech .dll for Kiosk III/IV is to expedite integration to the device by providing the connectivity and communication protocols. It also provides the main functions to get device info, perform Tap transactions, and to modify terminal configuration data files.

The Kiosk III/IV has an extensive and powerful command set based on the NEO platform. A NEO command consists of a Command, a Sub Command, and optionally data. To access these commands, please reference the NEO/IDG Command document included as a separate item in the SDK. Please note that Protocol 1 commands have been deprecated, and any existing Protocol 1 commands relevant to Kiosk III/IV can be accomplished by a Protocol 2 command. The library uses the following the command to send Protocol 2 commands to Kiosk III/IV:

IDTechSDK::IDT_KioskIII::device_sendVivoCommandP2()

Any function not supported by the SDK can be sent with the sendIDGCommand.

Chapter 4

Main Transaction Commands

The methods below are provided as a reference to the main commands needed to execute Contactless transaction.

4.1 Contactless Methods

Start CTLS Transaction

IDTechSDK::IDT_KioskIII::ctls_startTransaction()

Begins an amount authorization request with the card. Returns decision/tags in callback method. CTLS captured Track 1 data is returned in tag FFEE13, and CTLS captured Track 2 data is returned in tag FFEE14.

Cancel CTLS Transaction

IDTechSDK::IDT_KioskIII::ctls_cancelTransaction()

Powers down the CTLS antenna.

Auto Poll / Burst Mode

IDTechSDK::IDT_KioskIII::device_setBurstMode() IDTechSDK::IDT_KioskIII::device_setPollMode()

Setting the device to Auto Poll Mode will automatically capture taps. Setting the device to Burst Mode will automatically return data for processing.

Terminal Configuration

IDTechSDK::IDT_KioskIII::ctls_retrieveTerminalData()
IDTechSDK::IDT_KioskIII::ctls_setTerminalData()

Methods for terminal configuration. When setting the terminal data, you pass the tags in TLV format.

AID Management

IDTechSDK::IDT_KioskIII::ctls_retrieveApplicationData:response()
IDTechSDK::IDT_KioskIII::ctls_removeApplicationData()
IDTechSDK::IDT_KioskIII::ctls_removeAllApplicationData()
IDTechSDK::IDT_KioskIII::ctls_setApplicationData()
IDTechSDK::IDT_KioskIII::ctls_retrieveAIDList()

Methods for AID management. When setting the AID, you pass tags in TLV format. When retrieving AID, you can receive the results as tags in TLV format. When retrieving the AID list, the list of AID Names/length can be retrieved from the 2 dimensional byte array

CAPK Management

IDTechSDK::IDT_KioskIII::ctls_retrieveCAPK()
IDTechSDK::IDT_KioskIII::ctls_removeCAPK()
IDTechSDK::IDT_KioskIII::ctls_removeAllCAPK()
IDTechSDK::IDT_KioskIII::ctls_setCAPK()
IDTechSDK::IDT_KioskIII::ctls_retrieveCAPKList()

Methods for Certificate Authority Public Key management. When setting the CAPK, you populate and pass the key as a sequence of ordered bytes. When specifying a CAPK to retrieve or remove, you populate the name in the byte[] parameter. When retrieving the CAPK list, the list of RID/Index can be retrieved from the ordered byte stream, 6 bytes each, bytes 1-5 RID, byte 6 index

Chapter 5

Connecting to Kiosk III/IV

The Kiosk III/IV connects through either USB or Serial Interface (COM).

5.1 Connect with USB:

The Kiosk III/IV uses USB-HID (Human Interface Device) and does not need any vendor-specific drivers. Simply by plugging into an available USB port makes it available for SDK connectivity. To inform the SDK you are using the USB interface of the Kiosk III/IV, you would execute the following command during program initialization to establish a connection:

```
IDT_KioskIII.useUSB();
```

5.2 Connect with Serial Interface (COM)

The Kiosk III/IV can connect via Serial Interface. The serial port settings are as follows:

- Speed: 19,200
- Bits: 8
- Stop Bit: 1
- Parity: None

To inform the SDK you are using the Serial Interface of the Kiosk III, you would execute the following command during program initialization to establish a connection by passing the correct COM port number. In the following example, we are connecting to COM1 SRED device:

```
IDT_KioskIII.useSerialPort(1, true);
```

Chapter 6

Core Implementation: WinForms

IDTechSDK.dll includes class libraries to interface with the Kiosk III/IV. This guide assume a fair understanding of Visual Studio 2013+, C# and general Windows programming knowledge.

6.1 Integrating with IDTechSDK.dll

- Import the necessary libraries
- Add using statements to utilize library
- Implement the callback function
- Initialize Kiosk III/IV
- Sample Project Tutorial

6.2 Import the necessary libraries

Communicating with IDTech Devices requires the following library to be referenced by the project:

- IDTechSDK.dll

Add the reference as you would any .NET managed library reference. The most direct method would be right-click on the "References" in the Solution Explorer for the project, select "Add Reference...", click "Browse..." and locate IDTechSDK.dll.

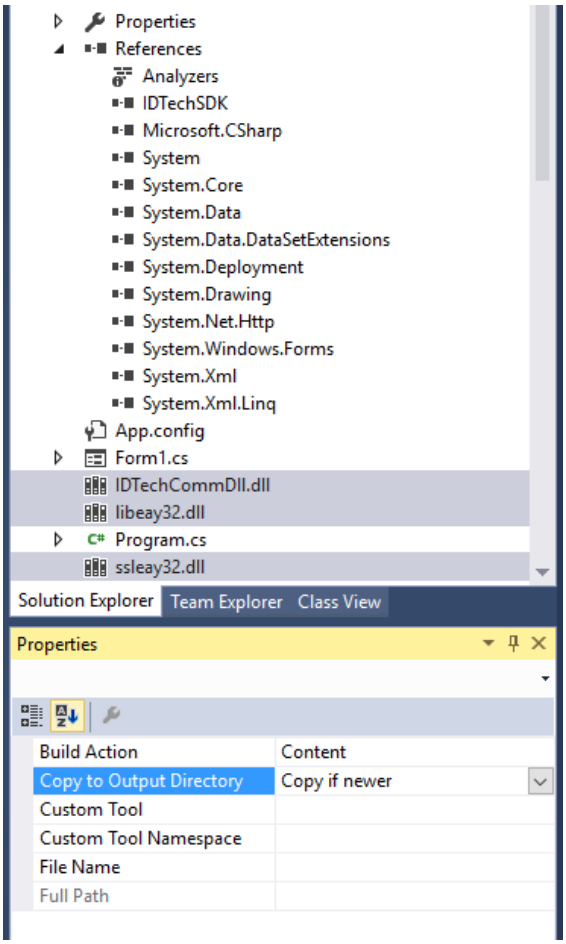
IDTechSDK.dll has a dependency of Microsoft .NET 4.50 or greater. Please make sure your final application installer checks for this dependency and installs it if not on the destination system.

In addition, the following libraries need to be added to project folder and included in with the application distribution:

- KioskIII_config.dll
- KioskIII_ctls.dll
- KioskIII_device.dll
- KioskIII_parse.dll

You can add these libraries by right-clicking on your project name in the solution Explorer and select "Add->Existing Item..." or keyboard shortcut Shift-Alt-A.

Once all three items are added, set their properties to "Copy if newer" so they will be included in the final applications destination folder.



6.3 Add using statements to utilize library

Add a line of code to the class that will utilize IDTechSDK.dll at the start of the file:

```
using IDTechSDK;
```

6.4 Implement the callback function

There is a single method that will receive all callback information from the SDK. It uses DeviceState to determine which action to take.

```
private void MessageCallBack(IDTechSDK.IDT_DEVICE_Types type, DeviceState state, byte[] data,
      IDTTransactionData cardData, EMV_Callback emvCallback, RETURN_CODE transactionResultCode)
{
    switch (state)
    {

        case DeviceState.ToConnect:
            //A connection attempt is starting for IDT_DEVICE_TYPES type
            break;
        case DeviceState.DefaultDeviceTypeChange:
            //The SDK is changing the default device to IDT_DEVICE_TYPES type
            break;
        case DeviceState.Connected:
            //A connection has been made to IDT_DEVICE_TYPES type
            break;
```

```
        case DeviceState.Disconnected:
            //A disconnection has occured with IDT_DEVICE_TYPES type
            break;
        case DeviceState.ConnectionFailed:
            //A connection attempt has failed for IDT_DEVICE_TYPES type
            break;
        case DeviceState.TransactionData:
            //Transaction data is beign returned in IDTTransactionData cardData
            break;
        case DeviceState.DataReceived:
            //Low-level data received  for IDT_DEVICE_TYPES type
            break;
        case DeviceState.DataSent:
            //Low-level data sent  for IDT_DEVICE_TYPES type
            break;
        case DeviceState.CommandTimeout:
            //Command timeout has occurred for IDT_DEVICE_TYPES type
            break;
        case DeviceState.ToSwipe:
            //Awaiting a swipe for IDT_DEVICE_TYPES type
            break;
        case DeviceState.MSRDecodeError:
            //Awaiting a swipe for IDT_DEVICE_TYPES type
            break;
        case DeviceState.ToTap:
            //Awaiting a contactless tap for IDT_DEVICE_TYPES type
            break;
        case DeviceState.SwipeTimeout:
            //Waiting for swipe timed out
            break;
        case DeviceState.TransactionCancelled:
            //Transaction has been cancelled
            break;
        case DeviceState.DeviceTimeout:
            //Waiting for transaction to complete timed out
            break;
        case DeviceState.TransactionFailed:
            //Transaction failed to complete
            break;
        case DeviceState.EMVCallback:
            //Callback during EMV transaction retrieved from EMV_Callback emvCallback
            break;
        default:
            break;

    }


}
```

6.5 Initialize Kiosk III/IV:

A Singleton instance has been established in the IDT_KioskIII class. Establish the callback, and then set a connection to Kiosk III/IV through either USB or Serial.

```
public KioskIII_Simple_Demo()
{
    InitializeComponent();
    IDT_KioskIII.setCallback(MessageCallBack);

    //USB Connection:
    IDT_KioskIII.useUSB();

    //Serial Connection SRED:
    //IDT_KioskIII.useSerialPort(1, true);
}
```

6.6 Sample Project Tutorial

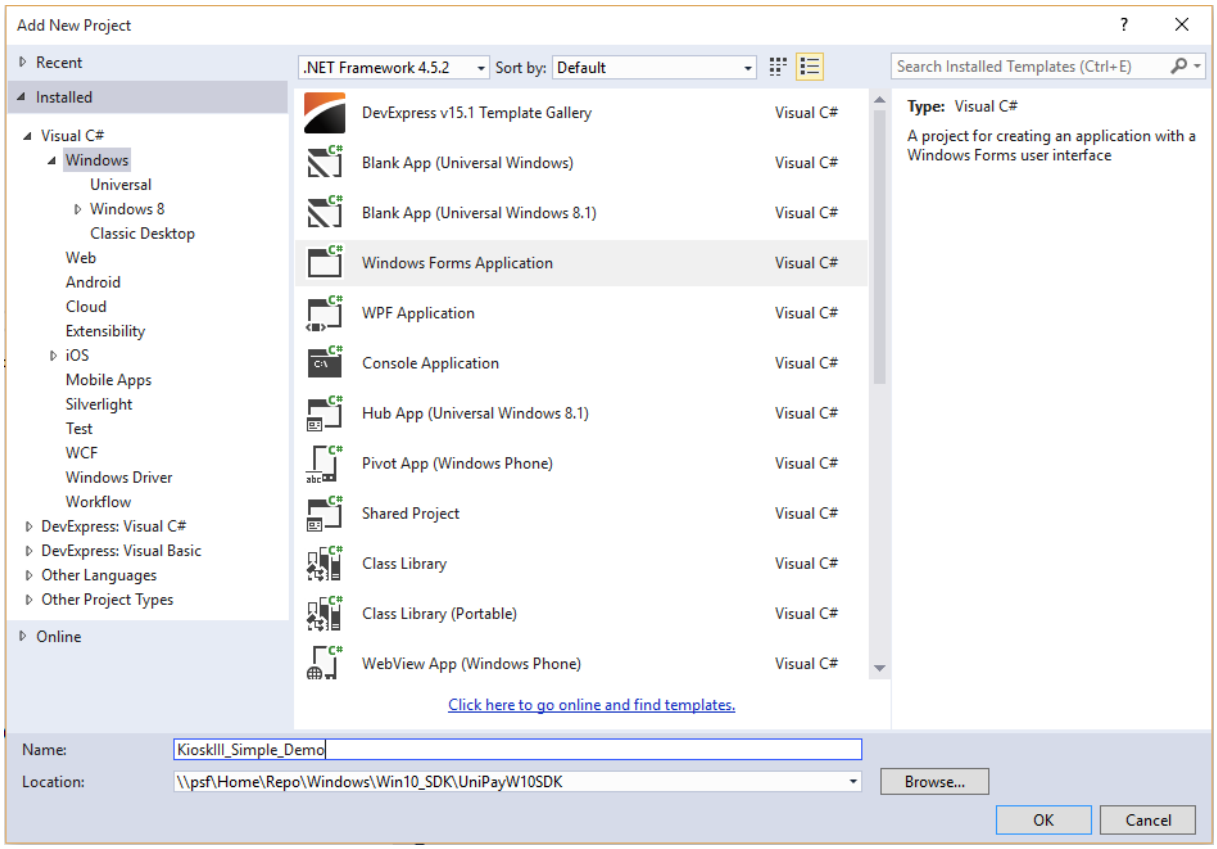
Using Visual Studio 2015, we will create a C# sample project that will interface with the Kiosk III and will perform the following activities:

- Get firmware version
- Perform CTLS Transaction

- Connect to either USB or Serial Versions of Kiosk III/IV.
- Show log of all data going to/from Kiosk III/IV

6.6.1 Step 1: Create New Project

Create a new Windows Form Application in Visual Studio. In our example, we use project name KioskIII_Simple_↩
Demo



6.6.2 Step 2: Import Libraries

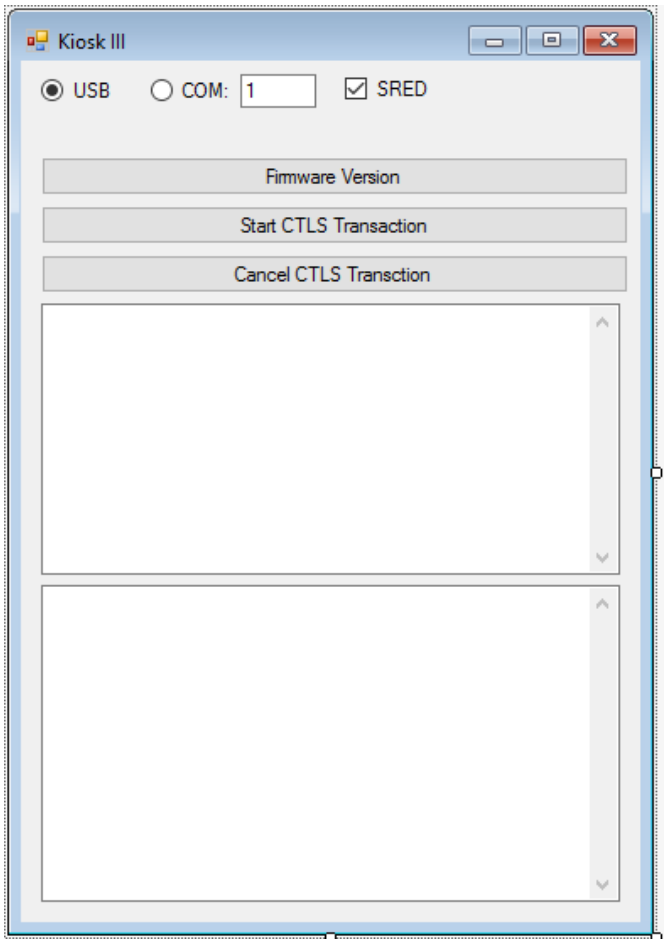
Import the necessary libraries

6.6.3 Step 3: Design Interface

Use the Form Designer to layout buttons/fields
Open your form designer and add items so it contains the following buttons/fields:

- Radio buttons for USB and COM selection. For COM selection, add a text box to specify COM port
- Add buttons to execute the following functions:
  - Get Firmware
  - Start ICC Transaction
  - Cancel ICC Transaction
- Add a text box to communicate data from the Kiosk III/IV.

- Add a text box for the log of Kiosk III/IV.



6.6.4 Step 4: Configure the project file

In the start of the class file, perform the following:

- Add using statements to utilize library
- set callback method and initialize KioskIII singleton object in the class initializer. Reference: Initialize Kiosk III/IV
- Implement the radio button methods CheckChanged handlers to set the proper interface for SDK communications

```
private void rbUSB_CheckedChanged(object sender, EventArgs e)
{
    if (rbUSB.Checked)
    {
        IDT_KioskIII.useUSB();
        return;
    }
    IDT_KioskIII.useSerialPort(Convert.ToInt32(tbCOMPort.Text), cbSRED.Checked);
}
```

- Implement the button methods Click handlers for button press code execution

```
private void btnFirmwareVersion_Click(object sender, EventArgs e)
{
    string firmwareVersion = "";
    byte[] reData = { };

    RETURN_CODE rt = IDT_KioskIII.SharedController.device_getFirmwareVersion(ref firmwareVersion);
```

```
        if (rt == RETURN_CODE.RETURN_CODE_DO_SUCCESS)
        {
            tbOutput.AppendText("Firmware Ver: " + firmwareVersion + "\r\n");
        }
        else
        {
            tbOutput.AppendText("Get Firmware Fail Error Code: " + "0x" + String.Format("{0:X}", (ushort)rt) +
          ": " + IDTechSDK.errorCode.getErrorString(rt) + "\r\n");
        }
    }


    private void btnStartEMVTransaction_Click(object sender, EventArgs e)
    {
        byte[] additionalTags = new byte[] { 0x8E, 0x5A };
        RETURN_CODE rt = IDT_KioskIII.SharedController.ctls_startTransaction(1.00, 0, 2,0, 30, additionalTags);
        if (rt == RETURN_CODE.RETURN_CODE_DO_SUCCESS)
        {
            tbOutput.AppendText("Start CTLS Successful\r\n");
        }
        else
        {
            tbOutput.AppendText("Start CTLS failed Error Code: " + "0x" + String.Format("{0:X}", (ushort)rt) +
          ": " + IDTechSDK.errorCode.getErrorString(rt) + "\r\n");
        }
    }

    private void btnCancelEMVTransaction_Click(object sender, EventArgs e)
    {
        RETURN_CODE rt = IDT_KioskIII.SharedController.ctls_cancelTransaction();
        if (rt == RETURN_CODE.RETURN_CODE_DO_SUCCESS)
        {
            tbOutput.AppendText("Cancel Transaction Successful\r\n");
            System.Diagnostics.Debug.WriteLine("Cancel Transaction Successful");
        }
        else
        {
            tbOutput.AppendText("Cancel Transaction failed Error Code: " + "0x" + String.Format("{0:X}", (
          ushort)rt) + ": " + IDTechSDK.errorCode.getErrorString(rt) + "\r\n");
            System.Diagnostics.Debug.WriteLine("Cancel Transaction failed Error Code: " + "0x" + String.
          Format("{0:X}", (ushort)rt));
        }
    }
```

