Windows SDK Guide

for NEO2

#80152503-001

Rev. C

Revision History

Revision	Description and Reason for Change	Date
A	Initial Release - Manual;User;NEO2;SDK;Windows	2/08/2016
A	Updated UWP Implementation Information	9/16/2016
C	Updated TLV Information	2/11/2018

Contents

Chapter 1

IDTech Windows SDK Reference Guide for NEO2 Device Family

IDTechSDK.dll is a Windows dynamic link libraries that will be provided by IDTech as the main interface between Windows Forms (WinForms) and applications, respectively, the NEO2 family of devices 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 or UWP 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

Main Transaction Commands

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

3.1 EMV Methods

Start EMV Transaction

IDTechSDK::IDT_NEO2::emv_startTransaction()

Begins an amount authorization request with the ICC. Returns authorization decision (approved, denied, or go online) in callback method.

Authenticate EMV Transaction

IDTechSDK::IDT_NEO2::emv_authenticateTransaction()

By default, auto-authenticate is ON and this step does not need to be performed. If auto-authenticate is OFF (IDTechSDK::IDT_NEO2::emv_autoAuthenticate), when the results to IDTechSDK::IDT_NEO2::emv_startTransaction() come back as EMV_RESULT_CODE.EMV_RESULT_CODE_AUTHENTICATE_TRANSACTION, this method must be called to continue the EMV transaction.

Complete Online EMV Transaction

IDTechSDK::IDT_NEO2::emv_completeTransaction()

After receiving a host response, pass host tags (minimum 8A Authorization Response Code) as a parameter.

If there was a communication error with host, you must still finish the EMV transaction by passing "TRUE" for commError.

Terminal Configuration

IDTechSDK::IDT_NEO2::emv_retrieveTerminalData()
IDTechSDK::IDT_NEO2::emv_setTerminalData()

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

AID Management

IDTechSDK::IDT_NEO2::emv_retrieveApplicationData
IDTechSDK::IDT_NEO2::emv_removeApplicationData()
IDTechSDK::IDT_NEO2::emv_removeAllApplicationData()
IDTechSDK::IDT_NEO2::emv_setApplicationData()
IDTechSDK::IDT_NEO2::emv_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_NEO2::emv_retrieveCAPK()
IDTechSDK::IDT_NEO2::emv_removeCAPK()
IDTechSDK::IDT_NEO2::emv_removeAllCAPK()
IDTechSDK::IDT_NEO2::emv_setCAPK()
IDTechSDK::IDT_NEO2::emv_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

CRL Management

IDTechSDK::IDT_NEO2::emv_removeCRL()
IDTechSDK::IDT_NEO2::emv_removeAllCRL()
IDTechSDK::IDT_NEO2::emv_retrieveCRLList()
IDTechSDK::IDT_NEO2::emv_setCRL:()

Methods for Certificate Revocation List management.

Kernel Version

IDTechSDK::IDT_NEO2::emv_getEMVKernelVersion()

Method to retrieve kernel version. Valued returned in IDTResult.data

APDU Communication

IDTechSDK::IDT_NEO2::device_setPassThroughMode:()
IDTechSDK::IDT_NEO2::icc_powerOnICC:()
IDTechSDK::IDT_NEO2::icc_powerOffICC:()

IDTechSDK::IDT_NEO2::icc_exchangeAPDU:()
Allows the direct sending of APDU packets to ICC. Pass through mode must first be enabled. Then Power On needs to complete successfully. Then APDU packet exchange can take place

3.2 MSR/CTLS

Request Swipe or Tap

IDTechSDK::IDT_NEO2::msr_startMSRSwipe()
IDTechSDK::IDT_NEO2::ctls_startTransaction()
Both methods perform identical operation. Enables MSR to receive Swipe and CTLS to receive tap.

Cancel Swipe or Tap

IDTechSDK::IDT_NEO2::msr_cancelMSRSwipe()
IDTechSDK::IDT_NEO2::ctls_cancelTransaction()
Both methods perform identical operation. Cancels the Swipe/Tap request.

3.3 All Interfaces

Request any interface (Swipe, Tap, Insert)

IDTechSDK::IDT_NEO2::device_startTransaction()
This method will allow any interface to capture data.

Chapter 4

Sending Direct Commands

The main purpose of DLL for NEO2 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 contact EMV Transactions, perform swipe/Tap transactions, and to modify contact EMV data files.

The NEO2 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. The DLL uses the following the command to send Protocol 2 commands to NEO2:

IDTechSDK::IDT_NEO2::device_sendVivoCommandP2()

Any function not supported by the SDK can be sent with the sendVivoCommandP2.

Chapter 5

Connecting to NEO2

Most NEO2 devices connects through USB, Serial Interface (COM), or IP depending on hardware capabnilities.

5.1 Connect with USB:

All NEO2 devices use 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 SpectrumPro, you would execute the following command during program initialization to establish a connection:

```
IDT_NEO2.useUSB();
```

5.2 Connect with Serial Interface (COM)

Most NEO2 devices can connect via Serial Interface. The default serial port settings are as follows:

- Speed: 115,200
- Bits: 8
- Stop Bit: 1
- Parity: None

To inform the SDK you are using the Serial Interface of a NEO2 device, 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 non-SRED device:

```
IDT_NEO2.useSerialPort(1, false);
```

If you change the default baud rate, then there is an overloaded method to also include the baud rate:

```
IDT_NEO2.useSerialPort(1, 115200, false);
```

5.3 Connect with TCP/IP

Some NEO2 devices can connect via a network connection. The default port is is 1025.

To inform the SDK you are using the IP interface of a NEO2 device, you would execute the following command during program initialization to establish a connection by passing the correct address. In the following example, we are connecting to a device as IP address 10.0.1.1:

```
IDT_NEO2.ip_connectToSocket("10.0.1.1")
```

The IP address can optionally contain a port, separated by a colon:

```
IDT_NEO2.ip_connectToSocket("10.0.1.1:1025")
```

The IP address can optionally contain a description, separated by hash tag:

```
IDT_NEO2.ip_connectToSocket("10.0.1.1#Description")
```

The IP address can optionally contain both a port and a description:

```
IDT_NEO2.ip_connectToSocket("10.0.1.1:1025#Description")
```

There can be multiple IP connected devices to a single SDK instance. Each one is initially connected and registered with ip_connectToSocket.

To get a listing of all registered devices the SDK has access to, a listing of all the strings passed to ip_connectTo↩ Socket can be retrieved by the following

```
List<string> devices = IDT_NEO2.ip_getSocketList();
```

To switch between devices listed in ip_getSocketList():

```
IDT_NEO2.ip_switchToSocket("10.0.1.1");
```

NOTE: the string passed to ip_switchToSocket must match the complete gateway string as originally defined, and reported by ip_getSocketList.'

5.4 Auto-connect with TCP/IP

The IDTechSDK.dll framework uses a data file called NEO2_Devices.xml that resides in the same location. This file provides information for automatically connecting to NEO2 devices via USB or TCP/IP. The file is comprised of Device entries as follows:

```
<Device>
    <Product>VP6800</Product>
    <Name>VP6800</Name>
    <VID>0x0acd</VID>
    <PID>0x4460</PID>
    <UsageID>0x0001</UsageID>
    <UsagePage>0xFF00</UsagePage>
    <BaseIP>10.12.34.98</BaseIP>
    <IP_Count>2</IP_Count>
    <IP_Port></IP_Port>
</Device>
```

- BaseIP = IP address of first device to look for
- IP_Count - # of consecutive IP address to search, up to an IP address of xx.xx.xx.255
- IP_Port - Optional. Defaults to port 1025

Multiple device connections can be defined by a single Device entry, if those devices will use consecutive IP addresses, or multiple device connections can be defined by multiple Device entries, each with unique IP.

Every IP connection attempt is reported to a callback as connection successful, or connection could not be established

To activate autoConnect when NEO2_Devices.xml is properly configured:

```
IDT_NEO2.ip_autoConnectToSocket();
```

5.5 Monitor Disconnect and Connect over TCP/IP

The SDK has a method that can ping any of it's known connected devices on a regular interval, and report when it is not longer reachable via a callback. It can also be configured to execute ip_autoConnectToSocket() on a regular interval to detect devices connected on a regular interval:

```
public static bool ip_monitorSocketConnectionStatus(bool enable, bool monitorConnect, int interval, int
      retryCount)
```

- enable TRUE = enable polling for device disconnect and optional connect, at specified interval and retryCount
- monitorConnect If TRUE, and auto-connect attempt will be made at the specified interval
- interval Interval, in seconds, to monitor IP. Minimum value 5 seconds
- retryCount Number of retries trying to communicate before determining device not available and issue disconnect notification.

Chapter 6

Core Implementation: WinForms

IDTechSDK.dll includes class libraries to interface with the NEO2. 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 functions
- Initialize NEO2
- 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 4.0 or greater. Please make sure your final application installer checks for this dependency and installs it if not on the destination system.