6.6.5 Step 5: Configure callback to receive important SDK data (messages,log info and transaction results)

```
private void MessageCallBack(IDTechSDK.IDT_DEVICE_Types type, DeviceState state, byte[] data,
      IDTTransactionData cardData, EMV_Callback emvCallback, RETURN_CODE transactionResultCode)
 {
     switch (state)
     {

         case DeviceState.ToConnect:
             //A connection attempt is starting for IDT_DEVICE_TYPES type
             SetOutputText("Callback:ToConnect\n");
             break;
         case DeviceState.DefaultDeviceTypeChange:
             //The SDK is changing the default device to IDT_DEVICE_TYPES type
             SetOutputText("Callback:DefaultDeviceTypeChange\n");
             break;
         case DeviceState.Connected:
             //A connection has been made to IDT_DEVICE_TYPES type
             SetOutputText("Callback:Connected\n");
             break;
         case DeviceState.Disconnected:
             //A disconnection has occured with IDT_DEVICE_TYPES type
             SetOutputText("Callback:Disconnected\n");
             break;
         case DeviceState.ConnectionFailed:
             //A connection attempt has failed for IDT_DEVICE_TYPES type
             SetOutputText("Callback:ConnectionFailed\n");
             break;
         case DeviceState.TransactionData:
             //Transaction data is beign returned in IDTTransactionData cardData
             SetOutputText("Callback:TransactionData\n");
             displayCardData(cardData);
             break;
         case DeviceState.DataReceived:
             //Low-level data received  for IDT_DEVICE_TYPES type
             SetOutputTextLog( " IN: " + Common.getHexStringFromBytes(data));
             break;
         case DeviceState.DataSent:
             //Low-level data sent  for IDT_DEVICE_TYPES type
             SetOutputTextLog(" OUT: " + Common.getHexStringFromBytes(data));
```

```
            break;
        case DeviceState.CommandTimeout:
            SetOutputText("Callback:CommandTimeout\n");
            //Command timeout has occurred for IDT_DEVICE_TYPES type
            break;
        case DeviceState.ToSwipe:
            //Awaiting a swipe for IDT_DEVICE_TYPES type
            SetOutputText("Callback:ToSwipe\n");
            break;
        case DeviceState.MSRDecodeError:
            //Awaiting a swipe for IDT_DEVICE_TYPES type
            SetOutputText("Callback:MSRDecodeError\n");
            break;
        case DeviceState.ToTap:
            //Awaiting a contactless tap for IDT_DEVICE_TYPES type
            SetOutputText("Callback:ToTap\n");
            break;
        case DeviceState.SwipeTimeout:
            //Waiting for swipe timed out
            SetOutputText("Callback:SwipeTimeout\n");
            break;
        case DeviceState.TransactionCancelled:
            //Transaction has been cancelled
            SetOutputText("Callback:TransactionCancelled\n");
            break;
        case DeviceState.EMVTimeout:
            //Waiting for EMV transaction to complete timed out
            SetOutputText("Callback:EMVTimeout\n");
            break;
        case DeviceState.TransactionFailed:
            //Transaction failed to complete
            SetOutputText("Callback:TransactionFailed\n");
            break;
        case DeviceState.EMVCallback:
            //Callback during EMV transaction retrieved from EMV_Callback emvCallback
            SetOutputText("Callback:EMVCallback\n");
            break;
        default:
            break;

    }


}



private void displayCardData(IDTTransactionData cardData)
{
    string text = "";



    if (cardData.Event == EVENT_TRANSACTION_DATA_Types.EVENT_TRANSACTION_DATA_CARD_DATA)
    {
        SetOutputText("Data received: (Length [" + cardData.msr_rawData.Length.ToString() + "])\n" +
     string.Concat(cardData.msr_rawData.ToArray().Select(b => b.ToString("X2")).ToArray()) + "\r\n");
        System.Diagnostics.Debug.WriteLine("Data received: (Length [" + cardData.msr_rawData.Length.
     ToString() + "])\n" + string.Concat(cardData.msr_rawData.ToArray().Select(b => b.ToString("X2")).ToArray())
     ;

    }

    if (cardData.device_RSN != null && cardData.device_RSN.Length > 0)
        text += "Serial Number: " + cardData.device_RSN + "\r\n";

    if (cardData.msr_track1Length > 0)
        text += "Track 1: " + cardData.msr_track1 + "\r\n";

    if (cardData.msr_encTrack1 != null)
        text += "Track 1 Encrypted: " + Common.getHexStringFromBytes(cardData.msr_encTrack1) + "\r\n";



    if (cardData.msr_hashTrack1 != null)
        text += "Track 1 Hash: " + Common.getHexStringFromBytes(cardData.msr_hashTrack1) + "\r\n";




    if (cardData.msr_track2Length > 0)
        text += "Track 2: " + cardData.msr_track2 + "\r\n";


    if (cardData.msr_encTrack2 != null)
        text += "Track 2 Encrypted: " + Common.getHexStringFromBytes(cardData.msr_encTrack2) + "\r\n";
```

```
if (cardData.msr_hashTrack2 != null)
    text += "Track 2 Hash: " + Common.getHexStringFromBytes(cardData.msr_hashTrack2) + "\r\n";



if (cardData.msr_track3Length > 0)
    text += "Track 3: " + cardData.msr_track3 + "\r\n";


if (cardData.msr_encTrack3 != null)
    text += "Track 3 Encrypted: " + Common.getHexStringFromBytes(cardData.msr_encTrack3) + "\r\n";


if (cardData.msr_hashTrack3 != null)
    text += "Track 3 Hash: " + Common.getHexStringFromBytes(cardData.msr_hashTrack3) + "\r\n";

if (cardData.msr_KSN != null)
    text += "KSN: " + Common.getHexStringFromBytes(cardData.msr_KSN) + "\r\n";


if (cardData.emv_clearingRecord != null)
{
    if (cardData.emv_clearingRecord.Length > 0)
    {
        text += "\r\nCTLS Clearing Record: \r\n";
        text += Common.getHexStringFromBytes(cardData.emv_clearingRecord) + "\r\n";
        Dictionary<string, string> dict = Common.processTLVUnencrypted(cardData.emv_clearingRecord);
        foreach (KeyValuePair<string, string> kvp in dict) text += kvp.Key + ": " + kvp.Value + "\r\n"
 ;
        text += "\r\n\r\n";

    }

}
if (cardData.emv_unencryptedTags != null)
{
    if (cardData.emv_unencryptedTags.Length > 0)
    {
        text += "\r\n========================= \r\n";

        text += "\r\nUnencrypted Tags: \r\n";
        text += Common.getHexStringFromBytes(cardData.emv_unencryptedTags) + "\r\n\r\n";
        text += tlvToValues(cardData.emv_unencryptedTags);
        text += "\r\n========================= \r\n";
    }

}
if (cardData.emv_encryptedTags != null)
{
    if (cardData.emv_encryptedTags.Length > 0)
    {
        text += "\r\n========================= \r\n";
        text += "\r\nEncrypted Tags: \r\n";
        text += Common.getHexStringFromBytes(cardData.emv_encryptedTags) + "\r\n\r\n";
        text += tlvToValues(cardData.emv_encryptedTags);
        text += "\r\n========================= \r\n";

    }

}
if (cardData.emv_maskedTags != null)
{
    if (cardData.emv_maskedTags.Length > 0)
    {
        text += "\r\n========================= \r\n";
        text += "\r\nMasked Tags: \r\n";
        text += Common.getHexStringFromBytes(cardData.emv_maskedTags) + "\r\n\r\n";
        text += tlvToValues(cardData.emv_maskedTags);
        text += "\r\n========================= \r\n";

    }

}

text += "ICC Present: ";
text += (cardData.iccPresent ? "TRUE" : "FALSE") + "\r\n";
text += "is CTLS: ";
text += (cardData.isCTLS ? "TRUE" : "FALSE") + "\r\n";


if (cardData.Event == EVENT_TRANSACTION_DATA_Types.EVENT_TRANSACTION_DATA_EMV_DATA)
{
    if (!cardData.isCTLS) text += "Capture Encrypt Type: " + ((cardData.emv_encryptionMode ==
 EMV_ENCRYPTION_MODE.EMV_ENCRYPTION_MODE_TDES) ? "TDES" : "AES") + "\r\n";
    switch (cardData.emv_resultCode)
```

```
        {
            case EMV_RESULT_CODE.EMV_RESULT_CODE_APPROVED:
                text += ("EMV RESULT: " + "EMV_RESULT_CODE_APPROVED" + "\r\n");
                break;
            case EMV_RESULT_CODE.EMV_RESULT_CODE_APPROVED_OFFLINE:
                text += ("EMV RESULT: " + "EMV_RESULT_CODE_APPROVED_OFFLINE" + "\r\n");
                break;
            case EMV_RESULT_CODE.EMV_RESULT_CODE_DECLINED_OFFLINE:
                text += ("EMV RESULT: " + "EMV_RESULT_CODE_DECLINED_OFFLINE" + "\r\n");
                break;
            case EMV_RESULT_CODE.EMV_RESULT_CODE_DECLINED:
                text += ("EMV RESULT: " + "EMV_RESULT_CODE_DECLINED" + "\r\n");
                break;
            case EMV_RESULT_CODE.EMV_RESULT_CODE_GO_ONLINE:
                text += ("EMV RESULT: " + "EMV_RESULT_CODE_GO_ONLINE" + "\r\n");
                break;
            case EMV_RESULT_CODE.EMV_RESULT_CODE_CALL_YOUR_BANK:
                text += ("EMV RESULT: " + "EMV_RESULT_CODE_CALL_YOUR_BANK" + "\r\n");
                break;
            case EMV_RESULT_CODE.EMV_RESULT_CODE_NOT_ACCEPTED:
                text += ("EMV RESULT: " + "EMV_RESULT_CODE_NOT_ACCEPTED" + "\r\n");
                break;
            case EMV_RESULT_CODE.EMV_RESULT_CODE_FALLBACK_TO_MSR:
                text += ("EMV RESULT: " + "EMV_RESULT_CODE_FALLBACK_TO_MSR" + "\r\n");
                break;
            case EMV_RESULT_CODE.EMV_RESULT_CODE_TIMEOUT:
                text += ("EMV RESULT: " + "EMV_RESULT_CODE_TIMEOUT" + "\r\n");
                break;
            case EMV_RESULT_CODE.EMV_RESULT_CODE_AUTHENTICATE_TRANSACTION:
                text += ("EMV RESULT: " + "EMV_RESULT_CODE_AUTHENTICATE_TRANSACTION" + "\r\n");
                break;
            case EMV_RESULT_CODE.EMV_RESULT_CODE_SWIPE_NON_ICC:
                text += ("EMV RESULT: " + "EMV_RESULT_CODE_SWIPE_NON_ICC" + "\r\n");
                break;
            case EMV_RESULT_CODE.EMV_RESULT_CODE_CTLS_TWO_CARDS:
                text += ("EMV RESULT: " + "EMV_RESULT_CODE_CTLS_TWO_CARDS" + "\r\n");
                break;
            case EMV_RESULT_CODE.EMV_RESULT_CODE_CTLS_TERMINATE:
                text += ("EMV RESULT: " + "EMV_RESULT_CODE_CTLS_TERMINATE" + "\r\n");
                break;
            case EMV_RESULT_CODE.EMV_RESULT_CODE_CTLS_TERMINATE_TRY_ANOTHER:
                text += ("EMV RESULT: " + "EMV_RESULT_CODE_CTLS_TERMINATE_TRY_ANOTHER" + "\r\n");
                break;
        }
    }



    SetOutputText(text);
}

private string tlvToValues(byte[] tlv)
{
    string text = "";
    Dictionary<string, string> dict = Common.processTLVUnencrypted(tlv);
    foreach (KeyValuePair<string, string> kvp in dict) text += kvp.Key + ": " + kvp.Value + "\r\n";
    return text;
}

delegate void SetTextCallback(string text);

private void SetOutputText(string text)
{
    // InvokeRequired required compares the thread ID of the
    // calling thread to the thread ID of the creating thread.
    // If these threads are different, it returns true.
    if (tbOutput.InvokeRequired)
    {
        SetTextCallback d = new SetTextCallback(SetOutputText);
        Invoke(d, new object[] { text });
    }
    else
    {
        try { tbOutput.AppendText(text + "\r\n"); } catch (Exception ex) { }

    }
}

private void SetOutputTextLog(string text)
{
    // InvokeRequired required compares the thread ID of the
    // calling thread to the thread ID of the creating thread.
    // If these threads are different, it returns true.
    if (logOutput.InvokeRequired)
    {
        SetTextCallback d = new SetTextCallback(SetOutputTextLog);
```

```
            Invoke(d, new object[] { text });
        }
        else
        {
            try { logOutput.AppendText(text + "\r\n"); }
            catch (Exception ex)
            {
                System.Diagnostics.Debug.WriteLine("Exception: " + ex);
            }

        }
    }
```

Chapter 7

LCD Foreign Language Mapping Table

ID	Message ID	English	French	Spanish	Chinese
0	MSG_NULL	-	-	-	-
1	MSG_AMOUNT	AMOUNT	MONTANT	CANTIDAD	金
2	MSG_AMOUNT_↩ OK	AMOUNT OK?	MONTANT OK	MONTO CORRE↩ CTO?	确定金
3	MSG_APPROVED	APPROVED	APPROUVE	APROVADO	通
4	MSG_CALL_YO↩ UR_BANK	CALL YOUR BANK	APPE VOTRE B↩ ANQE	LLAME A SU BA↩ NCO	系您的行
5	MSG_CANCEL_↩ OR_ENTER	CANCEL OR EN↩ TER	ANNULE OU EN↩ TRER	CANCEL O ENT↩ RAR	取消或确定
6	MSG_CARD_ER↩ ROR	CARD ERROR	ERREUR CARTE	ERROR DE TAR↩ JETA	卡
7	MSG_DECLINED	DECLINED	REFUSE	DECLINADO	卡被拒
8	MSG_ENTER_A↩ MOUNT	ENTER AMOUNT	ENTRER MONT↩ ANT	INGRESE MONTO	入金
9	MSG_ENTER_PIN	ENTER PIN:	ENTRER PIN:	ENTRAR NPI:	入密
10	MSG_INCORRE↩ CT_PIN	INCORRECT PIN	NIP INCORRECT	NPI INCORRECTO	密
11	MSG_ICC_MSR1	SWIPE OR INSE↩ RT	PASSER OU INS↩ ERT	MOVER O INSERT	刷卡或插卡
12	MSG_ICC_MSR2	CARD	CARTE	TARJETA	卡
13	MSG_INSERT_↩ CARD	INSERT CARD	INSERT CARTE	INSERTAR TAR↩ JETA	插卡
14	MSG_USE_CHI↩ P_READER	USE CHIP READ↩ ER UTI	LECTEUR CHIP	USO CHIP LECT↩ OR	使用芯片卡
15	MSG_NOT_ACC↩ EPTED	NOT ACCEPTED	PAS ACCEPTE	DENEGADO	法接受
16	MSG_PIN_OK	GET PIN OK			密正确
17	MSG_PLEASE_↩ WAIT	PLEASE WAIT...	ATTENDRE...	POR FAVOR ES↩ PERE	等候中
18	MSG_PROCESS↩ ING_ERROR	PROCESSING E↩ RROR	ERREUR DE TR↩ AITE	ERROR PROCE↩ SANDO	理
19	MSG_USE_MAG↩ STRIPE	USE MAGSTRIPE	USAGE MAGST↩ RIPE	USO DE MAGST↩ RIPE	使用磁卡
20	MSG_TRY_AGAIN	TRY AGAIN	REESSAYER	VUELV INTENTA↩ RLO	重
21	MSG_ONLINE	GO ONLINE	GO LIGNE	GO LINEA	在

ID	Message ID	English	French	Spanish	Chinese
22	MSG_TRANSAC↩ TION_ERROR	TRANSACTION ERR	ERREUR DE TR↩ ANS	ERROR DE TRA↩ NSAC	交易
23	MSG_TERMINATE	TERMINATE	RESILIER	TERMINAR	止
24	MSG_ADVICE	ADVICE	CONSEILS	CONSEJOS	建
25	MSG_TIMEOUT	TIME OUT	TIMEOUT	TIEMPO DE ESP↩ ERA	超
26	MSG_PROCESS↩ ING	PROCESSING...	PROCESSUS...	PROCESANDO...	理中。。。
27	MSG_PIN_TRY_↩ EX	PIN TRY LIMIT EX	PIN TRY DEPAS↩ SE	TRY PIN SUPER↩ ADA	密次多
28	MSG_ISSUER_↩ AUTH_FAIL	ISSUER AUTH F↩ AIL	EMETTEUR FAIL	EMISOR FALLA	与卡机构
29	MSG_CONTINU↩ E_PROCESS	CONTINUE PRO↩ CESS	CONTINUER LA	CONTINUAR PR↩ OCES	理
30	MSG_GET_PIN_↩ ERROR	GET PIN ERROR	GET PIN ERROR	OBTENER PIN E↩ RR	密
31	MSG_GET_PIN_↩ FAIL	GET PIN FAIL	GET PIN FAIL	OBTENER PIN F↩ ALL	取密
32	MSG_NOKEY_G↩ ET_PIN	NO KEY GET PIN	NO KEY GET PIN	NO CLAVE GET PIN	法入密
33	MSG_CANCELL↩ ED	CANCELLED	ANNULE	CANCELADO	取消
34	MSG_LAST_PIN↩ _TRY	LAST PIN TRY	-	-	最后一次入密