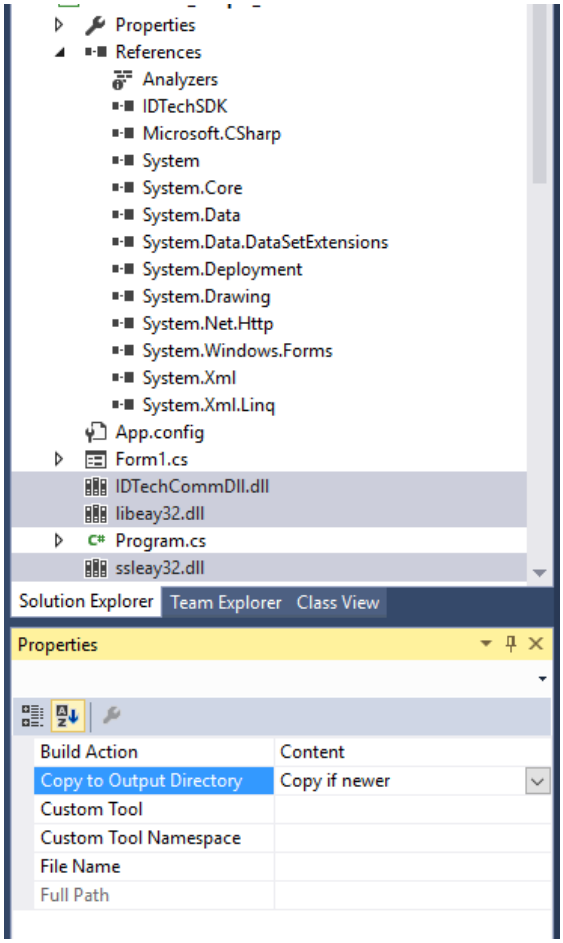
In addition, the following libraries need to be added to project folder and included in with the application distribution:

- NEO2_config.dll
- NEO2_ctls.dll
- NEO2_device.dll
- NEO2_emv.dll
- NEO2_icc.dll
- NEO2_msr.dll
- NEO2_parse.dll

- NEO2_lcd.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 functions

There are currently two callback functions defined, one for button press callback, and another universal callback for all other requirements. The universal callback uses DeviceState to determine which action to take. For IP connected devices, the last parameter is provided for instance where multiple IP connected devices may be sending data back to this callback.

```
private void MessageCallBack(IDTechSDK.IDT_DEVICE_Types type, DeviceState state, byte[] data,
      IDTTransactionData cardData, EMV_Callback emvCallback, RETURN_CODE transactionResultCode, string IP)
{
    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 device timeout
            SetOutputText("Callback:DeviceTimeout\n");
            break;
        case DeviceState.TransactionFailed:
            //Transaction failed to complete
            break;
        case DeviceState.EMVCallback:
            SetOutputText("EMV CAllback\n");
            break;
        case DeviceState.PINpadKeypress:
            SetOutputText("PINPad Key was pressed\n");
            break;
        case DeviceState.PINTimeout:
            SetOutputText("PINPad Timeout\n");
            break;
        case DeviceState.MenuTimeout:
            SetOutputText("Menu Entry Timeout\n");
            break;
        case DeviceState.TransactionIdle:
            SetOutputText("Ready For Transaction\n");
            break;
        case DeviceState.Signature:
            SetOutputText("Signature Captured\n");
            break;
        case DeviceState.FirmwareUpdate:
            SetOutputText("firmware update\n");
            break;
        case DeviceState.FileTransfer:
            SetOutputText("File Transfer\n");
            break;
        case DeviceState.ConfigurationGroup:
            SetOutputText("Config Group\n");
            break;
        case DeviceState.InvalidInterface:
            SetOutputText("Invalid Interface\n");
            break;
        case DeviceState.FunctionKey:
            SetOutputText("Function Key Pressed\n");
```

```
            break;
        case DeviceState.PINFail:
            SetOutputText("PIN Fail\n");
            break;
        case DeviceState.SocketEstablished:
            SetOutputText("Socket connection established to: " + IP + " \r\n");
            break;
        case DeviceState.SocketFailed:
            SetOutputText("Socket connection failed to: " + IP + " \r\n");
            break;
        case DeviceState.ButtonEvent:
            SetOutputText("Screen ID: " + screenID.ToString() + "\r\n");
            SetOutputText("Button ID: " + ID.ToString() + "\r\n");
            break;

        default:
            break;

    }


}
```

When lcd_addButton is executed, a buttonCallback can be passed that will report all press events for that button. If buttonCallback is set to NULL, then all press events will be returned to the main callback as a DeviceState.Button↩ Event. The buttonCallback has the following signature, passing back the device type, the screen, the button ID, and the device IP address:

```
private void ButtonCallback(IDT_DEVICE_Types sender, uInt16 screenID, uInt16 itemID, string IP)
{
    SetOutputText("BUTTON CALLBACK, IP : " + IP + "\n");
    SetOutputText("Button Pressed, Screen ID: " + screenID.ToString() + "\n");
    SetOutputText("Button Pressed, Button ID: " + itemID.ToString() + "\n");

}
```

For the universal callback, there is an OPTIONAL parameter to associate that callback with just a single IP address.

Setting button callback is established during lcd_addButton, in addition, there is another method to set the callback of any existing button.

6.5 Initialize NEO2:

A Singleton instance has been established in the IDT_NEO2 class. Establish the callback, and then access the NEO2 through USB via the SharedController instance. Or, you can establish through Serial Port or IP if available for your device

```
public NEO2_Simple_Demo()
{
    InitializeComponent();
    IDT_NEO2.setCallbackIP(MessageCallBack);
    IDT_NEO2.useUSB();  //connect via USB
    // bool isConnected = IDT_NEO2.useSerialPort(1); //use serial port 1
    // IDT_NEO2.ip_connectToSocket("10.0.1.1");  //connect to socket 10.0.1.1

}
```

6.6 Sample Project Tutorial

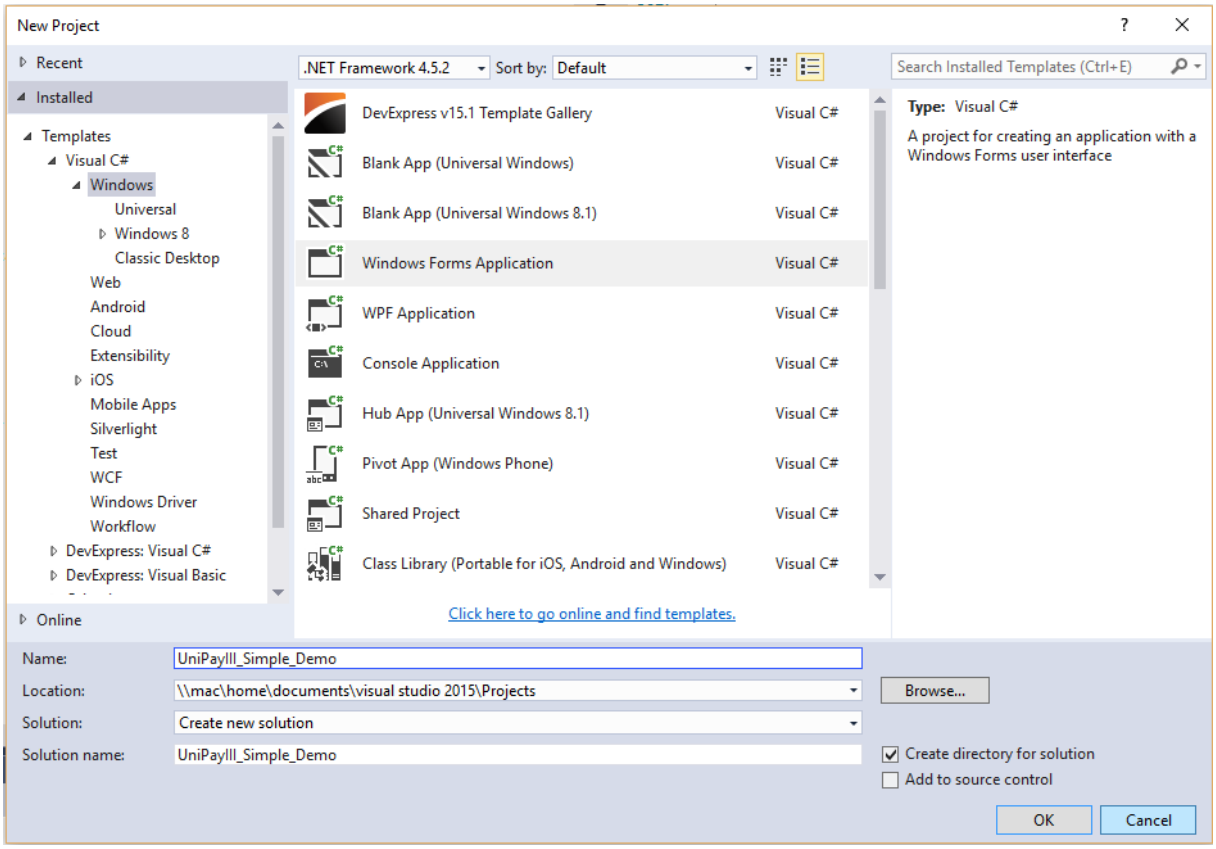
Using Visual Studio 2015, we will create a C# sample project that will interface with the NEO2 and will perform the following activities:

- Get firmware version
- Display ICC Status (card inserted/removed, ICC power on/off)
- Turn on/off ICC reader

- Get/Set terminal data file
- Get/Set AID for Visa
- Get/Set CAPKs for Visa
- Perform EMV Transaction
- Start/Stop a Swipe/Tap Request
- Show log of all data going to/from NEO2

6.6.1 Step 1: Create New Project

Create a new Windows Form Application in Visual Studio. In our example, we use project name NEO2_Simple_↩
Demo