Chapter 8

Error Code Reference

```
public static string getErrorString(RETURN_CODE code)
        {
            switch ((int)code)
            {
                case 0: return "no error, beginning task";
                case 1: return "no response from reader";
                case 2:return "invalid response data";
                case 3: return "time out for task or CMD";
                case 4:return "wrong parameter";
                case 5: return "SDK is doing MSR or ICC task";
                case 6: return "SDK is doing PINPad task";
                case 7:return "SDK is doing CTLS task";
                case 8:return "SDK is doing EMV task";
                case 9: return "SDK is doing Other task";
                case 10:return "err response or data";
                case 11: return "no reader attached";
                case 12:return "mono audio is enabled";
                case 13:return "did connection";
                case 14:return "audio volume is too low";
                case 15:return "task or CMD be canceled";
                case 16: return "UF wrong string format";
                case 17:return "UF file not found";
                case 18:return "UF wrong file format";
                case 19:return "Attempt to contact online host failed";
                case 20: return "Attempt to perform RKI failed";
                case 0x300: return "Key Type(TDES) of Session Key is not same as the related Master Key.";
                case 0x400: return "Related Key was not loaded.";
                case 0x500: return "Key Same.";
                case 0x501: return "Key is all zero";
                case 0x502: return "TR-31 format error";
                case 0x702: return "PAN is Error Key.";
                case 0x705: return "No Internal MSR PAN (or Internal MSR PAN is erased timeout)";
                case 0X0C01: return "Incorrect Frame Tag";
                case 0X0C02: return "Incorrect Frame Type";
                case 0X0C03: return "Unknown Frame Type";
                case 0X0C04: return "Unknown Command";
                case 0X0C05: return "Unknown Sub-Command";
                case 0X0C06: return "CRC Error";
                case 0X0C07: return "Failed";
                case 0X0C08: return "Timeout";
                case 0X0C0A: return "Incorrect Parameter";
                case 0X0C0B: return "Command Not Supported";
                case 0X0C0C: return "Sub-Command Not Supported";
                case 0X0C0D: return "Parameter Not Supported / Status Abort Command";
                case 0X0C0F: return "Sub-Command Not Allowed";
                case 0X0D01: return "Incorrect Header Tag";
                case 0X0D02: return "Unknown Command";
                case 0X0D03: return "Unknown Sub-Command";
                case 0X0D04: return "CRC Error in Frame";
                case 0X0D05: return "Incorrect Parameter";
                case 0X0D06: return "Parameter Not Supported";
                case 0X0D07: return "Mal-formatted Data";
                case 0X0D08: return "Timeout";
                case 0X0D0A: return "Failed / NACK";
                case 0X0D0B: return "Command not Allowed";
                case 0X0D0C: return "Sub-Command not Allowed";
                case 0X0D0D: return "Buffer Overflow (Data Length too large for reader buffer)";
                case 0X0D0E: return "User Interface Event";
                case 0X0D11: return "Communication type not supported, VT-1, burst, etc.";
```

```
        case 0X0D12: return "Secure interface is not functional or is in an intermediate state.";
        case 0X0D13: return "Data field is not mod 8";
        case 0X0D14: return "Pad 0x80 not found where expected";
        case 0X0D15: return "Specified key type is invalid";
        case 0X0D1: return "Could not retrieve key from the SAM(InitSecureComm)";
        case 0X0D17: return "Hash code problem";
        case 0X0D18: return "Could not store the key into the SAM(InstallKey)";
        case 0X0D19: return "Frame is too large";
        case 0X0D1A: return "Unit powered up in authentication state but POS must resend the
InitSecureComm command";
        case 0X0D1B: return "The EEPROM may not be initialized because SecCommInterface does not
make sense";
        case 0X0D1C: return "Problem encoding APDU";
        case 0X0D20: return "Unsupported Index(ILM) SAM Transceiver error - problem communicating
with the SAM(Key Mgr)";
        case 0X0D2: return "Unexpected Sequence Counter in multiple frames for single bitmap(ILM)
Length error in data returned from the SAM(Key Mgr)";
        case 0X0D22: return "Improper bit map(ILM)";
        case 0X0D23: return "Request Online Authorization";
        case 0X0D24: return "ViVOCard3 raw data read successful";
        case 0X0D25: return "Message index not available(ILM) ViVOcomm activate transaction card
type(ViVOcomm)";
        case 0X0D26: return "Version Information Mismatch(ILM)";
        case 0X0D27: return "Not sending commands in correct index message index(ILM)";
        case 0X0D28: return "Time out or next expected message not received(ILM)";
        case 0X0D29: return "ILM languages not available for viewing(ILM)";
        case 0X0D2A: return "Other language not supported(ILM)";
        case 0X0D41: return "Unknown Error from SAM";
        case 0X0D42: return "Invalid data detected by SAM";
        case 0X0D43: return "Incomplete data detected by SAM";
        case 0X0D44: return "Reserved";
        case 0X0D45: return "Invalid key hash algorithm";
        case 0X0D46: return "Invalid key encryption algorithm";
        case 0X0D47: return "Invalid modulus length";
        case 0X0D48: return "Invalid exponent";
        case 0X0D49: return "Key already exists";
        case 0X0D4A: return "No space for new RID";
        case 0X0D4B: return "Key not found";
        case 0X0D4C: return "Crypto not responding";
        case 0X0D4D: return "Crypto communication error";
        case 0X0D4E: return "Module-specific error for Key Manager";
        case 0X0D4F: return "All key slots are full (maximum number of keys has been installed)";
        case 0X0D50: return "Auto-Switch OK";
        case 0X0D51: return "Auto-Switch failed";
        case 0X0D90: return "Account DUKPT Key not exist";
        case 0X0D91: return "Account DUKPT Key KSN exausted";
        case 0x0D00: return "This Key had been loaded.";
        case 0x0E00: return "Base Time was loaded.";
        case 0x0F00: return "Encryption Or Decryption Failed.";
        case 0x1000: return "Battery Low Warning (It is High Priority Response while Battery is
Low.)";
        case 0x1800: return "Send “Cancel Command” after send “Get Encrypted PIN” &”Get Numeric “&
“Get Amount””";
        case 0x1900: return "Press “Cancel” key after send “Get Encrypted PIN” &”Get Numeric “&
“Get Amount””";
        case 0x30FF: return "Security Chip is not connect";
        case 0x3000: return "Security Chip is deactivation & Device is In Removal Legally State.";
        case 0x3101: return "Security Chip is activation &  Device is In Removal Legally State.";
        case 0x5500: return "No Admin DUKPT Key.";
        case 0x5501: return "Admin  DUKPT Key STOP.";
        case 0x5502: return "Admin DUKPT Key KSN is Error.";
        case 0x5503: return "Get Authentication Code1 Failed.";
        case 0x5504: return "Validate Authentication Code Error.";
        case 0x5505: return "Encrypt or Decrypt data failed.";
        case 0x5506: return "Not Support the New Key Type.";
        case 0x5507: return "New Key Index is Error.";
        case 0x5508: return "Step Error.";
        case 0x5509: return "KSN Error";
        case 0x550A: return "MAC Error.";
        case 0x550B: return "Key Usage Error.";
        case 0x550C: return "Mode Of Use Error.";
        case 0x550F: return "Other Error.";
        case 0x6000: return "Save or Config Failed / Or Read Config Error.";
        case 0x6200: return "No Serial Number.";
        case 0x6900: return "Invalid Command - Protocol is right, but task ID is invalid.";
        case 0x6A01: return "Unsupported Command - Protocol and task ID are right, but command is
invalid - In this State";
        case 0x6A00: return "Unsupported Command - Protocol and task ID are right, but command is
invalid.";
        case 0x6B00: return "Unknown parameter in command - Protocol task ID and command are right,
but parameter is invalid.";
        case 0x6C00: return "Unknown parameter in command - Protocol task ID and command are right,
but length is out of the requirement.";
        case 0x7200: return "Device is suspend (MKSK suspend or press password suspend).";
        case 0x7300: return "PIN DUKPT is STOP (21 bit 1).";
        case 0x7400: return "Device is Busy.";
        case 0xE100: return "Can not enter sleep mode";
```

```
            case 0xE200: return "File has existed";
            case 0xE300: return "File has not existed";
            case 0xE313: return "IO line low -- Card error after session start";
            case 0xE400: return "Open File Error";
            case 0xE500: return "SmartCard Error";
            case 0xE600: return "Get MSR Card data is error";
            case 0xE700: return "Command time out";
            case 0xE800: return "File read or write is error";
            case 0xE900: return "Active 1850 error!";
            case 0xEA00: return "Load bootloader error";
            case 0xEF00: return "Protocol Error- STX or ETX or check error.";
            case 0xEB00: return "Picture is not exist";
            case 0x2C02: return "No Microprocessor ICC seated";
            case 0x2C06: return "no card seated to request ATR";
            case 0x2D01: return "Card Not Supported,";
            case 0x2D03: return "Card Not Supported, wants CRC";
            case 0x690D: return "Command not supported on reader without ICC support";
            case 0x8100: return "ICC error time out on power-up";
            case 0x8200: return "invalid TS character received - Wrong operation step";
            case 0x8300: return "Decode MSR Error";
            case 0x8400: return "TriMagII no Response";
            case 0x8500: return "No Swipe MSR Card";
            case 0x8510: return "No Financial Card";
            case 0x8600: return "Unsupported F, D, or combination of F and D";
            case 0x8700: return "protocol not supported EMV TD1 out of range";
            case 0x8800: return "power not at proper level";
            case 0x8900: return "ATR length too long";
            case 0x8B01: return "EMV invalid TA1 byte value";
            case 0x8B02: return "EMV TB1 required";
            case 0x8B03: return "EMV Unsupported TB1 only 00 allowed";
            case 0x8B04: return "EMV Card Error, invalid BWI or CWI";
            case 0x8B06: return "EMV TB2 not allowed in ATR";
            case 0x8B07: return "EMV TC2 out of range";
            case 0x8B08: return "EMV TC2 out of range";
            case 0x8B09: return "per EMV96 TA3 must be > 0xF";
            case 0x8B10: return "ICC error on power-up";
            case 0x8B11: return "EMV T=1 then TB3 required";
            case 0x8B12: return "Card Error, invalid BWI or CWI";
            case 0x8B13: return "Card Error, invalid BWI or CWI";
            case 0x8B17: return "EMV TC1/TB3 conflict*";
            case 0x8B20: return "EMV TD2 out of range must be T=1";
            case 0x8C00: return "TCK error";
            case 0xA304: return "connector has no voltage setting";
            case 0xA305: return "ICC error on power-up invalid (SBLK(IFSD) exchange";
            case 0xE301: return "ICC error after session start";
            case 0xFF00: return "Request to go online";
            case 0xFF01: return "EMV: Accept the offline transaction";
            case 0xFF02: return "EMV: Decline the offline transaction";
            case 0xFF03: return "EMV: Accept the online transaction";
            case 0xFF04: return "EMV: Decline the online transaction";
            case 0xFF05: return "EMV: Application may fallback to magstripe technology";
            case 0xFF06: return "EMV: ICC detected tah the conditions of use are not satisfied";
            case 0xFF07: return "EMV: ICC didn't accept transaction";
            case 0xFF08: return "EMV: Transaction was cancelled";
            case 0xFF09: return "EMV: Application was not selected by kernel or ICC format error or ICC
missing data error";
            case 0xFF0A: return "EMV: Transaction is terminated";
            case 0xFF0B: return "EMV: Other EMV Error";
            case 0xFFFF: return "NO RESPONSE";
            case 0xF002: return "ICC communication timeout";
            case 0xF003: return "ICC communication Error";
            case 0xF00F: return "ICC Card Seated and Highest Priority, disable MSR work request";
            case 0xF200: return "AID List / Application Data is not exist";
            case 0xF201: return "Terminal Data is not exist";
            case 0xF202: return "TLV format is error";
            case 0xF203: return "AID List is full";
            case 0xF204: return "Any CA Key is not exist";
            case 0xF205: return "CA Key RID is not exist";
            case 0xF206: return "CA Key Index it not exist";
            case 0xF207: return "CA Key is full";
            case 0xF208: return "CA Key Hash Value is Error";
            case 0xF209: return "Transaction  format error";
            case 0xF20A: return "The command will not be processing";
            case 0xF20B: return "CRL is not exist";
            case 0xF20C: return "CRL number  exceed max number";
            case 0xF20D: return "Amount,Other Amount,Trasaction Type  are  missing";
            case 0xF20E: return "The Identification of algorithm is mistake";
            case 0xF20F: return "No Financial Card";
            case 0xF210: return "In Encrypt Result state, TLV total Length is greater than Max Length";
            case 0x1001: return "INVALID ARG";
            case 0x1002: return "FILE_OPEN_FAILED";
            case 0x1003: return "FILE OPERATION_FAILED";
            case 0x2001: return "MEMORY_NOT_ENOUGH";
            case 0x3002: return "SMARTCARD_FAIL";
            case 0x3003: return "SMARTCARD_INIT_FAILED";
            case 0x3004: return "FALLBACK_SITUATION";
            case 0x3005: return "SMARTCARD_ABSENT";
```

```
            case 0x3006: return "SMARTCARD_TIMEOUT";
            case 0x5001: return "EMV_PARSING_TAGS_FAILED";
            case 0x5002: return "EMV_DUPLICATE_CARD_DATA_ELEMENT";
            case 0x5003: return "EMV_DATA_FORMAT_INCORRECT";
            case 0x5004: return "EMV_NO_TERM_APP";
            case 0x5005: return "EMV_NO_MATCHING_APP";
            case 0x5006: return "EMV_MISSING_MANDATORY_OBJECT";
            case 0x5007: return "EMV_APP_SELECTION_RETRY";
            case 0x5008: return "EMV_GET_AMOUNT_ERROR";
            case 0x5009: return "EMV_CARD_REJECTED";
            case 0x5010: return "EMV_AIP_NOT_RECEIVED";
            case 0x5011: return "EMV_AFL_NOT_RECEIVED";
            case 0x5012: return "EMV_AFL_LEN_OUT_OF_RANGE";
            case 0x5013: return "EMV_SFI_OUT_OF_RANGE";
            case 0x5014: return "EMV_AFL_INCORRECT";
            case 0x5015: return "EMV_EXP_DATE_INCORRECT";
            case 0x5016: return "EMV_EFF_DATE_INCORRECT";
            case 0x5017: return "EMV_ISS_COD_TBL_OUT_OF_RANGE";
            case 0x5018: return "EMV_CRYPTOGRAM_TYPE_INCORRECT";
            case 0x5019: return "EMV_PSE_NOT_SUPPORTED_BY_CARD";
            case 0x5020: return "EMV_USER_SELECTED_LANGUAGE";
            case 0x5021: return "EMV_SERVICE_NOT_ALLOWED";
            case 0x5022: return "EMV_NO_TAG_FOUND";
            case 0x5023: return "EMV_CARD_BLOCKED";
            case 0x5024: return "EMV_LEN_INCORRECT";
            case 0x5025: return "CARD_COM_ERROR";
            case 0x5026: return "EMV_TSC_NOT_INCREASED";
            case 0x5027: return "EMV_HASH_INCORRECT";
            case 0x5028: return "EMV_NO_ARC";
            case 0x5029: return "EMV_INVALID_ARC";
            case 0x5030: return "EMV_NO_ONLINE_COMM";
            case 0x5031: return "TRAN_TYPE_INCORRECT";
            case 0x5032: return "EMV_APP_NO_SUPPORT";
            case 0x5033: return "EMV_APP_NOT_SELECT";
            case 0x5034: return "EMV_LANG_NOT_SELECT";
            case 0x5035: return "EMV_NO_TERM_DATA";
            case 0x6001: return "CVM_TYPE_UNKNOWN";
            case 0x6002: return "CVM_AIP_NOT_SUPPORTED";
            case 0x6003: return "CVM_TAG_8E_MISSING";
            case 0x6004: return "CVM_TAG_8E_FORMAT_ERROR";
            case 0x6005: return "CVM_CODE_IS_NOT_SUPPORTED";
            case 0x6006: return "CVM_COND_CODE_IS_NOT_SUPPORTED";
            case 0x6007: return "NO_MORE_CVM";
            case 0x6008: return "PIN_BYPASSED_BEFORE";
            case 0x7001: return "PK_BUFFER_SIZE_TOO_BIG";
            case 0x7002: return "PK_FILE_WRITE_ERROR";
            case 0x7003: return "PK_HASH_ERROR";
            case 0x8001: return "NO_CARD_HOLDER_CONFIRMATION";
            case 0x8002: return "GET_ONLINE_PIN";
            case 0xD000: return "Data not exist";
            case 0xD001: return "Data access error";
            case 0xD100: return "RID not exist";
            case 0xD101: return "RID existed";
            case 0xD102: return "Index not exist";
            case 0xD200: return "Maximum exceeded";
            case 0xD201: return "Hash error";
            case 0xD205: return "System Busy";
            case 0x0E01: return "Unable to go online";
            case 0x0E02: return "Technical Issue";
            case 0x0E03: return "Declined";
            case 0x0E04: return "Issuer Referral transaction";
            case 0x0F01: return "Decline the online transaction";
            case 0x0F02: return "Request to go online";
            case 0x0F03: return "Transaction is terminated";
            case 0x0F05: return "Application was not selected by kernel or ICC format error or ICC
missing data error";
            case 0x0F07: return "ICC didn't accept transaction";
            case 0x0F0A: return "Application may fallback to magstripe technology";
            case 0x0F0C: return "Transaction was cancelled";
            case 0x0F0D: return "Timeout";
            case 0x0F0F: return "Other EMV Error";
            case 0x0F10: return "Accept the offline transaction";
            case 0x0F11: return "Decline the offline transaction";
            case 0x0F21: return "ICC detected tah the conditions of use are not satisfied";
            case 0x0F22: return "No app were found on card matching terminal configuration";
            case 0x0F23: return "Terminal file does not exist";
            case 0x0F24: return "CAPK file does not exist";
            case 0x0F25: return "CRL Entry does not exist";
            case 0x0FFE: return "Return code when blocking is disabled";
            case 0x0FFF: return "Return code when command is not applicable on the selected device";
            case 0xF005: return "ICC Encrypted C-APDU Data Structure Length Error Or Format Error.";
            case 0xBBE0: return "CM100 Success";
            case 0xBBE1: return "CM100 Parameter Error";
            case 0xBBE2: return "CM100 Low Output Buffer";
            case 0xBBE3: return "CM100 Card Not Found";
            case 0xBBE4: return "CM100 Collision Card Exists";
            case 0xBBE5: return "CM100 Too Many Cards Exist";
```

```
            case 0xBBE6: return "CM100 Saved Data Does Not Exist";
            case 0xBBE8: return "CM100 No Data Available";
            case 0xBBE9: return "CM100 Invalid CID Returned";
            case 0xBBEA: return "CM100 Invalid Card Exists";
            case 0xBBEC: return "CM100 Command Unsupported";
            case 0xBBED: return "CM100 Error In Command Process";
            case 0xBBEE: return "CM100 Invalid Command";

            case 0X9031: return "Unknown command";
            case 0X9032: return "Wrong parameter (such as the length of the command is incorrect)";

            case 0X9038: return "Wait (the command couldn't be finished in BWT)";
            case 0X9039: return "Busy (a previously command has not been finished)";
            case 0X903A: return "Number of retries over limit";

            case 0X9040: return "Invalid Manufacturing system data";
            case 0X9041: return "Not authenticated";
            case 0X9042: return "Invalid Master DUKPT Key";
            case 0X9043: return "Invalid MAC Key";
            case 0X9044: return "Reserved for future use";
            case 0X9045: return "Reserved for future use";
            case 0X9046: return "Invalid DATA DUKPT Key";
            case 0X9047: return "Invalid PIN Pairing DUKPT Key";
            case 0X9048: return "Invalid DATA Pairing DUKPT Key";
            case 0X9049: return "No nonce generated";
            case 0X9949: return "No GUID available.  Perform getVersion first.";
            case 0X9950: return "MAC Calculation unsuccessful. Check BDK value.";

            case 0X904A: return "Not ready";
            case 0X904B: return "Not MAC data";

            case 0X9050: return "Invalid Certificate";
            case 0X9051: return "Duplicate key detected";
            case 0X9052: return "AT checks failed";
            case 0X9053: return "TR34 checks failed";
            case 0X9054: return "TR31 checks failed";
            case 0X9055: return "MAC checks failed";
            case 0X9056: return "Firmware download failed";

            case 0X9060: return "Log is full";
            case 0X9061: return "Removal sensor unengaged";
            case 0X9062: return "Any hardware problems";

            case 0X9070: return "ICC communication timeout";
            case 0X9071: return "ICC data error (such check sum error)";
            case 0X9072: return "Smart Card not powered up";

        }
        return "";
    }
```

Chapter 9

Enumeration Reference

Common

```
    public enum EVENT_TRANSACTION_DATA_Types
    {
        EVENT_TRANSACTION_DATA_UNKNOWN, EVENT_TRANSACTION_DATA_CARD_DATA, EVENT_TRANSACTION_DATA_EMV_DATA,
          EVENT_TRANSACTION_DATA_MSR_CANCEL_KEY, EVENT_TRANSACTION_DATA_MSR_BACKSPACE_KEY,
        EVENT_TRANSACTION_DATA_MSR_ENTER_KEY, EVENT_TRANSACTION_DATA_MSR_DATA_ERROR, EVENT_TRANSACTION_PIN_DATA
    }

public enum CAPTURE_ENCODE_TYPE
{
    CAPTURE_ENCODE_TYPE_ISOABA, CAPTURE_ENCODE_TYPE_AAMVA, CAPTURE_ENCODE_TYPE_Other,
      CAPTURE_ENCODE_TYPE_Raw, CAPTURE_ENCODE_TYPE_JisI_II
}


public enum CAPTURE_ENCRYPT_TYPE
{
    CAPTURE_ENCRYPT_TYPE_TDES, CAPTURE_ENCRYPT_TYPE_AES, CAPTURE_ENCRYPT_TYPE_NONE
}


public enum EMV_ENCRYPTION_MODE
{
    EMV_ENCRYPTION_MODE_TDES = 0, EMV_ENCRYPTION_MODE_AES = 1
}


public enum EMV_ENCRYPTION_MODE
{
    EMV_ENCRYPTION_MODE_TDES = 0, EMV_ENCRYPTION_MODE_AES = 1
}


public enum EMV_LCD_DISPLAY_MODE
{
    EMV_LCD_DISPLAY_MODE_CANCEL = 0,EMV_LCD_DISPLAY_MODE_MENU = 1, EMV_LCD_DISPLAY_MODE_PROMPT = 2,
      EMV_LCD_DISPLAY_MODE_MESSAGE = 3, EMV_LCD_DISPLAY_MODE_LANGUAGE_SELECT = 8, EMV_LCD_DISPLAY_MODE_CLEAR_SCREEN = 16
}


public enum EMV_RESULT_CODE
{
    EMV_RESULT_CODE_APPROVED_OFFLINE = 0,
    EMV_RESULT_CODE_DECLINED_OFFLINE = 1,
    EMV_RESULT_CODE_APPROVED = 2,
    EMV_RESULT_CODE_DECLINED = 3,
    EMV_RESULT_CODE_GO_ONLINE = 4,
    EMV_RESULT_CODE_CALL_YOUR_BANK = 5,
    EMV_RESULT_CODE_NOT_ACCEPTED = 6,
    EMV_RESULT_CODE_FALLBACK_TO_MSR = 7,
    EMV_RESULT_CODE_TIMEOUT = 8,
    EMV_RESULT_CODE_GO_ONLINE_CTLS = 9,
    EMV_RESULT_CODE_AUTHENTICATE_TRANSACTION = 0x0010,
    EMV_RESULT_CODE_SWIPE_NON_ICC = 17,
    EMV_RESULT_CODE_CTLS_TWO_CARDS = 0x7A,
    EMV_RESULT_CODE_CTLS_TERMINATE = 0x7E,
    EMV_RESULT_CODE_CTLS_TERMINATE_TRY_ANOTHER = 0x7D,
    EMV_RESULT_CODE_UNABLE_TO_REACH_HOST
}
```