6.6.2 Step 2: Import Libraries

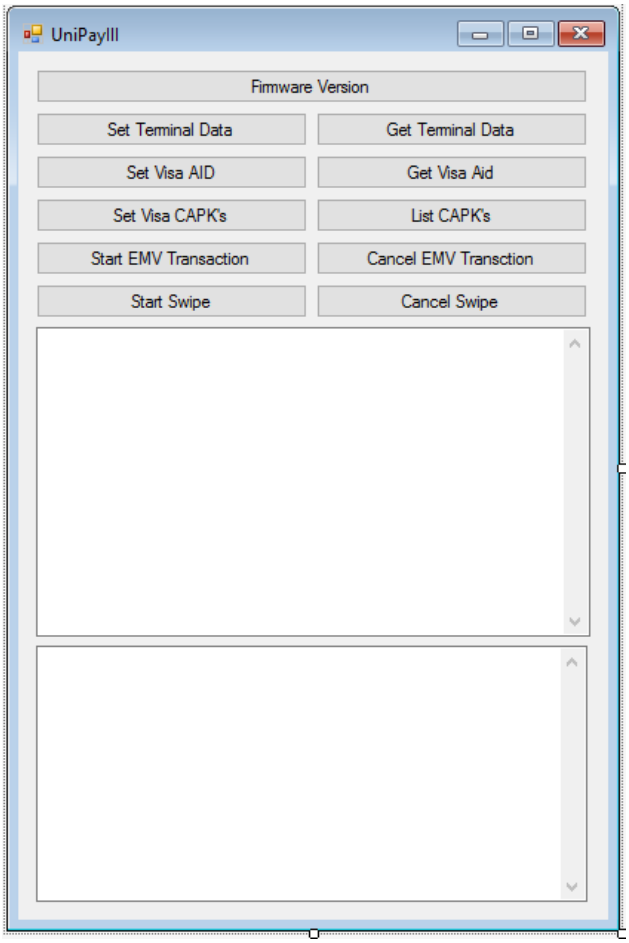
Import the necessary libraries

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:

  - Show Firmware
  - Set Terminal Data
  - Get Terminal Data
  - Set Visa AID
  - Get Visa AID
  - Set Visa CAPKs
  - List All CAPKs
  - Start EMV Transaction
  - Cancel EMV Transaction
  - Start Swipe/Tap
  - Cancel Swipe/Tap
- Add a text box to communicate data from the NEO2.
- Add a text box for the log of NEO2.



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 NEO2 singleton object in the class initializer. Reference: Initialize NEO2
- 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_NEO2.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_NEO2.SharedController.emv_startTransaction(1.00, 0, 0, 30, null, false,
      additionalTags);
    if (rt == RETURN_CODE.RETURN_CODE_DO_SUCCESS)
    {
        tbOutput.AppendText("Start EMV Successful\r\n");
    }
    else
    {
        tbOutput.AppendText("Start EMV 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_NEO2.SharedController.emv_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));
    }
}

private void btnSetTerminalData_Click(object sender, EventArgs e)
{
    byte[] term = Common.getByteArray("
      5f3601029f1a0208409f3501229f330360f8c89f4005f00000a0019f1e085465726d6
      96e616c9f150212349f160f30303030303030303030303030309f1c0838373635343332319f4e22313037323120576 16c6b6572205
      3742e20437970726573732c204341202c5553412edf260101df1008656e667265737a68df110101df270100dfee150101dfee160100d
      fee170107dfee180180dfee1e08f0dc3cf0c29e9400dfee1f0180dfee1b0830303031353130 30dfee20013cdfee21010adfee2203323c3c");
    RETURN_CODE rt = IDT_NEO2.SharedController.emv_setTerminalData(term);
    if (rt == RETURN_CODE.RETURN_CODE_DO_SUCCESS)
    {

        tbOutput.AppendText("Set Terminal Successful:" + "  \r\n");


    }
    else
    {
        tbOutput.AppendText("Save Terminal failed Error Code: " + "0x" + String.Format("{0:X}", (ushort)rt)
       + ": " + IDTechSDK.errorCode.getErrorString(rt) + "\r\n");
    }
}

private void btnGetTerminalData_Click(object sender, EventArgs e)
{
    byte[] tlv = null;
    RETURN_CODE rt = IDT_NEO2.SharedController.emv_retrieveTerminalData(ref tlv);
    if (rt == RETURN_CODE.RETURN_CODE_DO_SUCCESS)
    {
        tbOutput.AppendText("Retrieve Terminal Successful- TLV: " + "  \r\n" + Common.getHexStringFromBytes
      (tlv) + "\r\n");
    }
    else
    {
        tbOutput.AppendText("Retrieve Terminal failed Error Code: " + "0x" + String.Format("{0:X}", (ushort
      )rt) + ": " + IDTechSDK.errorCode.getErrorString(rt) + "\r\n");
    }
```

```
}

private void btnSetVisaAid_Click(object sender, EventArgs e)
{
    RETURN_CODE rt;
    byte[] name = Common.getByteArray("a0000000031010");
    byte[] aid = Common.getByteArray("
      9f01065649534130305f5701005f2a0208409f090200965f3601029f1b0400003a98df
      25039f3704df28039f0802dfee150101df13050000000000df14050000000000df15050000000000df180100df170400002710df190100");
    rt = IDT_NEO2.SharedController.emv_setApplicationData(name, aid);
    if (rt == RETURN_CODE.RETURN_CODE_DO_SUCCESS)
    {
        tbOutput.AppendText("Default AID Successful\r\n");
    }
    else
    {
        tbOutput.AppendText("Default AID failed Error Code: " + "0x" + String.Format("{0:X}", (ushort)rt) +
       ": " + IDTechSDK.errorCode.getErrorString(rt) + "\r\n");
    }
}

private void btnGetVisaAid_Click(object sender, EventArgs e)
{
    byte[] tlv = null;
    RETURN_CODE rt = IDT_NEO2.SharedController.emv_retrieveApplicationData(Common.getByteArray("
      a0000000031010"), ref tlv);
    if (rt == RETURN_CODE.RETURN_CODE_DO_SUCCESS)
    {
        tbOutput.AppendText("Retrieve AID Successful- TLV: " + "  \r\n" + Common.getHexStringFromBytes(tlv)
      + "  \r\n");
    }
    else
    {
        tbOutput.AppendText("Retrieve AID failed Error Code: " + "0x" + String.Format("{0:X}", (ushort)rt)
      + ": " + IDTechSDK.errorCode.getErrorString(rt) + "\r\n");
        System.Diagnostics.Debug.WriteLine("Retrieve AID failed Error Code: " + "0x" + String.Format(
      "{0:X}", (ushort)rt));
    }
}

private void btnSetVisaCAPKs_Click(object sender, EventArgs e)
{
    byte[] capk = Common.getByteArray("
      a000000003500101b769775668cacb5d22a647d1d993141edab7237b000100018000d
      11197590057b84196c2f4d11a8f3c05408f422a35d702f90106ea5b019bb28ae607aa9cdebcd0d81a38d48c7ebb0062d287369ec0c42
      124246ac30d80cd602ab7238d51084ded4698162c59d25eac1e66255b4db2352526ef0982c3b8ad3d1cce85b01db5788e75e09f44be7361360
    RETURN_CODE rt = IDT_NEO2.SharedController.emv_setCAPK(capk, null);
    capk = Common.getByteArray("
      a000000003510101b9d248075a3f23b522fe45573e04374dc4995d71000000039000db5fa29d
      1fda8c1634b04dccff148abee63c772035c79851d3512107586e02a917f7c7e885e7c4a7d529710a145334ce67dc412cb1597b77aa25
      43b98d19cf2cb80c522bdbea0f1b113fa2c86216c8c610a2d58f29cf3355ceb1bd3ef410d1edd1f7ae0f16897979de28c6ef293e0a19282bd1
    rt = IDT_NEO2.SharedController.emv_setCAPK(capk, null);
    capk = Common.getByteArray("
      a000000003530101ac213a2e0d2c0ca35ad0201323536d58097e4e5700000003f800bcd83721
      be52cccc4b6457321f22a7dc769f54eb8025913be804d9eabbfa19b3d7c5d3ca658d768caf57067eec83c7e6e9f81d0586703ed9ddda
      dd20675d63424980b10eb364e81eb37db40ed100344c928886ff4ccc37203ee6106d5b59d1ac102e2cd2d7ac17f4d96c398e5fd993ec
      b4ffdf79b17547ff9fa2aa8eefd6cbda124cbb17a0f8528146387135e226b005a474b9062ff264d2ff8efa36814aa2950065b1b04c0a
      1ae9b2f69d4a4aa979d6ce95fee9485ed0a03aee9bd953e81cfd1ef6e814dfd3c2ce37aefa38c1f9877371e91d6a5eb59fdedf75d3325fa3ca
    rt = IDT_NEO2.SharedController.emv_setCAPK(capk, null);
    capk = Common.getByteArray("
      a0000000039601017616e9ac8be014af88ca11a8fb17967b7394030e000000038000b74586d1
      9a207be6627c5b0aafbc44a2ecf5a2942d3a26ce19c4ffaeee920521868922e893e7838225a3947a2614796fb2c0628ce8c11e3825a5
      6d3b1bbaef783a5c6a81f36f8625395126fa983c5216d3166d48acde8a431212ff763a7f79d9edb7fed76b485de45beb829a3d4730848a366c
    rt = IDT_NEO2.SharedController.emv_setCAPK(capk, null);
    capk = Common.getByteArray("
      a0000000035701012 51a5f5de61cf28b5c6e2b5807c0644a01d46ff5000100016000942b7f2b
      a5ea307312b63df77c5243618acc2002bd7ecb74d821fe7bdc78bf28f49f74190ad9b23b9713b140ffec1fb429d93f56bdc7ade4ac075d7553
    rt = IDT_NEO2.SharedController.emv_setCAPK(capk, null);
    capk = Common.getByteArray("
      a0000000035801017 53ed0aa23e4cd5abd69eae7904b684a34a57c2200010001c80099552c4a
      1ecd68a0260157fc4151b5992837445d3fc57365ca5692c87be358cdcdf2c92fb6837522842a48eb11cdffe2fd91770c7221e4af6207
      c2de4004c7dee1b6276dc62d52a87d2cd01fbf2dc4065db52824d2a2167a06d19e6a0f781071cdb2dd314cb94441d8dc0e936317b77b
      f06f5177f6c5aba3a3bc6aa30209c97260b7a1ad3a192c9b8cd1d153570afcc87c3cd681d13e997fe33b3963a0a1c79772acf991033e1b8397
    rt = IDT_NEO2.SharedController.emv_setCAPK(capk, null);
    capk = Common.getByteArray("
      a0000000035401010696061879 1a86d387301edd4a3baf2d34fef1b400010001f800c6ddc0b7
      645f7f16286ab7e4116655f56dd0c944766040dc68664dd973bd3bfd4c525bcbb95272b6b3ad9ba8860303ad08d9e8cc344a4070f4cf
      b9eeaf29c8a3460850c264cda39bbe3a7e7d08a69c31b5c8dd9f94ddbc9265758c0e7399adcf4362caee458d414c52b498274881b196
      dacca7273f687f2a65faeb809d4b2ac1d3d1efb4f6490322318bd296d153b307a3283ab4e5be6ebd910359a8565eb9c4360d24baaca3
      dbfe393f3d6c830d603c6fc1e83409dfcd80d3a33ba243813bbb4ceaf9cbab6b74b00116f72ab278a88a011d70071e06cab140646438d986d4
    rt = IDT_NEO2.SharedController.emv_setCAPK(capk, null);
    if (rt == RETURN_CODE.RETURN_CODE_DO_SUCCESS)
    {
        tbOutput.AppendText("Load Visa CAPK Successful:" + "  \r\n");