Chapter 10

EMV Tag Reference

Tag	Description
42	Issuer Identification Number (IIN)
4F	Application Identifier (ADF Name)
50	Application Label
52	Command to perform
56	Track 1 Data
57	Track 2 Equivalent Data
5A	Application Primary Account Number (PAN)
5D	Deleted (see 9D)
5F20	Cardholder Name
5F24	Application Expiration Date
5F25	Application Effective Date
5F28	Issuer Country Code
5F2A	Transaction Currency Code
5F2D	Language Preference
5F30	Service Code
5F34	Application Primary Account Number (PAN) Sequence Number (PSN)
5F36	Transaction Currency Exponent
5F3C	Transaction Reference Currency Code
5F3D	Transaction Reference Currency Exponent
5F50	Issuer URL
5F53	International Bank Account Number (IBAN)
5F54	Bank Identifier Code (BIC)
5F55	Issuer Country Code (alpha2 format)
5F56	Issuer Country Code (alpha3 format)
5F57	Account Type
61	Application Template
62	File Control Parameters (FCP) Template
6F	File Control Information (FCI) Template
70	READ RECORD Response Message Template
71	Issuer Script Template 1
72	Issuer Script Template 2
73	Directory Discretionary Template
77	Response Message Template Format 2
80	Response Message Template Format 1
81	Amount, Authorised (Binary)

Tag	Description
82	Application Interchange Profile (AIP)
83	Command Template
84	Dedicated File (DF) Name
86	Issuer Script Command
87	Application Priority Indicator
88	Short File Identifier (SFI)
89	Authorisation Code
8A	Authorisation Response Code (ARC)
8C	Card Risk Management Data Object List 1 (CDOL1)
8D	Card Risk Management Data Object List 2 (CDOL2)
8E	Cardholder Verification Method (CVM) List
8F	Certification Authority Public Key Index (PKI)
90	Issuer Public Key Certificate
91	Issuer Authentication Data
92	Issuer Public Key Remainder
93	Signed Application Data
94	Application File Locator (AFL)
95	Terminal Verification Results (TVR)
97	Transaction Certificate Data Object List (TDOL)
98	Transaction Certificate (TC) Hash Value
99	Transaction Personal Identification Number (PIN) Data
9A	Transaction Date
9B	Transaction Status Information
9C	Transaction Type
9D	Directory Definition File (DDF) Name
9F01	Acquirer Identifier
9F02	Amount, Authorised (Numeric)
9F03	Amount, Other (Numeric)
9F04	Amount, Other (Binary)
9F05	Application Discretionary Data
9F06	Application Identifier (AID) - terminal
9F07	Application Usage Control (AUC)
9F08	Application Version Number
9F09	Application Version Number
9F0B	Cardholder Name Extended
9F0D	Issuer Action Code - Default
9F0E	Issuer Action Code - Denial
9F0F	Issuer Action Code - Online
9F10	Issuer Application Data (IAD)
9F11	Issuer Code Table Index
9F12	Application Preferred Name
9F13	Last Online Application Transaction Counter (ATC) Register
9F14	Lower Consecutive Offline Limit
9F15	Merchant Category Code
9F16	Merchant Identifier
9F17	Personal Identification Number (PIN) Try Counter
9F18	Issuer Script Identifier
9F19	Deleted (see 9F49)
9F1A	Terminal Country Code

Tag	Description
9F1B	Terminal Floor Limit
9F1C	Terminal Identification
9F1D	Terminal Risk Management Data
9F1E	Interface Device (IFD) Serial Number
9F1F	Track 1 Discretionary Data
9F20	Track 2 Discretionary Data
9F21	Transaction Time
9F22	Certification Authority Public Key Index (PKI)
9F23	Upper Consecutive Offline Limit
9F26	Application Cryptogram (AC)
9F27	Cryptogram Information Data (CID)
9F29	Extended Selection
9F2A	Kernel Identifier
9F2D	Integrated Circuit Card (ICC) PIN Encipherment Public Key Certificate
9F2E	Integrated Circuit Card (ICC) PIN Encipherment Public Key Exponent
9F2F	Integrated Circuit Card (ICC) PIN Encipherment Public Key Remainder
9F32	Issuer Public Key Exponent
9F33	Terminal Capabilities
9F34	Cardholder Verification Method (CVM) Results
9F35	Terminal Type
9F36	Application Transaction Counter (ATC)
9F37	Unpredictable Number (UN)
9F37	Unpredictable Number (UN) (Reader/Terminal)
9F38	Processing Options Data Object List (PDOL)
9F39	Point-of-Service (POS) Entry Mode
9F3A	Amount, Reference Currency
9F3B	Application Reference Currency
9F3C	Transaction Reference Currency Code
9F3D	Transaction Reference Currency Exponent
9F40	Additional Terminal Capabilities
9F41	Transaction Sequence Counter
9F42	Application Currency Code
9F43	Application Reference Currency Exponent
9F44	Application Currency Exponent
9F45	Data Authentication Code
9F46	Integrated Circuit Card (ICC) Public Key Certificate
9F46	Application Public Key Certificate
9F47	Integrated Circuit Card (ICC) Public Key Exponent
9F47	Application Public Key Exponent
9F48	Integrated Circuit Card (ICC) Public Key Remainder
9F48	Application Public Key Remainder
9F49	Dynamic Data Authentication Data Object List (DDOL)
9F4A	Static Data Authentication Tag List (SDA)
9F4B	Signed Dynamic Application Data (SDAD)
9F4C	ICC Dynamic Number
9F4D	Log Entry
9F4E	Merchant Name and Location
9F4F	Log Format
9F50	Offline Accumulator Balance

Tag	Description
9F50	Cardholder Verification Status
9F51	Application Currency Code
9F51	DRDOL
9F52	Application Default Action (ADA)
9F52	Terminal Compatibility Indicator
9F53	Consecutive Transaction Counter International Limit (CTCIL)
9F53	Transaction Category Code
9F53	Terminal Interchange Profile (dynamic)
9F54	Cumulative Total Transaction Amount Limit (CTTAL)
9F54	DS ODS Card
9F55	Geographic Indicator
9F56	Issuer Authentication Indicator
9F57	Issuer Country Code
9F58	Consecutive Transaction Counter Limit (CTCL)
9F59	Consecutive Transaction Counter Upper Limit (CTCUL)
9F5A	Application Program Identifier (Program ID)
9F5B	Issuer Script Results
9F5B	DSDOL
9F5C	Cumulative Total Transaction Amount Upper Limit (CTTAUL)
9F5C	DS Requested Operator ID
9F5C	Magstripe Data Object List (MDOL)
9F5D	Available Offline Spending Amount (AOSA)
9F5D	Application Capabilities Information (ACI)
9F5E	Consecutive Transaction International Upper Limit (CTIUL)
9F5E	DS ID
9F5F	DS Slot Availability
9F5F	Offline Balance
9F60	CVC3 (Track1)
9F60	Issuer Update Parameter
9F60	P3 Generated 3DES KEYS
9F61	CVC3 (Track2)
9F62	PCVC3 (Track1)
9F62	Encrypted PIN - ISO 95641 Format 0 (Thales P3 Format 01)
9F63	Offline Counter Initial Value
9F63	PUNATC (Track1)
9F64	NATC (Track1)
9F65	PCVC3 (Track2)
9F66	Terminal Transaction Qualifiers (TTQ)
9F66	PUNATC (Track2)
9F67	MSD Offset
9F67	NATC (Track2)
9F68	Card Additional Processes
9F69	Card Authentication Related Data
9F69	UDOL
9F6A	Unpredictable Number (Numeric)
9F6B	Card CVM Limit
9F6B	Track 2 Data
9F6C	Card Transaction Qualifiers (CTQ)
9F6D	VLP Reset Threshold
9F6D	Mag-stripe Application Version Number (Reader)

Tag	Description
9F6D	Kernel 4 Reader Capabilities
9F6E	Third Party Data
9F6E	Form Factor Indicator (FFI)
9F6E	Terminal Transaction Capabilities
9F6F	DS Slot Management Control
9F70	Protected Data Envelope 1
9F70	Card Interface Capabilities
9F71	Protected Data Envelope 2
9F71	Mobile CVM Results
9F72	Protected Data Envelope 3
9F72	Consecutive Transaction Limit (International—Country)
9F73	Protected Data Envelope 4
9F73	Currency Conversion Parameters
9F74	Protected Data Envelope 5
9F74	VLP Issuer Authorisation Code
9F75	Unprotected Data Envelope 1
9F75	Cumulative Total Transaction Amount Limit-Dual Currency
9F76	Unprotected Data Envelope 2
9F76	Secondary Application Currency Code
9F77	Unprotected Data Envelope 3
9F78	Unprotected Data Envelope 4
9F79	Unprotected Data Envelope 5
9F77	VLP Funds Limit
9F78	VLP Single Transaction Limit
9F79	VLP Available Funds
9F7A	VLP Terminal Support Indicator
9F7B	VLP Terminal Transaction Limit
9F7C	Customer Exclusive Data (CED)
9F7C	Merchant Custom Data
9F7D	DS Summary 1
9F7D	VISA Applet Data
9F7E	Mobile Support Indicator
9F7E	Application life cycle data (8 first bytes)
9F7F	DS Unpredictable Number
9F7F	Card Production Life Cycle (CPLC) Data
A5	File Control Information (FCI) Proprietary Template
BF0C	File Control Information (FCI) Issuer Discretionary Data
BF50	Visa Fleet - CDO
BF60	Integrated Data Storage Record Update Template
C3	Card issuer action code -decline
C4	Card issuer action code -default
C5	Card issuer action code online
C6	PIN Try Limit
C7	CDOL 1 Related Data Length
C8	Card risk management country code
C9	Card risk management currency code
CA	Lower cummulative offline transaction amount
CB	Upper cumulative offline transaction amount
CD	Card Issuer Action Code (PayPass) – Default

Tag	Description
CE	Card Issuer Action Code (PayPass) – Online
CF	Card Issuer Action Code (PayPass) – Decline
D1	Currency conversion table
D2	Integrated Data Storage Directory (IDSD)
D3	Additional check table
D5	Application Control
D6	Default ARPC response code
D7	Application Control (PayPass)
D8	AIP (PayPass)
D9	AFL (PayPass)
DA	Static CVC3-TRACK1
DB	Static CVC3-TRACK2
DC	IVCVC3-TRACK1
DD	IVCVC3-TRACK2
DF70	Generic Name String
DF71	Value Added Tax 1
DF71	Generic Numeric
DF72	Value Added Tax 2
DF72	Generic Specification String
DF73	Merchant Category Code
DF73	Generic Implementation String
DF74	Discover Optional Features
DF75	Communications Error Message Delay
DF76	TVR from GenAC
DF77	ViVOpay MSR Custom Data Output Tag
DF78	MC Timing Performance Enable
DF79	Card Disable Mask
DF7A	Card Disable Interval
DF7B	Serial Port (UART) Inter-character Timeout Period
DF7C	Auto Switch Feature
DF7D	Track Formatting Feature
DF7F	Improved Collision Detection & Media Removal Feature
FF70	Serial Finite State Machine Version
FF71	Transaction Finite State Machine Version
FF72	System Information Suite
FF73	Serial Protocol Version
FF74	Serial Protocol Suite
FF75	L1 Paypass Version
FF76	L1 LCR Version
FF77	L2 Card App Version
FF78	L2 Card App Suite
FF79	GMEDs Data
FF79	User Experience Version
FF7A	User Experience Suite
FF7B	ViVOtech Proprietary Suite
FF7C	VIUDS Scheme IDs Supported
FF7D	VIUDS Scheme ID Selection Criteria
FFE0	Registered Application Provider Identifier (RID)
FFE1	Partial Selection Allowed
FFE2	Application Flow

Tag	Description
FFE3	Selection Features - GR 1.2.10
FFE4	Group Number / Fallback Group
FFE5	Max AID Length
FFE6	AID Disabled
FFE7	Interface Support
FFE8	Exclude from Processing
FFE9	Kernel ID Transaction Type Group List
FFEA	Default Kernel ID
FFF0	Specific Features Switch
FFF1	Terminal Contactless Transaction Limit
FFF2	Terminal IFD
FFF3	Application Capability
FFF4	Visa Reader Risk Flags
FFF5	CVM Required Limit
FFF6	Torn Transaction Log Clean Interval (minutes)
FFF7	Burst Mode
FFF8	UI Scheme
FFF9	LCD Font Size
FFFA	LCD delay Time
FFFB	Language Option for LCD
FFFC	Force MagStripe
FFFD	TAC - Online
FFFE	TAC - Default
FFFF	TAC - Denial
DF8123	Reader Contactless Floor Limit Data
DFDE04	MSR Encryption Option
DFEE12	KSN of Account DUKPT Key
DFEE15	Application Selection Indicator
DFEE16	DUKPT Key or MKSK Select for Online PIN Encrypted
DFEE17	ICC Terminal Entry Mode
DFEE18	MSR Terminal Entry Mode
DFEE19	Online DOL
DFEE1A	Output data element
DFEE1B	Authorization Request data elements
DFEE1E	Terminal Configuration
DFEE1F	Issuer Script Limit
DFEE20	ICC power on waiting time
DFEE21	ICC L1 data transaction waiting time
DFEE22	Driver (Menu, Get PIN, Get MSR) Timeout
DFEE23	MSR all track data
DFEE24	Force Acceptance
DFEE25	ICC Response Code
DFEE26	Encryption Status Information
DFEE27	MSR Control
FFEE01	ViVOpay TLV Group Tag
FFEE02	ViVOpay Pre-PPSE Special Flow Group Tag
FFEE03	ViVOpay Post-PPSE Special Flow Group Tag
FFEE04	M/Chip3 Intermediate Message Data
FFEE04	ViVOpay MChip Group Status

Tag	Description
FFEE05	M/Chip3 Intermediate Message Marker
FFEE06	ApplePay VAS Container
FFEE10	ViVOpay MChip Group Tag
FFEE11	ViVOpay Discover Group Tag
FFEE12	KSN of Account DUKPT Key
FFEE13	Track 1 Data
FFEE14	Track 2 Data
FFEE1C	Unpredictable Number Range
FFEE1D	Sensitive Data Mask

Chapter 11

Namespace Index

11.1 Packages

Here are the packages with brief descriptions (if available):

Chapter 12

Class Index

12.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 13

Namespace Documentation

13.1 IDTechSDK Namespace Reference

Classes

- class EMV_Callback
- class IDT_KioskIII
- class IDTCryptoData
- class IDTTransactionData

Enumerations

- enum EMV_CALLBACK_TYPE { **EMV_CALLBACK_TYPE_LCD** =1, **EMV_CALLBACK_TYPE_PINPAD** =2, **EMV_CALLBACK_MSR** =3 }
- enum EMV_LCD_DISPLAY_MODE { **EMV_LCD_DISPLAY_MODE_CANCEL** = 0, **EMV_LCD_DISPLAY_MODE_MENU** = 1, **EMV_LCD_DISPLAY_MODE_PROMPT** = 2, **EMV_LCD_DISPLAY_MODE_MESSAGE** = 3, **EMV_LCD_DISPLAY_MODE_LANGUAGE_SELECT** = 8, **EMV_LCD_DISPLAY_MODE_CLEAR_SCREEN** = 16 }
- enum CTLS_APPLICATION { **CTLS_APPLICATION_NONE** = 0, **CTLS_APPLICATION_MASTERCARD** = 1, **CTLS_APPLICATION_VISA** = 2, **CTLS_APPLICATION_AMEX** = 3, **CTLS_APPLICATION_DISCOVER** = 4, **CTLS_APPLICATION_SPEEDPASS** = 5, **CTLS_APPLICATION_GIFT_CARD** = 6, **CTLS_APPLICATION_DINERS_CLUB** = 7, **CTLS_APPLICATION_EN_ROUTE** = 8, **CTLS_APPLICATION_JCB** = 9, **CTLS_APPLICATION_VIVO_DIAGNOSTIC** = 10, **CTLS_APPLICATION_HID** = 11, **CTLS_APPLICATION_MSR_SWIPE** = 12, **CTLS_APPLICATION_RESERVED** = 13, **CTLS_APPLICATION_DES_FIRE_TRACK_DATA** = 14, **CTLS_APPLICATION_DES_FIRE_RAW_DATA** = 15, **CTLS_APPLICATION_RBS** = 17, **CTLS_APPLICATION_VIVO_COMM** = 20 }
- enum EMV_PIN_MODE { **EMV_PIN_MODE_CANCEL** =0, **EMV_PIN_MODE_ONLINE_DUKPT** =1, **EMV_PIN_MODE_ONLINE_MKSK** =2, **EMV_PIN_MODE_OFFLINE** =3 }
- enum EMV_RESULT_CODE { **EMV_RESULT_CODE_APPROVED_OFFLINE** = 0, **EMV_RESULT_CODE_DECLINED_OFFLINE** = 1, **EMV_RESULT_CODE_APPROVED** = 2, **EMV_RESULT_CODE_DECLINED** = 3, **EMV_RESULT_CODE_GO_ONLINE** = 4, **EMV_RESULT_CODE_CALL_YOUR_BANK** = 5, **EMV_RESULT_CODE_NOT_ACCEPTED** = 6, **EMV_RESULT_CODE_FALLBACK_TO_MSR** = 7, **EMV_RESULT_CODE_TIMEOUT** = 8, **EMV_RESULT_CODE_GO_ONLINE_CTLS** = 9, **EMV_RESULT_CODE_FAILED** = 10, **EMV_RESULT_CODE_AUTHENTICATE_TRANSACTION** = 0x0010, **EMV_RESULT_CODE_TRANSACTION_CANCELED** = 0x0012, **EMV_RESULT_CODE_SWIPE_NON_ICC** = 0x11, **EMV_RESULT_CODE_CTLS_TWO_CARDS** = 0x7A, **EMV_RESULT_CODE_CTLS_TERMI**