    }
    else
```

```
            {
                tbOutput.AppendText("Load Visa CAPK failed Error Code: " + "0x" + String.Format("{0:X}", (ushort)rt
              ) + ": " + IDTechSDK.errorCode.getErrorString(rt) + "\r\n");
            }
        }

        private void btnListCAPKs_Click(object sender, EventArgs e)
        {
            byte[] capk = null;
            RETURN_CODE rt = IDT_NEO2.SharedController.emv_retrieveCAPKList(ref capk);
            if (rt == RETURN_CODE.RETURN_CODE_DO_SUCCESS)
            {

                tbOutput.AppendText("List CAPK Successful  \r\n");
                if (capk.Length > 0)
                {
                    for (int x = 0; x < capk.Length; x = x + 6)
                    {
                        byte[] thecapk = new byte[] { capk[x], capk[x + 1], capk[x + 2], capk[x + 3], capk[x + 4],
          capk[x + 5] };
                        tbOutput.AppendText(Common.getHexStringFromBytes(thecapk) + " \r\n");
                    }

                }
            }
            else
            {
                tbOutput.AppendText("List CAPK failed Error Code: " + "0x" + String.Format("{0:X}", (ushort)rt) + "
          : " + IDTechSDK.errorCode.getErrorString(rt) + "\r\n");
            }
        }

            private void btnStartSwipe_Click(object sender, EventArgs e)
        {
            RETURN_CODE rt = IDT_NEO2.SharedController.msr_startMSRSwipe(60);
            if (rt == RETURN_CODE.RETURN_CODE_DO_SUCCESS)
            {
                tbOutput.AppendText("MSR Turned On successfully; Ready to swipe\r\n");
                tbOutput.ReadOnly = false;
                tbOutput.Focus();
            }
            else
            {
                tbOutput.AppendText("MSR Turned On failed Error Code: " + "0x" + String.Format("{0:X}", (ushort)rt)
               + ": " + IDTechSDK.errorCode.getErrorString(rt) + "\r\n");
            }
        }