NATE = 0x7E,
EMV_RESULT_CODE_CTLS_TERMINATE_TRY_ANOTHER = 0x7D, **EMV_RESULT_CODE_MSR_SW↩IPE_CAPTURED** = 0x80, **EMV_RESULT_CODE_REQUEST_ONLINE_PIN** = 0x81, **EMV_RESULT_CO↩DE_REQUEST_SIGNATURE** = 0x82,
EMV_RESULT_CODE_FALLBACK_TO_CONTACT = 0x83, **EMV_RESULT_CODE_FALLBACK_TO_O↩THER** = 0x84, **EMV_RESULT_CODE_REVERSAL_REQUIRED** = 0x85, **EMV_RESULT_CODE_ADVISE↩_REQUIRED** = 0x86,
EMV_RESULT_CODE_ADVISE_REVERSAL_REQUIRED = 0x87, **EMV_RESULT_CODE_NO_ADVISE↩_REVERSAL_REQUIRED** = 0x88, **EMV_RESULT_CODE_UNABLE_TO_REACH_HOST** = 0xFF, **EMV_↩RESULT_CODE_FILE_ARG_INVALID** = 0x1001,
EMV_RESULT_CODE_FILE_OPEN_FAILED = 0x1002, **EMV_RESULT_CODE_FILE_OPERATION_FA↩ILED** = 0x1003, **EMV_RESULT_CODE_MEMORY_NOT_ENOUGH** = 0x2001, **EMV_RESULT_CODE_S↩MARTCARD_OK** = 0x3001,
EMV_RESULT_CODE_SMARTCARD_FAIL = 0x3002, **EMV_RESULT_CODE_SMARTCARD_INIT_FAI↩LED** = 0x3003, **EMV_RESULT_CODE_FALLBACK_SITUATION** = 0x3004, **EMV_RESULT_CODE_SMA↩RTCARD_ABSENT** = 0x3005,
EMV_RESULT_CODE_SMARTCARD_TIMEOUT = 0x3006, **EMV_RESULT_CODE_MSR_CARD_ERROR** = 0x3007, **EMV_RESULT_CODE_MSR_CARD_READ_ERROR** = 0x3012, **EMV_RESULT_CODE_PAR↩SING_TAGS_FAILED** = 0x5001,
EMV_RESULT_CODE_CARD_DATA_ELEMENT_DUPLICATE = 0x5002, **EMV_RESULT_CODE_DAT↩A_FORMAT_INCORRECT** = 0x5003, **EMV_RESULT_CODE_APP_NO_TERM** = 0x5004, **EMV_RESULT↩_CODE_APP_NO_MATCHING** = 0x5005,
EMV_RESULT_CODE_MANDATORY_OBJECT_MISSING = 0x5006, **EMV_RESULT_CODE_APP_SEL↩ECTION_RETRY** = 0x5007, **EMV_RESULT_CODE_AMOUNT_ERROR_GET** = 0x5008, **EMV_RESULT_↩CODE_CARD_REJECTED** = 0x5009,
EMV_RESULT_CODE_AIP_NOT_RECEIVED = 0x5010, **EMV_RESULT_CODE_AFL_NOT_RECEIVED** = 0x5011, **EMV_RESULT_CODE_AFL_LEN_OUT_OF_RANGE** = 0x5012, **EMV_RESULT_CODE_SFI_O↩UT_OF_RANGE** = 0x5013,
EMV_RESULT_CODE_AFL_INCORRECT = 0x5014, **EMV_RESULT_CODE_EXP_DATE_INCORRECT** = 0x5015, **EMV_RESULT_CODE_EFF_DATE_INCORRECT** = 0x5016, **EMV_RESULT_CODE_ISS_COD_↩TBL_OUT_OF_RANGE** = 0x5017,
EMV_RESULT_CODE_CRYPTOGRAM_TYPE_INCORRECT = 0x5018, **EMV_RESULT_CODE_PSE↩_BY_CARD_NOT_SUPPORTED** = 0x5019, **EMV_RESULT_CODE_USER_LANGUAGE_SELECTED** = 0x5020, **EMV_RESULT_CODE_SERVICE_NOT_ALLOWED** = 0x5021,
EMV_RESULT_CODE_NO_TAG_FOUND = 0x5022, **EMV_RESULT_CODE_CARD_BLOCKED** = 0x5023, **EMV_RESULT_CODE_LEN_INCORRECT** = 0x5024, **EMV_RESULT_CODE_CARD_COM_ERROR** = 0x5025,
EMV_RESULT_CODE_TSC_NOT_INCREASED = 0x5026, **EMV_RESULT_CODE_HASH_INCORRECT** = 0x5027, **EMV_RESULT_CODE_ARC_NOT_PRESENCED** = 0x5028, **EMV_RESULT_CODE_ARC_INVA↩LID** = 0x5029,
EMV_RESULT_CODE_COMM_NO_ONLINE = 0x5030, **EMV_RESULT_CODE_TRAN_TYPE_INCORR↩ECT** = 0x5031, **EMV_RESULT_CODE_APP_NO_SUPPORT** = 0x5032, **EMV_RESULT_CODE_APP_NO↩T_SELECT** = 0x5033,
EMV_RESULT_CODE_LANG_NOT_SELECT = 0x5034, **EMV_RESULT_CODE_TERM_DATA_NOT_P↩RESENCED** = 0x5035, **EMV_RESULT_CODE_CVM_TYPE_UNKNOWN** = 0x6001, **EMV_RESULT_COD↩E_CVM_AIP_NOT_SUPPORTED** = 0x6002,
EMV_RESULT_CODE_CVM_TAG_8E_MISSING = 0x6003, **EMV_RESULT_CODE_CVM_TAG_8E_↩FORMAT_ERROR** = 0x6004, **EMV_RESULT_CODE_CVM_CODE_IS_NOT_SUPPORTED** = 0x6005, **EMV_RESULT_CODE_CVM_COND_CODE_IS_NOT_SUPPORTED** = 0x6006,
EMV_RESULT_CODE_CVM_NO_MORE = 0x6007, **EMV_RESULT_CODE_PIN_BYPASSED_BEFORE** = 0x6008, **EMV_RESULT_CODE_UNKONWN** = 0xffff }

13.1.1 Enumeration Type Documentation

13.1.1.1 enum IDTechSDK.CTLS_APPLICATION [strong]

Define CTLS_APPLICATION.

13.1.1.2 enum IDTechSDK.EMV_CALLBACK_TYPE [strong]

Define EMV_CALLBACK_TYPES.

13.1.1.3 enum IDTechSDK.EMV_LCD_DISPLAY_MODE [strong]

Define EMV_LCD_DISPLAY_MODE.

13.1.1.4 enum IDTechSDK.EMV_PIN_MODE [strong]

Define EMV_PIN_MODE.

13.1.1.5 enum IDTechSDK.EMV_RESULT_CODE [strong]

Define EMV_PIN_MODE.

Chapter 14

Class Documentation

14.1 IDTechSDK.EMV_Callback Class Reference

Public Attributes

- int msr_swipeTimeout
- int msr_displayMessage
- EMV_PIN_MODE pin_pinMode
- int pin_entryStartTimeout
- int pin_entryInterval
- byte[ ] pin_KSN
- byte[ ] pin_truncatedPAN
- EMV_CALLBACK_TYPE callbackType
- EMV_LCD_DISPLAY_MODE lcd_displayMode
- int lcd_entryTimeout
- int lcd_entryTimeoutMinor
- byte[ ] language
- byte[ ] lcd_messages
- UInt16 lcd_backlightTimeout
- bool maskEntry

14.1.1 Detailed Description

Class for LCD Message

14.1.2 Member Data Documentation

14.1.2.1 EMV_CALLBACK_TYPE IDTechSDK.EMV_Callback.callbackType

Callback Type.
1- EMV_CALLBACK_TYPE_LCD: LCD Display Hardware Event 2- EMV_CALLBACK_TYPE_PINPAD: Pinpad Hardware Event 3- EMV_CALLBACK_MSR: MSR Hardware Event

14.1.2.2 byte [ ] IDTechSDK.EMV_Callback.language

Message Language

2 Bytes

- EN - English (default)
- ES - Spanish
- ZH - Chinese
- FR – French

14.1.2.3 UInt16 IDTechSDK.EMV_Callback.lcd_backlightTimeout

Backlight Timeout

If Normal Display or Menu Display, Total timeout for keypad entry, in second default is 30 seconds. 0x0000 = backlight off, 0xFFFF = backlight on

14.1.2.4 EMV_LCD_DISPLAY_MODE IDTechSDK.EMV_Callback.lcd_displayMode

Display Mode.
1- LCD_DISPLAY_MODE_MENU: Menu selection, response required with selected menu index #, or 0 to cancel 2- LCD_DISPLAY_MODE_PROMPT: Message Prompt, response required 'E' for Enter/Accept, or 'C' for cancel 3- LCD_DISPLAY_MODE_MESSAGE: Display Message, no response required 8 – LCD_DISPLAY_MODE_LANG↩ UAGE_SELECT: Language selection, response required with selected language index # 16 - LCD_DISPLAY_M↩ ODE_CLEAR_SCREEN: Request to clear LCD screen of information

14.1.2.5 int IDTechSDK.EMV_Callback.lcd_entryTimeout

Keypad Entry Timeout

If Normal Display or Menu Display, Total timeout for keypad entry, in second default is 30 seconds.

14.1.2.6 int IDTechSDK.EMV_Callback.lcd_entryTimeoutMinor

Keypad Entry Timeout Minor

If Normal Display or Menu Display, minor timeout during each keypad entry, in second, little endian, default is 10 seconds. Note: Minor timeout will erase all previous keypad entry.

14.1.2.7 byte [ ] IDTechSDK.EMV_Callback.lcd_messages

Display Message

repeatable combination of [Line][Message][0x1C] [Line] - Display line number (1-First Line, n-nth Line), Maximum 16 lines. •The lower 7 bits is for line number. •The MSB is to indicate following message is a Message String or Message ID. •MSB – 0: Message String. (It is valid for “Menu Display” and “Language Menu Display”) •MSB – 1: Message ID. (It is only valid for “Menu Display”) [Message] - Message String or Message ID. Message String: •“Menu Display” : character in the range of 0x20 – 0x7f, Maximum 16 characters • “Language Menu Display” : 2 bytes Language ID EN - English (default) ES - Spanish ZH - Chinese FR – French

14.1.2.8 bool IDTechSDK.EMV_Callback.maskEntry

Mask Entry

If True, keypad entry should be masked with '∗'

14.1.2.9 int IDTechSDK.EMV_Callback.msr_displayMessage

MSR Message

Message to display during swipe request

14.1.2.10 int IDTechSDK.EMV_Callback.msr_swipeTimeout

Swipe Timeout

Timeout value waiting for MSR Swipe

14.1.2.11 int IDTechSDK.EMV_Callback.pin_entryInterval

PIN Entry Interval Timeout value of interval between input each PIN

14.1.2.12 int IDTechSDK.EMV_Callback.pin_entryStartTimeout

PIN Entry Start Timeout

Timeout value waiting for PIN entry to start

14.1.2.13 byte [ ] IDTechSDK.EMV_Callback.pin_KSN

PIN KSN

Pairing DUKPT KSN

14.1.2.14 EMV_PIN_MODE IDTechSDK.EMV_Callback.pin_pinMode

PIN Mode.
0- EMV_PIN_MODE_CANCEL: Entry cancel through command. No response required 1- EMV_PIN_MODE_O↩
NLINE_DUKPT: PIN_DUKPT_KEY required as response 2- EMV_PIN_MODE_ONLINE_MKSK: PIN_SESSION↩
_KEY required as response 3 – EMV_PIN_MODE_OFFLINE: PIN_PAIRING_DUKPT_KEY required as response, unless devices does not implement pairing function, then plaintext PIN required as response

14.1.2.15 byte [ ] IDTechSDK.EMV_Callback.pin_truncatedPAN

Truncated PAN

Truncated PAN

The documentation for this class was generated from the following file:

- Source/IDT_Transactions.cs

14.2 IDTechSDK.IDT_KioskIII Class Reference

Public Member Functions

- RETURN_CODE config_getSerialNumber (ref string response)
- RETURN_CODE device_retrieveAIDList (ref byte[][] response)
- RETURN_CODE device_sendDataCommand (string cmd, bool calcLRC, ref byte[] response)
- RETURN_CODE device_sendDataCommand_ext (string cmd, bool calcLRC, ref byte[] response, int timeout, bool noResponse)

- RETURN_CODE device_getFirmwareVersion (ref string response)
- RETURN_CODE device_getMerchantRecord (int index, ref byte[] record)
- RETURN_CODE device_getTransactionResults (ref IDTTransactionData results)
- RETURN_CODE device_pingDevice ()
- RETURN_CODE device_controlUserInterface (byte[] values)
- RETURN_CODE device_sendVivoCommandP2 (byte command, byte subCommand, byte[] data, ref byte[] response, string ip="")
- RETURN_CODE device_sendVivoCommandP2_ext (byte command, byte subCommand, byte[] data, ref byte[] response, int timeout, bool noResponse, string ip="")
- RETURN_CODE ctls_retrieveAIDList (ref byte[][] response)
- RETURN_CODE ctls_getAllConfigurationGroups (ref byte[][] response)
- RETURN_CODE ctls_retrieveApplicationData (byte[] AID, ref byte[] tlv)
- RETURN_CODE ctls_removeApplicationData (byte[] AID)
- RETURN_CODE ctls_setApplicationData (byte[] tlv)
- RETURN_CODE ctls_setDefaultConfiguration ()
- RETURN_CODE ctls_setConfigurationGroup (byte[] tlv)
- RETURN_CODE ctls_retrieveTerminalData (ref byte[] tlv)
- RETURN_CODE ctls_setTerminalData (byte[] tlv)
- RETURN_CODE ctls_getConfigurationGroup (int group, ref byte[] tlv)
- RETURN_CODE ctls_removeConfigurationGroup (int group)
- RETURN_CODE ctls_setCAPK (byte[] key)
- RETURN_CODE ctls_retrieveCAPK (byte[] capk, ref byte[] key)
- RETURN_CODE ctls_removeCAPK (byte[] capk)
- RETURN_CODE ctls_removeAllCAPK ()
- RETURN_CODE ctls_retrieveCAPKList (ref byte[] keys)
- RETURN_CODE device_startTransaction (double amount, double amtOther, int exponent, int type, int timeout, byte[] tags, bool isFastEMV=false)
- RETURN_CODE device_activateTransaction (int timeout, byte[] tags, bool isFastEMV=false)
- RETURN_CODE ctls_startTransaction (double amount, double amtOther, int exponent, int type, int timeout, byte[] tags, bool isFastEMV=false)
- RETURN_CODE ctls_updateBalance (byte statusCode, byte[] authCode, byte[] date, byte[] time)
- RETURN_CODE ctls_activateTransaction (int timeout, byte[] tags, bool isFastEMV=false)
- RETURN_CODE ctls_cancelTransaction ()
- RETURN_CODE device_setBurstMode (byte mode)
- RETURN_CODE device_setMerchantRecord (int index, bool enabled, string merchantID, string merchantURL)
- RETURN_CODE device_pollForToken (byte seconds, ref byte card, ref byte[] serialNumber)
- RETURN_CODE device_sendPAE (string command, ref string response, int timeout, string ip="")
- RETURN_CODE device_setPollMode (byte mode)
- RETURN_CODE device_startRKI ()
- RETURN_CODE device_SymmetricRKI (int type)
- RETURN_CODE device_updateDeviceFirmware (byte[] firmwareData)
- RETURN_CODE device_enablePassThrough (bool enablePassThrough)
- RETURN_CODE felica_authentication (byte[] key, string ip="")
- RETURN_CODE felica_readWithMac (int numBlocks, byte[] blockList, ref byte[] blocks, string ip="")
- RETURN_CODE felica_writeWithMac (int blockNumber, byte[] data, string ip="")
- RETURN_CODE felica_read (byte[] serviceCode, int numBlocks, byte[] blockList, ref byte[] blocks, string ip="")
- RETURN_CODE felica_write (byte[] serviceCode, int blockCount, byte[] blockList, byte[] data, ref byte[] statusFlag, string ip="")
- RETURN_CODE ctls_nfcCommand (byte[] nfcCmdPkt, ref byte[] response, string ip="")
- RETURN_CODE felica_requestService (byte[] nodeCode, ref byte[] response, string ip="")

Static Public Member Functions

- static bool useSerialPort (int port)
- static int getCommandTimeout ()
- static void setCommandTimeout (int milliseconds)
- static bool useSerialPort (int port, int baud)
- static bool useUSB ()
- static void setCallback (CallBack my_Callback)
- static void setCallback (IntPtr my_Callback, SynchronizationContext context)
- static String SDK_Version ()
- static void lcd_retrieveMessage (DisplayMessages.DISPLAY_MESSAGE_LANGUAGE lang, DisplayMessages.DISPLAY_MESSAGE_IDENTIFIER id, ref string line1, ref string line2)

Properties

- static IDT_KioskIII SharedController `[get]`

14.2.1 Detailed Description

Class for KioskIII and KioskIV ICC reader

14.2.2 Member Function Documentation

14.2.2.1 RETURN_CODE IDTechSDK.IDT_KioskIII.config_getSerialNumber (ref string *response*)

Polls device for Serial Number

Parameters

response	Returns Serial Number

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

14.2.2.2 RETURN_CODE IDTechSDK.IDT_KioskIII.ctls_activateTransaction (int *timeout*, byte[] *tags*, bool *isFastEMV* = `false`)

Start CTLS Transaction Request

Authorizes the CTLS transaction

The tags will be returned in the callback routine.

Parameters

timeout	Timeout value in seconds.
tags	The tags to be included in the request. Passed as TLV Data. Example, tag 9F02 with amount 0x000000000100 would be 0x9F0206000000000100
isFastEMV	If TRUE, it will populate the IDTTransactionData.fastEMV with ASCII data similar to IDTech FastEMV KB output, after performing an auto-authenticate and auto-complete with ResultCode = Could Not Contact Host

Returns

RETURN_CODE: Values can be parsed with errorCode.getErrorString()

NOTE ON APPLEPAY VAS: To enable ApplePay VAS, first a merchant record must be defined in one of the six available index positions (1-6) using device_setMerchantRecord, then container tag FFEE06 must be sent as part of the additional tags parameter of ctls_startTransaction. Tag FFEE06 must contain tag 9F26 and 9F22, and can optionanally contain tags 9F2B and DFO1. Example FFEE06189F220201009F2604000000009F2B050100000000↩ DF010101 9F22 = two bytes = ApplePay Terminal Applicaiton Version Number. Hard defined as 0100 for now. (required) 9F26 = four bytes = ApplePay Terminal Capabilities Information (required)

- Byte 1 = RFU
- Byte 2 = Terminal Type
- - Bit 8 = VAS Support (1=on, 0 = off)
- - Bit 7 = Touch ID Required (1=on, 0 = off)
- - Bit 6 = RFU
- - Bit 5 = RFU
- - Bit 1,2,3,4
- - - 0 = Payment Terminal
- - - 1 = Transit Terminal
- - - 2 = Access Terminal
- - - 3 = Wireless Handoff Terminal
- - - 4 = App Handoff Terminal
- - - 15 = Other Terminal
- Byte 3 = RFU
- Byte 4 = Terminal Mode
- - 0 = ApplePay VAS OR ApplePay
- - 1 = ApplePay VAS AND ApplePay
- - 2 = ApplePay VAS ONLY
- - 3 = ApplePay ONLY 9F2B = 5 bytes = ApplePay VAS Filter. Each byte filters for that specific merchant index (optional) DF01 = 1 byte = ApplePay VAS Protocol. (optional)
- - Bit 1 : 1 = URL VAS, 0 = Full VAS
- - Bit 2 : 1 = VAS Beeps, 0 = No VAS Beeps
- - Bit 3 : 1 = Silent Comm Error, 2 = EMEA Comm Error
- - Bit 4-8 : RFU

14.2.2.3 RETURN_CODE IDTechSDK.IDT_KioskIII.ctls_cancelTransaction ( )

Cancel EMV Transaction

Cancels the currently executing EMV transaction.

Returns

RETURN_CODE: Values can be parsed with errorCode.getErrorString()

14.2.2.4 RETURN_CODE IDTechSDK.IDT_KioskIII.ctls_getAllConfigurationGroups (ref byte *response[ ][ ]*)

Retrieve All Configuration Groups

Returns all the Configuration Groups installed on the terminal for CTLS

Parameters

response	array of CTLS groups as TLV bytes

Returns

RETURN_CODE: Values can be parsed with errorCode.getErrorString()

14.2.2.5 RETURN_CODE IDTechSDK.IDT_KioskIII.ctls_getConfigurationGroup (int *group,* ref byte[ ] *tlv*)

Get Configuration Group

Retrieves the Configuration for the specified Group.

Parameters

group	Configuration Group
tlv	return data

Returns

RETURN_CODE: Values can be parsed with errorCode.getErrorString()

14.2.2.6 RETURN_CODE IDTechSDK.IDT_KioskIII.ctls_nfcCommand (byte[ ] *nfcCmdPkt,* ref byte[ ] *response,* string *ip* = " ")

NFC Command

This command uses nfcCmdPkt[0] in command data field to implement different functions. This command should be used in Pass-Through mode and command with “Poll for a NFC Tag” data should be used first. Command with other data can only be used once the “Poll for a NFC Tag” command has indicated that a NFC tag is present.

Parameters

nfcCmdPkt	System Code • Poll for NFC Tag: nfcCmdPkt[0] = 0xff, nfcCmdPkt[1] = timeout value (in seconds) • Tag1 Static Get All Data: nfcCmdPkt[0] = 0x11 • Tag1 Static Read a Byte: nfcCmdPkt[0] = 0x12, nfcCmdPkt[1] = Address of Data • Tag1 Static Write a Byte: nfcCmdPkt[0] = 0x13 nfcCmdPkt[1] = Address of Data, nfcCmdPkt[2] = Data to be written • Tag1 Static Write a Byte NE: nfcCmdPkt[0] = 0x14, nfcCmdPkt[1] = Address of Data, nfcCmdPkt[2] = Data to be written • Tag1 Dynamic Read a Segment: nfcCmdPkt[0] = 0x15, nfcCmdPkt[1] = Address of Segment • Tag1 Dynamic Read 8 Bytes: nfcCmdPkt[0] = 0x16, nfcCmdPkt[1] = Address of Data • Tag1 Dynamic Write 8 Bytes: nfcCmdPkt[0] = 0x17, nfcCmdPkt[1] = Address of Data, nfcCmdPkt[2]∼nfcCmdPkt[9] = Data to be written • Tag1 Dynamic Write 8 Bytes NE: nfcCmdPkt[0] = 0x18, nfcCmdPkt[1] = Address of Data, nfcCmdPkt[2]∼nfcCmdPkt[9] = Data to be written • Tag2 Read Data (16 bytes): nfcCmdPkt[0] = 0x21, nfcCmdPkt[1] = Address of Data • Tag2 Write Data (4 bytes): nfcCmdPkt[0] = 0x22, nfcCmdPkt[1] = Address of Data, nfcCmdPkt[2]∼nfcCmdPkt[5] = Data to be written • Tag2 Select Sect: nfcCmdPkt[0] = 0x23, nfcCmdPkt[1] = Sect number • Tag3 Read Data: – nfcCmdPkt[0] = 0x41, – nfcCmdPkt[1] = Number of services, value n – nfcCmdPkt[2]∼nfcCmdPkt[2n+1]: Service code list – nfcCmdPkt[2n+2]: Number of blocks, value m. – nfcCmdPkt[2n+3....]: Block list, length is 2m∼3m • Tag3 Write Data: – nfcCmdPkt[0] = 0x41, – nfcCmdPkt[1] = Number of services, value n – nfcCmdPkt[2]∼nfcCmdPkt[2n+1]: Service code list – nfcCmdPkt[2n+2]: Number of blocks, value m. – nfcCmdPkt[2n+3....]: Block list, length is 2m∼3m – nfcCmdPkt[...]: Block data, length is 16m • Tag4 Command: nfcCmdPkt[0] = 0x81, nfcCmdPkt[1]∼nfcCmdPkt[n]:data
response	Response as explained in FeliCA Lite-S User's Manual
ip	IP Address of target device (optional)

Returns

RETURN_CODE: Values can be parsed with errorCode.getErrorString()

14.2.2.7 RETURN_CODE IDTechSDK.IDT_KioskIII.ctls_removeAllCAPK ( )

Remove All Certificate Authority Public Key

Removes all the CAPK for CTLS

Returns

RETURN_CODE: Values can be parsed with errorCode.getErrorString()

14.2.2.8 RETURN_CODE IDTechSDK.IDT_KioskIII.ctls_removeApplicationData (byte[ ] *AID*)

Remove Application Data by AID

Removes the Application Data for CTLS as specified by the AID name passed as a parameter

Parameters

AID	Name of ApplicationID Must be between 5 and 16 bytes

Returns

RETURN_CODE: Values can be parsed with errorCode.getErrorString()

14.2.2.9 RETURN_CODE IDTechSDK.IDT_KioskIII.ctls_removeCAPK (byte[ ] *capk*)

Remove Certificate Authority Public Key

Removes the CAPK as specified by the RID/Index

Parameters

capk	6 byte CAPK = 5 bytes RID + 1 byte INDEX

Returns

RETURN_CODE: Values can be parsed with errorCode.getErrorString()

14.2.2.10 RETURN_CODE IDTechSDK.IDT_KioskIII.ctls_removeConfigurationGroup (int *group*)

Remove Configuration Group

Removes the Configuration as specified by the Group. Must not by group 0

Parameters

group	Configuration Group

Return values

RETURN_CODE	Return codes listed as typedef enum in IDTCommon:RETURN_CODE. Values can be parsed with IDT_BTPay::device_getResponseCodeString:()

14.2.2.11 RETURN_CODE IDTechSDK.IDT_KioskIII.ctls_retrieveAIDList (ref byte *response[][]*)

Retrieve AID list

Returns all the AID names and their assigned groups installed on the terminal for CTLS.

Parameters

response	array of 2-tag TLV data objects: FFE4 (group name) followed by 9F06 (AID)

Returns

RETURN_CODE: Values can be parsed with errorCode.getErrorString()

14.2.2.12 RETURN_CODE IDTechSDK.IDT_KioskIII.ctls_retrieveApplicationData (byte[ ] *AID,* ref byte[ ] *tlv*)

Retrieve Application Data by AID

Retrieves the CTLS Application Data as specified by the AID name passed as a parameter.

Parameters

AID	Name of ApplicationID. Must be between 5 and 16 bytes
tlv	The TLV elements of the requested AID

Returns

RETURN_CODE: Values can be parsed with errorCode.getErrorString()

14.2.2.13 RETURN_CODE IDTechSDK.IDT_KioskIII.ctls_retrieveCAPK (byte[ ] *capk,* ref byte[ ] *key*)

Retrieve Certificate Authority Public Key

Retrieves the CAPK for CTLS as specified by the RID/Index passed as a parameter.

Parameters

capk	6 bytes CAPK = 5 bytes RID + 1 byte Index
key	Response returned as a CAKey format: [5 bytes RID][1 byte Index][1 byte Hash Algorithm][1 byte Encryption Algorithm][20 bytes HashValue][4 bytes Public Key Exponent][2 bytes Modulus Length][Variable bytes Modulus] Where: • Hash Algorithm: The only algorithm supported is SHA-1.The value is set to 0x01 • Encryption Algorithm: The encryption algorithm in which this key is used. Currently support only one type: RSA. The value is set to 0x01. • HashValue: Which is calculated using SHA-1 over the following fields: RID & Index & Modulus & Exponent • Public Key Exponent: Actually, the real length of the exponent is either one byte or 3 bytes. It can have two values: 3 (Format is 0x00 00 00 03), or 65537 (Format is 0x00 01 00 01) • Modulus Length: LenL LenH Indicated the length of the next field. • Modulus: This is the modulus field of the public key. Its length is specified in the field above.

Returns

RETURN_CODE: Values can be parsed with errorCode.getErrorString()

14.2.2.14 RETURN_CODE IDTechSDK.IDT_KioskIII.ctls_retrieveCAPKList (ref byte[ ] *keys*)

Retrieve the Certificate Authority Public Key list

Returns all the CAPK RID and Index installed on the terminal for CTLS.

Parameters

keys	[key1][key2]...[keyn], each key 6 bytes where key = 5 bytes RID + 1 byte index

Returns

RETURN_CODE: Values can be parsed with errorCode.getErrorString()

14.2.2.15 RETURN_CODE IDTechSDK.IDT_KioskIII.ctls_retrieveTerminalData (ref byte[ ] *tlv*)

Retrieve Terminal Data

Retrieves the Terminal Data for CTLS. This is configuration group 0 (Tag FFEE - > FFEE0100). The terminal data can also be retrieved by ctls_getConfigurationGroup(0).

Parameters

tlv	Response returned as a TLV

Returns

RETURN_CODE: Values can be parsed with errorCode.getErrorString()

14.2.2.16 RETURN_CODE IDTechSDK.IDT_KioskIII.ctls_setApplicationData (byte[ ] *tlv*)

Set Application Data by AID

Sets the Application Data for CTLS as specified by TLV data

Parameters

tlv	Application data in TLV format The first tag of the TLV data must be the group number (FFE4). The second tag of the TLV data must be the AID (9F06)

Example valid TLV, for Group #2, AID a0000000035010: "ffe401029f0607a0000000051010ffe10101ffe50110ffe30114ffe20106"

Returns

RETURN_CODE: Values can be parsed with errorCode.getErrorString()

14.2.2.17 RETURN_CODE IDTechSDK.IDT_KioskIII.ctls_setCAPK (byte[ ] *key*)

Set Certificate Authority Public Key

Sets the CAPK for CTLS as specified by the CAKey structure

Parameters

key	CAKey format: [5 bytes RID][1 byte Index][1 byte Hash Algorithm][1 byte Encryption Algorithm][20 bytes HashValue][4 bytes Public Key Exponent][2 bytes Modulus Length][Variable bytes Modulus] Where: • Hash Algorithm: The only algorithm supported is SHA-1.The value is set to 0x01 • Encryption Algorithm: The encryption algorithm in which this key is used. Currently support only one type: RSA. The value is set to 0x01. • HashValue: Which is calculated using SHA-1 over the following fields: RID & Index & Modulus & Exponent • Public Key Exponent: Actually, the real length of the exponent is either one byte or 3 bytes. It can have two values: 3 (Format is 0x00 00 00 03), or 65537 (Format is 0x00 01 00 01) • Modulus Length: LenL LenH Indicated the length of the next field. • Modulus: This is the modulus field of the public key. Its length is specified in the field above.

Returns

RETURN_CODE: Values can be parsed with errorCode.getErrorString()

14.2.2.18 RETURN_CODE IDTechSDK.IDT_KioskIII.ctls_setConfigurationGroup (byte[ ] *tlv*)

Set Configuration Group

Sets the Configuration Group for CTLS as specified by the TLV data

Parameters

tlv	Configuration Group Data in TLV format The first tag of the TLV data must be the group number (FFE4). A second tag must exist

Returns

RETURN_CODE: Values can be parsed with errorCode.getErrorString()

14.2.2.19 RETURN_CODE IDTechSDK.IDT_KioskIII.ctls_setDefaultConfiguration ( )

Set Default Configuration Group

Resets the device to default CTLS configuration group settings

Returns

RETURN_CODE: Values can be parsed with errorCode.getErrorString()

14.2.2.20 RETURN_CODE IDTechSDK.IDT_KioskIII.ctls_setTerminalData (byte[ ] *tlv*)

Set Terminal Data

Sets the Terminal Data for CTLS as specified by the TLV. The first TLV must be Configuration Group Number (Tag FFE4). The terminal global data is group 0, so the first TLV would be FFE40100. Other groups can be defined using

this method (1 or greater), and those can be retrieved with emv_getConfigurationGroup(int group), and deleted with emv_removeConfigurationGroup(int group). You cannot delete group 0.

Parameters

tlv	TerminalData configuration data

Return values

RETURN_CODE	Return codes listed as typedef enum in IDTCommon:RETURN_CODE. Values can be parsed with IDT_BTPay::device_getResponseCodeString:()

14.2.2.21 RETURN_CODE IDTechSDK.IDT_KioskIII.ctls_startTransaction (double *amount,* double *amtOther,* int *exponent,* int *type,* int *timeout,* byte[ ] *tags,* bool *isFastEMV =* `false`)

Start CTLS Transaction Request

The tags will be returned in the callback routine.

Parameters

amount	Transaction amount value (tag value 9F02)
amtOther	Other amount value, if any (tag value 9F03)
type	Transaction type (tag value 9C).
timeout	Timeout value in seconds.
tags	Any other tags to be included in the request. Passed as TLV data. Example, tag 9F02 with amount 0x000000000100 would be 0x9F0206000000000100 If tags 9F02 (amount),9F03 (other amount), or 9C (transaction type) are included, they will take priority over these values supplied as individual parameters to this method.
isFastEMV	If TRUE, it will populate the IDTTransactionData.fastEMV with ASCII data similar to IDTech FastEMV KB output, after performing an auto-authenticate and auto-complete with ResultCode = Could Not Contact Host

Returns

RETURN_CODE: Values can be parsed with errorCode.getErrorString()

NOTE ON APPLEPAY VAS: To enable ApplePay VAS, first a merchant record must be defined in one of the six available index positions (1-6) using device_setMerchantRecord, then container tag FFEE06 must be sent as part of the additional tags parameter of ctls_startTransaction. Tag FFEE06 must contain tag 9F26 and 9F22, and can optionanally contain tags 9F2B and DFO1. Example FFEE06189F220201009F2604000000009F2B050100000000↩ DF010101 9F22 = two bytes = ApplePay Terminal Applicaiton Version Number. Hard defined as 0100 for now. (required) 9F26 = four bytes = ApplePay Terminal Capabilities Information (required)

- Byte 1 = RFU
- Byte 2 = Terminal Type
- - Bit 8 = VAS Support (1=on, 0 = off)
- - Bit 7 = Touch ID Required (1=on, 0 = off)
- - Bit 6 = RFU
- - Bit 5 = RFU
- - Bit 1,2,3,4
- - - 0 = Payment Terminal

- - - 1 = Transit Terminal
- - - 2 = Access Terminal
- - - 3 = Wireless Handoff Terminal
- - - 4 = App Handoff Terminal
- - - 15 = Other Terminal
- Byte 3 = RFU
- Byte 4 = Terminal Mode
- - 0 = ApplePay VAS OR ApplePay
- - 1 = ApplePay VAS AND ApplePay
- - 2 = ApplePay VAS ONLY
- - 3 = ApplePay ONLY 9F2B = 5 bytes = ApplePay VAS Filter. Each byte filters for that specific merchant index (optional) DF01 = 1 byte = ApplePay VAS Protocol. (optional)
- - Bit 1 : 1 = URL VAS, 0 = Full VAS
- - Bit 2 : 1 = VAS Beeps, 0 = No VAS Beeps
- - Bit 3 : 1 = Silent Comm Error, 2 = EMEA Comm Error
- - Bit 4-8 : RFU

14.2.2.22 RETURN_CODE IDTechSDK.IDT_KioskIII.ctls_updateBalance (byte *statusCode,* byte[ ] *authCode,* byte[ ] *date,* byte[ ] *time*)

Update Balance

This command is the authorization response sent by the issuer to the terminal including the Authorization Status (OK or NOT OK).

This command is also being used in some implementations (i.e. EMEA) to communicate the results of Issuer Authentication to the reader in order to display the correct LCD messages. With this command, the POS passes the authorization result (OK/NOT OK), and possibly the Authorization Code (Auth_Code)/Date/Time to the terminal.

Parameters

statusCode	00: OK, 01: NOT OK, 02: (ARC response 89 for Interac)
authCode	Authorization code from host. Six bytes. Optional
date	Transaction date. If null, uses current terminal date. 3 bytes compressed numeric YYMMDD (tag value 9A).
time	Transaction time. If null, uses current terminal time. 3 bytes compressed numeric HHMMSS (tag value 9F21).

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

14.2.2.23 RETURN_CODE IDTechSDK.IDT_KioskIII.device_activateTransaction (int *timeout,* byte[ ] *tags,* bool *isFastEMV* = `false`)

Start CTLS Transaction Request

Authorizes the CTLS transaction

The tags will be returned in the callback routine.

Parameters

timeout	Timeout value in seconds.
tags	The tags to be included in the request. Passed as TLV Data. Example, tag 9F02 with amount 0x000000000100 would be 0x9F0206000000000100
isFastEMV	If TRUE, it will populate the IDTTransactionData.fastEMV with ASCII data similar to IDTech FastEMV KB output, after performing an auto-authenticate and auto-complete with ResultCode = Could Not Contact Host

Returns

RETURN_CODE: Values can be parsed with errorCode.getErrorString()

NOTE ON APPLEPAY VAS: To enable ApplePay VAS, first a merchant record must be defined in one of the six available index positions (1-6) using device_setMerchantRecord, then container tag FFEE06 must be sent as part of the additional tags parameter of ctls_startTransaction. Tag FFEE06 must contain tag 9F26 and 9F22, and can optionanally contain tags 9F2B and DFO1. Example FFEE06189F220201009F2604000000009F2B050100000000↩ DF010101 9F22 = two bytes = ApplePay Terminal Applicaiton Version Number. Hard defined as 0100 for now. (required) 9F26 = four bytes = ApplePay Terminal Capabilities Information (required)

- Byte 1 = RFU
- Byte 2 = Terminal Type
- - Bit 8 = VAS Support (1=on, 0 = off)
- - Bit 7 = Touch ID Required (1=on, 0 = off)
- - Bit 6 = RFU
- - Bit 5 = RFU
- - Bit 1,2,3,4
- - - 0 = Payment Terminal
- - - 1 = Transit Terminal
- - - 2 = Access Terminal
- - - 3 = Wireless Handoff Terminal
- - - 4 = App Handoff Terminal
- - - 15 = Other Terminal
- Byte 3 = RFU
- Byte 4 = Terminal Mode
- - 0 = ApplePay VAS OR ApplePay
- - 1 = ApplePay VAS AND ApplePay
- - 2 = ApplePay VAS ONLY
- - 3 = ApplePay ONLY 9F2B = 5 bytes = ApplePay VAS Filter. Each byte filters for that specific merchant index (optional) DF01 = 1 byte = ApplePay VAS Protocol. (optional)
- - Bit 1 : 1 = URL VAS, 0 = Full VAS
- - Bit 2 : 1 = VAS Beeps, 0 = No VAS Beeps
- - Bit 3 : 1 = Silent Comm Error, 2 = EMEA Comm Error
- - Bit 4-8 : RFU

14.2.2.24 RETURN_CODE IDTechSDK.IDT_KioskIII.device_controlUserInterface (byte[] *values*)

Control User Interface

Controls the User Interface: Display, Beep, LED

```
@param values Four bytes to control the user interface
Byte[0] = LCD Message
Messages 00-07 are normally controlled by the reader.
- 00h: Idle Message (Welcome)
- 01h: Present card (Please Present Card)
- 02h: Time Out or Transaction cancel (No Card)
- 03h: Transaction between reader and card is in the middle (Processing...)
- 04h: Transaction Pass (Thank You)
- 05h: Transaction Fail (Fail)
- 06h: Amount (Amount $ 0.00 Tap Card)
- 07h: Balance or Offline Available funds (Balance $ 0.00) Messages 08-0B are controlled by the terminal
- 08h: Insert or Swipe card (Use Chip & PIN)
- 09h: Try Again(Tap Again)
- 0Ah: Tells the customer to present only one card (Present 1 card only)
- 0Bh: Tells the customer to wait for authentication/authorization (Wait)
- FFh: indicates the command is setting the LED/Buzzer only.
Byte[1] = Beep Indicator
- 00h: No beep
- 01h: Single beep
- 02h: Double beep
- 03h: Three short beeps
- 04h: Four short beeps
- 05h: One long beep of 200 ms
- 06h: One long beep of 400 ms
- 07h: One long beep of 600 ms
- 08h: One long beep of 800 ms
Byte[2] = LED Number
- 00h: LED 0 (Power LED) 01h: LED 1
- 02h: LED 2
- 03h: LED 3
- FFh: All LEDs
Byte[3] = LED Status
- 00h: LED Off
- 01h: LED On
```

Returns

RETURN_CODE: Values can be parsed with errorCode.getErrorString()

14.2.2.25 RETURN_CODE IDTechSDK.IDT_KioskIII.device_enablePassThrough (bool *enablePassThrough*)

Enable Pass Through

Enables Pass Through Mode for direct communication with L1 interface (power on icc, send apdu, etc).

Parameters

enablePassThrough	true = pass through ON, false = pass through OFF

Returns

RETURN_CODE: Values can be parsed with errorCode.getErrorString()

14.2.2.26 RETURN_CODE IDTechSDK.IDT_KioskIII.device_getFirmwareVersion (ref string *response*)

Polls device for Firmware Version

Parameters

response	Response returned of Firmware Version

Returns

RETURN_CODE: Values can be parsed with errorCode.getErrorString()

14.2.2.27 RETURN_CODE IDTechSDK.IDT_KioskIII.device_getMerchantRecord (int *index,* ref byte[ ] *record*)

Get Merchant Record Gets the merchant record for ApplePay VAS

Parameters

index	Merchant Record index, valid values 1-6
record	Data returned containing 99 bytes: Byte 0 = Merchand Index Byte 1 = Merchant Enabled (1 = enabled) Byte 2 - 33 = Merchant Protocol Hash-256 value Byte 34 = Length of Merchant URL Bytes 35 - 99 = URL

Returns

RETURN_CODE: Values can be parsed with errorCode.getErrorString()

14.2.2.28 RETURN_CODE IDTechSDK.IDT_KioskIII.device_getTransactionResults (ref **IDTTransactionData** *results*)

Get Transaction Results Gets the transaction results when the reader is functioning in "Auto Poll" mode

Parameters

results	The transaction results

Returns

RETURN_CODE: Values can be parsed with errorCode.getErrorString(). When no data is available, return code = RETURN_CODE_NO_DATA_AVAILABLE

14.2.2.29 RETURN_CODE IDTechSDK.IDT_KioskIII.device_pingDevice ( )

Ping Device

Pings the reader. If connected, returns success. Otherwise, returns timeout.

Returns

RETURN_CODE: Values can be parsed with errorCode.getErrorString()

14.2.2.30 RETURN_CODE IDTechSDK.IDT_KioskIII.device_pollForToken (byte *seconds,* ref byte *card,* ref byte[ ] *serialNumber*)

Poll for Token

Once Pass-Through Mode is started, ViVOpay will not poll for any cards until the “Poll for Token” command is received. This command tells ViVOpay to start polling for a Type A or Type B PICC until a PICC is detected or a timeout occurs.

This command automatically turns the RF Antenna on.

If a PICC is detected within the specified time limit, ViVOpay activates it and responds back to the terminal with card related data such as the Serial Number. If no PICC is detected within the specified time limit, ViVOpay stops polling and responds back indicating that no card was found. No card related data is returned in this case

Parameters

timeout	Timeout, in seconds to wait for card to be detected
card	Card Type: • 00h None (Card Not Detected or Could not Activate) • 01h ISO 14443 Type A (Supports ISO 14443-4 Protocol) • 02h ISO 14443 Type B (Supports ISO 14443-4 Protocol) • 03h Mifare Type A (Standard) • 04h Mifare Type A (Ultralight) • 05h ISO 14443 Type A (Does not support ISO 14443-4 Protocol) • 06h ISO 14443 Type B (Does not support ISO 14443-4 Protocol) • 07h ISO 14443 Type A and Mifare (NFC phone)
serialNumber	Serial Number or the UID of the PICC

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

14.2.2.31 RETURN_CODE IDTechSDK.IDT_KioskIII.device_retrieveAIDList (ref byte *response[][]*)

Retrieve AID list

Returns all the AID names and their assigned groups installed on the terminal for CTLS/CONTACT.

Parameters

response	array of TLV data objects: FFE4 (group name) followed by 9F06 (AID), and DFEE4F (Interface 01 = CTLS, 02 = CONTACT)

Returns

RETURN_CODE: Values can be parsed with errorCode.getErrorString()

14.2.2.32 RETURN_CODE IDTechSDK.IDT_KioskIII.device_sendDataCommand (string *cmd,* bool *calcLRC,* ref byte[ ] *response*)

Send a data command to the device

Sends a command to the device.

Parameters

cmd	String representation of command to execute
calcLRC	If TRUE, this will wrap command with start/length/lrc/sum/end: '{STX}{Len_Low}{Len_High} data {CheckLRC} {CheckSUM} {ETX}'
response	Response data

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

14.2.2.33 RETURN_CODE IDTechSDK.IDT_KioskIII.device_sendDataCommand_ext (string *cmd,* bool *calcLRC,* ref byte[ ] *response,* int *timeout,* bool *noResponse*)

Send a data command to the device - extended

Sends a command to the device.

Parameters

cmd	String representation of command to execute
calcLRC	If TRUE, this will wrap command as NGA with start/length/lrc/sum/end: '{STX}{Len_Low}{Len_High} data {CheckLRC} {CheckSUM} {ETX}'
response	Response data
timeout	Timeout value waiting for response, in milliseconds (1000 = 1 second)
noResponse	if TRUE, this will not wait for a response and immediately return SUCCESS
calcITP	If TRUE, this will wrap command as ITP with start/end/lrc: '{STX} data {ETX}{CheckLRC}'

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

14.2.2.34 RETURN_CODE IDTechSDK.IDT_KioskIII.device_sendPAE (string *command,* ref string *response,* int *timeout,* string *ip =* " ")

Send Payment Application Engine Command

Executes a PAE command

Parameters

command	ASCII command string, should start with "∗PAE"
response	command response
timeout	timeout waiting for PAE response
ip	Optional IP address when connected via TCP/IP

Returns

RETURN_CODE: Values can be parsed with errorCode.getErrorString()

14.2.2.35 RETURN_CODE IDTechSDK.IDT_KioskIII.device_sendVivoCommandP2 (byte *command,* byte *subCommand,* byte[ ] *data,* ref byte[ ] *response,* string *ip* = " ")

Send Vivo Command Protocol 2

Sends a protocol 2 command to Vivo readers (IDG/NEO)

Parameters

command	Command
subCommand	Sub-Command
data	Data. May be null
response	Response
ip	Optional IP

Returns

RETURN_CODE: Values can be parsed with errorCode.getErrorString()

14.2.2.36 RETURN_CODE IDTechSDK.IDT_KioskIII.device_sendVivoCommandP2_ext (byte *command,* byte *subCommand,* byte[ ] *data,* ref byte[ ] *response,* int *timeout,* bool *noResponse,* string *ip* = " ")

Send Vivo Command Protocol 2 Extended

Sends a protocol 2 command to Vivo readers (IDG/NEO)

Parameters

command	Command
subCommand	Sub-Command
data	Data. May be null
response	Response
timeout	Timeout, in milliseconds (3000 = 3 seconds)
noResponse	TRUE = don't wait for response, FALSE = wait for response defined by timeout
ip	Optional IP

Returns

RETURN_CODE: Values can be parsed with errorCode.getErrorString()

14.2.2.37 RETURN_CODE IDTechSDK.IDT_KioskIII.device_setBurstMode (byte *mode*)

Send Burst Mode

Sets the burst mode forthe device.

Parameters

mode	0 = OFF, 1 = Always On, 2 = Auto Exit

Returns

RETURN_CODE: Values can be parsed with errorCode.getErrorString()

14.2.2.38 RETURN_CODE IDTechSDK.IDT_KioskIII.device_setMerchantRecord (int *index,* bool *enabled,* string *merchantID,* string *merchantURL*)

Set Merchant Record Sets the merchant record for ApplePay VAS

Parameters

index	Merchant Record index, valid values 1-6
enabled	Merchant Enabled/Valid flag
merchantID	Merchant unique identifer registered with Apple. Example com.idtechproducts.applePay
merchantURL	Merchant URL, when applicable

Returns

RETURN_CODE: Values can be parsed with errorCode.getErrorString()

14.2.2.39 RETURN_CODE IDTechSDK.IDT_KioskIII.device_setPollMode (byte *mode*)

Send Poll Mode

Sets the poll mode forthe device. Auto Poll keeps reader active, Poll On Demand only polls when requested by terminal

Parameters

mode	0 = Auto Poll, 1 = Poll On Demand

Returns

RETURN_CODE: Values can be parsed with errorCode.getErrorString()

14.2.2.40 RETURN_CODE IDTechSDK.IDT_KioskIII.device_startRKI ( )

Start Legacy Remote Key Injection

Starts a remote key injection request with IDTech RKI Legacy servers.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

14.2.2.41 RETURN_CODE IDTechSDK.IDT_KioskIII.device_startTransaction (double *amount,* double *amtOther,* int *exponent,* int *type,* int *timeout,* byte[ ] *tags,* bool *isFastEMV =* `false`)

Start CTLS Transaction Request

The tags will be returned in the callback routine.

Parameters

amount	Transaction amount value (tag value 9F02)
amtOther	Other amount value, if any (tag value 9F03)
type	Transaction type (tag value 9C).
timeout	Timeout value in seconds.
tags	Any other tags to be included in the request. Passed as TLV data. Example, tag 9F02 with amount 0x000000000100 would be 0x9F0206000000000100 If tags 9F02 (amount),9F03 (other amount), or 9C (transaction type) are included, they will take priority over these values supplied as individual parameters to this method.
isFastEMV	If TRUE, it will populate the IDTTransactionData.fastEMV with ASCII data similar to IDTech FastEMV KB output, after performing an auto-authenticate and auto-complete with ResultCode = Could Not Contact Host

Returns

RETURN_CODE: Values can be parsed with errorCode.getErrorString()

NOTE ON APPLEPAY VAS: To enable ApplePay VAS, first a merchant record must be defined in one of the six available index positions (1-6) using device_setMerchantRecord, then container tag FFEE06 must be sent as part of the additional tags parameter of ctls_startTransaction. Tag FFEE06 must contain tag 9F26 and 9F22, and can optionanally contain tags 9F2B and DFO1. Example FFEE06189F220201009F2604000000009F2B050100000000↩ DF010101 9F22 = two bytes = ApplePay Terminal Applicaiton Version Number. Hard defined as 0100 for now. (required) 9F26 = four bytes = ApplePay Terminal Capabilities Information (required)

- Byte 1 = RFU
- Byte 2 = Terminal Type
- - Bit 8 = VAS Support (1=on, 0 = off)
- - Bit 7 = Touch ID Required (1=on, 0 = off)
- - Bit 6 = RFU
- - Bit 5 = RFU
- - Bit 1,2,3,4
- - - 0 = Payment Terminal
- - - 1 = Transit Terminal
- - - 2 = Access Terminal
- - - 3 = Wireless Handoff Terminal
- - - 4 = App Handoff Terminal
- - - 15 = Other Terminal
- Byte 3 = RFU
- Byte 4 = Terminal Mode
- - 0 = ApplePay VAS OR ApplePay
- - 1 = ApplePay VAS AND ApplePay
- - 2 = ApplePay VAS ONLY
- - 3 = ApplePay ONLY 9F2B = 5 bytes = ApplePay VAS Filter. Each byte filters for that specific merchant index (optional) DF01 = 1 byte = ApplePay VAS Protocol. (optional)

- - Bit 1 : 1 = URL VAS, 0 = Full VAS
- - Bit 2 : 1 = VAS Beeps, 0 = No VAS Beeps
- - Bit 3 : 1 = Silent Comm Error, 2 = EMEA Comm Error
- - Bit 4-8 : RFU

14.2.2.42 RETURN_CODE IDTechSDK.IDT_KioskIII.device_SymmetricRKI (int *type*)

Start Remote Key Injection

Starts a remote key injection request with IDTech Symmetric RKI servers. Set/Get RKI url with IDT_Device.RKI_↩ URL.

Parameters

type	0 = Type A Demo 1 = Type A Production 2 = Type B Demo 3 = Type B Production

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

14.2.2.43 RETURN_CODE IDTechSDK.IDT_KioskIII.device_updateDeviceFirmware (byte[ ] *firmwareData*)

Update Firmware

Updates the firmware .

Parameters

firmwareData	Signed binary data of a firmware file provided by IDTech

Returns

RETURN_CODE: Values can be parsed with errorCode.getErrorString()

After you pass the firmwareData file, a new thread will start to execute the firmware download. You will receive status of the progress through callbacks to the IDTechSDK.Callback() delegate. The following parameters will be passed back:

- sender = IDT_DEVICE_Types.IDT_DEVICE_AUGUSTA
- state = DeviceState.FirmwareUpdate
- transactionResultCode = status of the firmware update (starting, entering bootloader, applying update, block success, firmware success)
- data = File Progress. Four bytes, with bytes [0][1] = current block, and bytes [2][3] = total blocks. 0x00030010 = block 3 of 16

Example code starting a firmware update

```
OpenFileDialog diag = new OpenFileDialog();

diag.Filter = "NGA FW Files|*.fm";

if (diag.ShowDialog() == DialogResult.OK)
```

```
{

    byte[] file = File.ReadAllBytes(diag.FileName);
    RETURN_CODE rt = IDT_Device.SharedController.device_updateDeviceFirmware(file);
    if (rt == RETURN_CODE.RETURN_CODE_DO_SUCCESS)
    {

        //Was a success
    }
    else
    {
        //Error starting firmware download
    }
}
```

Example monitoring firmware update status / success

```
 private void MessageCallBack(IDTechSDK.IDT_DEVICE_Types type, DeviceState state, byte[] data,
      IDTTransactionData cardData, EMV_Callback emvCallback, RETURN_CODE transactionResultCode)
{
    switch (state)
    {
        case DeviceState.FirmwareUpdate:
            switch (transactionResultCode)
            {

                case RETURN_CODE.RETURN_CODE_FW_STARTING_UPDATE:
                    SetOutputText("Starting Firmware Update\n");
                    break;
                case RETURN_CODE.RETURN_CODE_DO_SUCCESS:
                    SetOutputText("Firmware Update Successful\n");
                    break;
                case RETURN_CODE.RETURN_CODE_APPLYING_FIRMWARE_UPDATE:
                    SetOutputText("Applying Firmware Update....\n");
                    break;
                case RETURN_CODE.RETURN_CODE_ENTERING_BOOTLOADER_MODE:
                    SetOutputText("Entering Bootloader Mode....\n");
                    break;
                case RETURN_CODE.RETURN_CODE_BLOCK_TRANSFER_SUCCESS:

                    int start = data[0] * 0x100 + data[1];
                    int end = data[2] * 0x100 + data[3];

                    SetOutputText("Sent block " + start.ToString() + " of " + end.ToString() + "\n");
                    break;
                default:
                    SetOutputText("Firmware Update Error Code: " + "0x" + String.Format("{0:X}", (ushort)
     transactionResultCode) + ": " + IDTechSDK.errorCode.getErrorString(transactionResultCode) + "\r\n");
                    break;
            }
            break;
    }
}
```

14.2.2.44 RETURN_CODE IDTechSDK.IDT_KioskIII.felica_authentication (byte[] *key*, string *ip* = " ")

FeliCa Authentication

Provides a key to be used in a follow up FeliCa Read with MAC (3 blocks max) or Write with MAC (1 block max). This command must be executed before each Read w/MAC or Write w/MAC command

NOTE: The reader must be in Pass Through Mode for FeliCa commands to work.

Parameters

key	16 byte key used for MAC generation of Read or Write with MAC
ip	IP Address of target device (optional)

Returns

RETURN_CODE: Values can be parsed with errorCode.getErrorString()

14.2.2.45 RETURN_CODE IDTechSDK.IDT_KioskIII.felica_read (byte[] *serviceCode,* int *numBlocks,* byte[] *blockList,* ref byte[] *blocks,* string *ip* = " ")

FeliCa Read

Reads up to 4 blocks.

NOTE: The reader must be in Pass Through Mode for FeliCa commands to work.

Parameters

serviceCode	Service Code List. Each service code in Service Code List = 2 bytes of data
numBlocks	Number of blocks
blockList	Blocks to read. Maximum 4 block requests
blocks	Blocks read. Each block 16 bytes.
ip	IP Address of target device (optional)

Returns

RETURN_CODE: Values can be parsed with errorCode.getErrorString()

14.2.2.46 RETURN_CODE IDTechSDK.IDT_KioskIII.felica_readWithMac (int *numBlocks,* byte[] *blockList,* ref byte[] *blocks,* string *ip* = " ")

FeliCa Read with MAC Generation

Reads up to 3 blocks with MAC Generation. FeliCa Authentication must be performed first

NOTE: The reader must be in Pass Through Mode for FeliCa commands to work.

Parameters

numBlocks	Number of blocks
blockList	Block to read. Each block in blockList Maximum 3 block requests
blocks	Blocks read. Each block 16 bytes.
ip	IP Address of target device (optional)

Returns

RETURN_CODE: Values can be parsed with errorCode.getErrorString()

14.2.2.47 RETURN_CODE IDTechSDK.IDT_KioskIII.felica_requestService (byte[] *nodeCode,* ref byte[] *response,* string *ip* = " ")

FeliCa Request Service

Perform functions a Felica Request Service

NOTE: The reader must be in Pass Through Mode for FeliCa commands to work.

Parameters

nodeCode	Node Code
response	Response as explained in FeliCA Lite-S User's Manual

Returns

RETURN_CODE: Values can be parsed with errorCode.getErrorString()

14.2.2.48 RETURN_CODE IDTechSDK.IDT_KioskIII.felica_write (byte[ ] *serviceCode,* int *blockCount,* byte[ ] *blockList,* byte[ ] *data,* ref byte[ ] *statusFlag,* string *ip* = " ")

FeliCa Write

Writes a block

NOTE: The reader must be in Pass Through Mode for FeliCa commands to work.

Parameters

serviceCode	Service Code list. Each service code must be be 2 bytes
blockCount	Block Count
blockList	Block list.
data	Block to write. Must be 16 bytes.
statusFlag	Status flag response as explained in FeliCA Lite-S User's Manual, Section 4.5
ip	IP Address of target device (optional)

Returns

RETURN_CODE: Values can be parsed with errorCode.getErrorString()

14.2.2.49 RETURN_CODE IDTechSDK.IDT_KioskIII.felica_writeWithMac (int *blockNumber,* byte[ ] *data,* string *ip* = " ")

FeliCa Write with MAC Generation

Writes a block with MAC Generation. FeliCa Authentication must be performed first

NOTE: The reader must be in Pass Through Mode for FeliCa commands to work.

Parameters

blockNumber	Number of block
data	Block to write. Must be 16 bytes.
ip	IP Address of target device (optional)

Returns

RETURN_CODE: Values can be parsed with errorCode.getErrorString()

14.2.2.50 static int IDTechSDK.IDT_KioskIII.getCommandTimeout ( ) [static]

Get Command Timeout

Gets the default timeout (in milliseconds) waiting for a blocking command response

Return values

time	Time

14.2.2.51 static void IDTechSDK.IDT_KioskIII.lcd_retrieveMessage (DisplayMessages.DISPLAY_MESSAGE_LANGUAGE *lang*, DisplayMessages.DISPLAY_MESSAGE_IDENTIFIER *id*, ref string *line1*, ref string *line2*) [static]

Retrieve LCD Message

Returns the string value for a message ID returned for LCD messaging

Parameters

lang	Language.
id	Message ID
line1	Line 1 string value
line2	Line 2 string value

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

14.2.2.52 static String IDTechSDK.IDT_KioskIII.SDK_Version ( ) [static]

SDK Version

- All Devices

Returns the current version of SDK

Returns

Framework version

14.2.2.53 static void IDTechSDK.IDT_KioskIII.setCallback (CallBack *my_Callback*) [static]

Set Callback

```
Sets the class callback
```

14.2.2.54 static void IDTechSDK.IDT_KioskIII.setCallback (IntPtr *my_Callback*, SynchronizationContext *context*) [static]

Set Callback

Sets the class callback

Parameters

my_Callback	The callback function to receive the response message from device. defined as follows. public unsafe delegate void MFCCallBack(Parameters parameters);
context	The context of the UI thread

14.2.2.55 static void IDTechSDK.IDT_KioskIII.setCommandTimeout (int *milliseconds*) [static]

Set Command Timeout

Sets the default timeout (in milliseconds) waiting for a blocking command response

Parameters

milliseconds	Time

14.2.2.56 static bool IDTechSDK.IDT_KioskIII.useSerialPort (int *port*) [static]

Use Serial Port Interface

Instructs SDK to attempt to use the Serial Port for communication with Kiosk III/IV using default baud rate

Parameters

port	Serial Port to use. Example COM1 = 1.

Returns

bool TRUE=successful, FALSE=failure

14.2.2.57 static bool IDTechSDK.IDT_KioskIII.useSerialPort (int *port,* int *baud*) [static]

Use Serial Port Interface with baud rate

Instructs SDK to attempt to use the Serial Port for communication with Kiosk III/IV

Parameters

port	Serial Port to use. Example COM1 = 1.
baud	Baud rate to override default. Example 115200;

Returns

bool TRUE=successful, FALSE=failure

14.2.2.58 static bool IDTechSDK.IDT_KioskIII.useUSB ( ) [static]

Use USB Interface

Instructs SDK to attempt to use USB for communication with Kiosk III/IV

14.2.3 Property Documentation

14.2.3.1 IDT_KioskIII IDTechSDK.IDT_KioskIII.SharedController [static], [get]

Singleton Instance

Establishes an singleton instance of IDT_KioskIII class.

Returns

Instance of IDT_KioskIII

The documentation for this class was generated from the following file:

- Source/IDT_KioskIII.cs

14.3 IDTechSDK.IDTCryptoData Class Reference

Public Attributes

- byte[ ] BDK
- byte[ ] KSN
- byte[ ] IPEK
- byte[ ] DEK
- byte[ ] DataVariant
- byte[ ] PINVariant
- byte[ ] MACVariant
- bool isTDES
- int keyVariant
- bool isDecryption
- byte[ ] dataToProcess
- byte[ ] dataResults
- byte[ ] pinBlock
- byte[ ] clearPinBlock
- string PAN
- string PIN
- int PINBlockType
- string errorString
- bool MAC_Command

14.3.1 Detailed Description

Class used when Encrypting/Decrypting DUKPT data Used in Common.processDUKPT(ref IDTCryptoData data)

14.3.2 Member Data Documentation

14.3.2.1 byte [ ] IDTechSDK.IDTCryptoData.BDK

Base Derivation Key.

14.3.2.2 byte [ ] IDTechSDK.IDTCryptoData.clearPinBlock

Decrypted Pin Block

14.3.2.3 byte [ ] IDTechSDK.IDTCryptoData.dataResults

Data that has been Decrypted (isDecryption = TRUE), or Data that has been ecrypted (isDecryption = FALSE), or Data that has been MAC (isMAC_Command = TRUE)

14.3.2.4 byte [ ] IDTechSDK.IDTCryptoData.dataToProcess

Data to encrypt (isDecryption = false) or data to decrypt (isDecryption = true)

14.3.2.5 byte [ ] IDTechSDK.IDTCryptoData.DataVariant

Data Encryption Key (variant of DEK).

14.3.2.6 byte [ ] IDTechSDK.IDTCryptoData.DEK

Derived Encryption Key.

14.3.2.7 string IDTechSDK.IDTCryptoData.errorString

Encryption/Decryption Error.

14.3.2.8 byte [ ] IDTechSDK.IDTCryptoData.IPEK

Initial Public Encryption Key.

14.3.2.9 bool IDTechSDK.IDTCryptoData.isDecryption

TRUE = Decrypt Data. FALSE = Encrypt Data

14.3.2.10 bool IDTechSDK.IDTCryptoData.isTDES

TRUE = Use TDES. FALSE = Use AES

14.3.2.11 int IDTechSDK.IDTCryptoData.keyVariant

0 = Use Data Variant. 1 = Use PIN Variant. 2 = Use MAC Variant

14.3.2.12 byte [ ] IDTechSDK.IDTCryptoData.KSN

Key Serial Number.

14.3.2.13 bool IDTechSDK.IDTCryptoData.MAC_Command

FALSE = Don't MAC (use encryption/decryption setting), TRUE = Return MAC (override encyrption/decryption setting)

14.3.2.14 byte [ ] IDTechSDK.IDTCryptoData.MACVariant

Message Authentication Challenge Key (variant of DEK).

14.3.2.15 string IDTechSDK.IDTCryptoData.PAN

Primary Account Number used with clearPinBlock to derive/encode PIN

14.3.2.16 string IDTechSDK.IDTCryptoData.PIN

PIN derived from clearPinBlock, or used to create clearPinBlock

14.3.2.17 byte [ ] IDTechSDK.IDTCryptoData.pinBlock

Encrypted Pin Block

14.3.2.18 int IDTechSDK.IDTCryptoData.PINBlockType

PIN Block Type. TDES can be 0 or 3. AES will be 4.

14.3.2.19 byte [ ] IDTechSDK.IDTCryptoData.PINVariant

PIN Encryption Key (variant of DEK).

The documentation for this class was generated from the following file:

- Source/IDT_Transactions.cs

14.4 IDTechSDK.IDTTransactionData Class Reference

Static Public Member Functions

- static void **setTransactionAttributes** (byte attribute, ref IDTTransactionData data, byte attribute2=0)

Public Attributes

- EVENT_TRANSACTION_DATA_Types Event
- EVENT_NOTIFICATION_Types Notification
- byte[ ] msr_rawData
- byte[ ] msr_encTrack1
- byte[ ] msr_encTrack2
- byte[ ] msr_encTrack3
- String msr_track1
- String msr_track2
- String msr_track3

- String device_RSN
- byte[] msr_KSN
- int msr_track1Length
- int msr_track2Length
- int msr_track3Length
- CAPTURE_ENCODE_TYPE msr_cardType
- byte msr_captureEncodeStatus
- CAPTURE_ENCRYPT_TYPE captureEncryptType
- CAPTURE_ENCRYPT_TYPE captureEncryptTypeEMV
- CAPTURE_CARD_TYPE captureCardType
- int msr_errorCode
- int emv_rfStateCode
- int iccPresent
- byte[] msr_sessionID
- byte[] msr_hashTrack1
- byte[] msr_hashTrack2
- byte[] msr_hashTrack3
- KEY_VARIANT_TYPE msr_keyVariantType
- byte[] msr_extendedField
- int isCTLS
- CTLS_APPLICATION ctlsApplication
- byte[] emv_clearingRecord
- byte[] emv_encryptedTags
- byte[] emv_unencryptedTags
- EMV_RESULT_CODE emv_resultCode
- byte[] emv_maskedTags
- byte[] emv_encipheredOnlinePIN
- bool emv_hasAdvise
- bool emv_hasReversal
- string pin_pinblock
- string pin_KSN
- string pin_KeyEntry
- byte SW1
- byte SW2
- byte[] mac
- byte[] macKSN
- bool hasMACVerificationData
- TRANS_ERROR_CODE emv_transaction_Error_Code
- RF_STATE emv_RF_State
- EXTENDED_STATUS_CODES emv_ESC
- CEMV_APP_ERROR_FN emv_appErrorFn
- CEMV_APP_ERROR_STATE emv_appErrorState
- byte[] captured_PAN
- byte[] captured_KSN
- string captured_firstPANDigits
- string captured_lastPANDigits
- byte[] captured_Expiry
- byte[] captured_CSC
- bool captured_SHA256
- byte[] captured_MACValue
- byte[] captured_MACKSN
- byte[] captured_InitialVector

14.4.1 Detailed Description

Class for swipe data

14.4.2 Member Data Documentation

14.4.2.1 CAPTURE_CARD_TYPE IDTechSDK.IDTTransactionData.captureCardType

Get the captured card type,please see CAPTURE_CARD_TYPE for more information.
CAPTURE_CARD_TYPE_UNKNOWN;
CAPTURE_CARD_TYPE_CONTACT;
CAPTURE_CARD_TYPE_CTLS_EMV;
CAPTURE_CARD_TYPE_CTLS_MSD;
CAPTURE_CARD_TYPE_MSR;

14.4.2.2 byte [ ] IDTechSDK.IDTTransactionData.captured_CSC

Captured Customer Service Code

14.4.2.3 byte [ ] IDTechSDK.IDTTransactionData.captured_Expiry

Captured Expiry Date

14.4.2.4 string IDTechSDK.IDTTransactionData.captured_firstPANDigits

First plaintext PAN Digits

14.4.2.5 byte [ ] IDTechSDK.IDTTransactionData.captured_InitialVector

This initial vector is used for all encryptions in this command. If encryption is off thisfield will be filled with zeros (00h).

14.4.2.6 byte [ ] IDTechSDK.IDTTransactionData.captured_KSN

KSN used to encrypt manually captured PAN from keyed input

14.4.2.7 string IDTechSDK.IDTTransactionData.captured_lastPANDigits

Last plaintext PAN digits

14.4.2.8 byte [ ] IDTechSDK.IDTTransactionData.captured_MACKSN

KSN for MAC DUKPT key.

14.4.2.9 byte [ ] IDTechSDK.IDTTransactionData.captured_MACValue

Authenticate message from “Initial Vector” field to “MAC Value Length” field

14.4.2.10 byte [ ] IDTechSDK.IDTTransactionData.captured_PAN

Manually captured PAN from keyed input

14.4.2.11 bool IDTechSDK.IDTTransactionData.captured_SHA256

TRUE = SHA-256, FALSE = SHA-1

14.4.2.12 CAPTURE_ENCRYPT_TYPE IDTechSDK.IDTTransactionData.captureEncryptType

Get the encrypted type,please see CAPTURE_ENCRYPT_TYPE for more information.
CAPTURE_ENCRYPT_TYPE_UNKNOWN;
CAPTURE_ENCRYPT_TYPE_TDES;
CAPTURE_ENCRYPT_TYPE_AES;
CAPTURE_ENCRYPT_TYPE_NONE;
CAPTURE_ENCRYPT_TRANS_ARMOR_PKI;
CAPTURE_ENCRYPT_VOLTAGE;
CAPTURE_ENCRYPT_VISA_FPE;
CAPTURE_ENCRYPT_VERIFONE_FPE;
CAPTURE_ENCRYPT_DESJARDIN

14.4.2.13 CAPTURE_ENCRYPT_TYPE IDTechSDK.IDTTransactionData.captureEncryptTypeEMV

Get the encrypted type for EMV,please see CAPTURE_ENCRYPT_TYPE for more information.
CAPTURE_ENCRYPT_TYPE_UNKNOWN;
CAPTURE_ENCRYPT_TYPE_TDES;
CAPTURE_ENCRYPT_TYPE_AES;
CAPTURE_ENCRYPT_TYPE_NONE;
CAPTURE_ENCRYPT_TRANS_ARMOR_PKI;
CAPTURE_ENCRYPT_VOLTAGE;
CAPTURE_ENCRYPT_VISA_FPE;
CAPTURE_ENCRYPT_VERIFONE_FPE;
CAPTURE_ENCRYPT_DESJARDIN

14.4.2.14 CTLS_APPLICATION IDTechSDK.IDTTransactionData.ctlsApplication

CTLS Application

14.4.2.15 String IDTechSDK.IDTTransactionData.device_RSN

Get the Reader Serial Number.

14.4.2.16 CEMV_APP_ERROR_FN IDTechSDK.IDTTransactionData.emv_appErrorFn

EMV App Error Function (select AR products)

14.4.2.17 CEMV_APP_ERROR_STATE IDTechSDK.IDTTransactionData.emv_appErrorState

EMV App Error State (select AR products)

14.4.2.18 byte [ ] IDTechSDK.IDTTransactionData.emv_clearingRecord

clearing record TLV

14.4.2.19 byte [ ] IDTechSDK.IDTTransactionData.emv_encipheredOnlinePIN

enciphered Online Pin

14.4.2.20 byte [ ] IDTechSDK.IDTTransactionData.emv_encryptedTags

Encrypted Tags TLV

14.4.2.21 EXTENDED_STATUS_CODES IDTechSDK.IDTTransactionData.emv_ESC

Extended Status Code (select AR products)

14.4.2.22 bool IDTechSDK.IDTTransactionData.emv_hasAdvise

Advise

14.4.2.23 bool IDTechSDK.IDTTransactionData.emv_hasReversal

Reversal

14.4.2.24 byte [ ] IDTechSDK.IDTTransactionData.emv_maskedTags

Masked Tags TLV

14.4.2.25 EMV_RESULT_CODE IDTechSDK.IDTTransactionData.emv_resultCode

EMV Result Code

14.4.2.26 RF_STATE IDTechSDK.IDTTransactionData.emv_RF_State

RF_State (select AR products)

14.4.2.27 int IDTechSDK.IDTTransactionData.emv_rfStateCode

For some Error Codes, the RF State Code indicates the exact Reader-Card command that failed. This helps determine the exact place where the failure occurred.

14.4.2.28 TRANS_ERROR_CODE IDTechSDK.IDTTransactionData.emv_transaction_Error_Code

Transaction Error Code (select AR products)

14.4.2.29 byte [ ] IDTechSDK.IDTTransactionData.emv_unencryptedTags

Unencrypted Tags TLV

14.4.2.30 EVENT_TRANSACTION_DATA_Types IDTechSDK.IDTTransactionData.Event

Transaction Data type,please see EVENT_TRANSACTION_DATA_Types for more information.

14.4.2.31 bool IDTechSDK.IDTTransactionData.hasMACVerificationData

Existance of MAC Verification Data for Encrypted Data

14.4.2.32 int IDTechSDK.IDTTransactionData.iccPresent

Get the swiped card ICC Status.
0 = Unknown 1 = True 2 = False

14.4.2.33 int IDTechSDK.IDTTransactionData.isCTLS

Track data was captured via CTLS interface 0 = Unknown 1 = True 2 = False

14.4.2.34 byte [ ] IDTechSDK.IDTTransactionData.mac

Message Authentication Code

14.4.2.35 byte [ ] IDTechSDK.IDTTransactionData.macKSN

Message Authentication Code Key Serial Number

14.4.2.36 byte IDTechSDK.IDTTransactionData.msr_captureEncodeStatus

Get the swiped card decoded status.
0x00:decoded data success;
Bit0:1-track1 data error;
Bit1:1-track2 data error;
Bit2:1-track3 data error;
Bit3:1-track1 encrypted data error;
Bit4:1-track2 encrypted data error;
Bit5:1-track3 encrypted data error;
Bit6:1-KSN error;

14.4.2.37 CAPTURE_ENCODE_TYPE IDTechSDK.IDTTransactionData.msr_cardType

Get the swiped card type,please see CAPTURE_ENCODE_TYPE for more information.
MSR card type:
CAPTURE_ENCODE_TYPE_ISOABA:ISO/ABA format
CAPTURE_ENCODE_TYPE_AAMVA:AAMVA format
CAPTURE_ENCODE_TYPE_Other:Other
CAPTURE_ENCODE_TYPE_Raw:Raw; undecoded format
CAPTURE_ENCODE_TYPE_JisI_II:JIS I or JIS II

14.4.2.38 byte [ ] IDTechSDK.IDTTransactionData.msr_encTrack1

Get the swiped card Track1 encrypted data.
A byte array containing Track1 encrypted data.

14.4.2.39 byte [ ] IDTechSDK.IDTTransactionData.msr_encTrack2

Get the swiped card Track2 encrypted data.
A byte array containing Track2 encrypted data.

14.4.2.40 byte [ ] IDTechSDK.IDTTransactionData.msr_encTrack3

Get the swiped card Track3 encrypted data.
A byte array containing Track3 encrypted data.

14.4.2.41 int IDTechSDK.IDTTransactionData.msr_errorCode

Contains error code when data is not returned

14.4.2.42 byte [ ] IDTechSDK.IDTTransactionData.msr_extendedField

Extended Field Data. Byte 0: 1 = Hash-SHA256

14.4.2.43 byte [ ] IDTechSDK.IDTTransactionData.msr_hashTrack1

Get the swiped card Track1 hash data.
A byte array containing Track1 hash data.

14.4.2.44 byte [ ] IDTechSDK.IDTTransactionData.msr_hashTrack2

Get the swiped card Track2 hash data.
A byte array containing Track2 hash data.

14.4.2.45 byte [ ] IDTechSDK.IDTTransactionData.msr_hashTrack3

Get the swiped card Track3 hash data.
A byte array containing Track3 hash data.

14.4.2.46 KEY_VARIANT_TYPE IDTechSDK.IDTTransactionData.msr_keyVariantType

KEY_VARIANT_TYPE_DATA = Data Variant key used
KEY_VARIANT_TYPE_PIN = PIN Variant key used

14.4.2.47 byte [ ] IDTechSDK.IDTTransactionData.msr_KSN

Get the swiped card KSN (Key Serial Number).
A byte array containing 10 bytes.

14.4.2.48 byte [ ] IDTechSDK.IDTTransactionData.msr_rawData

Get the card data raw data.
Containing complete unparsed transaction data as received from device.

14.4.2.49 byte [ ] IDTechSDK.IDTTransactionData.msr_sessionID

Get the swiped card Session ID.
A byte arrary to get session ID, if exists.

14.4.2.50 String IDTechSDK.IDTTransactionData.msr_track1

Get the swiped card Track1 data.
A string containing Track1 masked data expressed as hex characters.

14.4.2.51 int IDTechSDK.IDTTransactionData.msr_track1Length

Get the swiped card length of Track1 data.

14.4.2.52 String IDTechSDK.IDTTransactionData.msr_track2

Get the swiped card Track2 data.
A string containing Track2 masked data expressed as hex characters.

14.4.2.53 int IDTechSDK.IDTTransactionData.msr_track2Length

Get the swiped card length of Track2 data.

14.4.2.54 String IDTechSDK.IDTTransactionData.msr_track3

Get the swiped card Track3 data.
A string containing Track3 masked data expressed as hex characters.

14.4.2.55 int IDTechSDK.IDTTransactionData.msr_track3Length

Get the swiped card length of Track3 data.

14.4.2.56 EVENT_NOTIFICATION_Types IDTechSDK.IDTTransactionData.Notification

Event Notification type,please see EVENT_NOTIFICATION_Types for more information.

14.4.2.57 string IDTechSDK.IDTTransactionData.pin_KeyEntry

KSN for Pinblock

14.4.2.58 string IDTechSDK.IDTTransactionData.pin_KSN

KSN for Pinblock

14.4.2.59 string IDTechSDK.IDTTransactionData.pin_pinblock

PIN block from PINPAD

14.4.2.60 byte IDTechSDK.IDTTransactionData.SW1

SW1

14.4.2.61 byte IDTechSDK.IDTTransactionData.SW2

SW2

The documentation for this class was generated from the following file:

- Source/IDT_Transactions.cs

Index