        private void btnCancelSwipe_Click(object sender, EventArgs e)
        {
            RETURN_CODE rt = IDT_NEO2.SharedController.msr_cancelMSRSwipe();
            if (rt == RETURN_CODE.RETURN_CODE_DO_SUCCESS)
            {
                tbOutput.AppendText("MSR Turned Off successfully\r\n");
                System.Diagnostics.Debug.WriteLine("MSR Turned Off successfully");
            }
            else
            {
                tbOutput.AppendText("MSR Turned Off failed Error Code: " + "0x" + String.Format("{0:X}", (ushort)rt
              ) + ": " + IDTechSDK.errorCode.getErrorString(rt) + "\r\n");
                System.Diagnostics.Debug.WriteLine("MSR Turned Off 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");
            if (cardData.emv_resultCode == EMV_RESULT_CODE.EMV_RESULT_CODE_GO_ONLINE)
            {
                //auto complete.  Normally, a host response is required here
                SetOutputText("Online request. Auto Complete EMV Transaction.\n");
                byte[] responseCode = new byte[] { 0x30, 0x30 };
                byte[] iad = new byte[] { 0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77, 0x88, 0x30, 0x30 };
                RETURN_CODE rt = IDT_NEO2.SharedController.emv_completeTransaction(false, responseCode, iad
, null, null);
                return;
            }
            displayCardData(cardData);
            break;
        case DeviceState.DataReceived:
            //Low-level data received  for IDT_DEVICE_TYPES type
            SetOutputTextLog(GetTimestamp() + " IN: " + Common.getHexStringFromBytes(data));
            break;
        case DeviceState.DataSent:
            //Low-level data sent  for IDT_DEVICE_TYPES type
            SetOutputTextLog(GetTimestamp() + " 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.DeviceTimeout:
            //Waiting for device timeout
            SetOutputText("Callback:DeviceTimeout\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");
            if (emvCallback.callbackType == EMV_CALLBACK_TYPE.EMV_CALLBACK_TYPE_LCD) //LCD
 Callback Type
            {
                if (emvCallback.lcd_displayMode == EMV_LCD_DISPLAY_MODE.
EMV_LCD_DISPLAY_MODE_CLEAR_SCREEN)
                {
                    SetOutputText("LCD Callback:Clear LCD Screen\n");

                    return;
                }
                else if (emvCallback.lcd_displayMode == EMV_LCD_DISPLAY_MODE.
EMV_LCD_DISPLAY_MODE_MESSAGE)
                {
                    // A message is requested to be displayed. See other test app included with SDK for a
 complete example.
                    SetOutputText("LCD Callback:Display Message\n");
                }
                else
                {
                    //Display message with menu/language or prompt, and return result to
```

```
        emv_callbackResponseLCD
                    //Kernel will not proceed until this step is complete
                    //seeing to default value of 1.  See other test app included with SDK for a complete
        example
                    SetOutputText("LCD Callback:Menu Display Request.  Sending result 1\n");
                    IDT_NEO2.SharedController.emv_callbackResponseLCD(emvCallback.lcd_displayMode, 1);
                }

            }
            break;
         default:
            break;

    }


}



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

    if (cardData.Event == EVENT_TRANSACTION_DATA_Types.EVENT_TRANSACTION_PIN_DATA)
    {
        SetOutputText("PIN Data received:\r\nKSN: " + cardData.pin_KSN + "\r\nPINBLOCK: " + cardData.
      pin_pinblock + "\r\n");
        return;

    }

    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

L100 Pass-Through Mode

On some products, the L100 Pinpad interfaces directly with the unit. Normally, the L100 would be controlled by the NEOII device as needed, for example, collecting a PIN during an EMV transaction. But there may be instances where sending direct commands to the L100 is required by the integrator. For this, we have a L100 Pass-Through Mode.

Enabling L100 Pass-Through Mode

Pass a value to TRUE to IDTechSDK::IDT_NEO2::device_enablePassThrough()
You then send IDT_L100.SharedController commands according to the commands available in the IDT_L100 class. For example, to reboot device, you send:

```
IDT_L100.SharedController.rebootDevice();
```

Disabling L100 Pass-Through Mode

Pass a value to FALSE to IDTechSDK::IDT_NEO2::device_enablePassThrough()
You now resume sending all commands to the IDT_NEO2.SharedController

Chapter 8

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 9

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 10

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 11

EMV Callback

During an EMV transaction, without a built-in LCD display on the NEO2, LCD Display messages will be returned as an EMV Callback.

In the MessageCallback for the SDK, there is a DeviceState EMVCallback. When this DeviceState is received, a IDTechSDK::EMV_Callback class object will be returned. IDTechSDK::EMV_Callback::callbackType will specify the type of callback: EMV_CALLBACK_TYPE_LCD, EMV_CALLBACK_TYPE_PINPAD, EMV_CALLBACK_MSR. The NEO2 will not utilize the EMV_CALLBACK_TYPE_PINPAD.

If NEO2 terminal settings specify MSR is part of the configuration, for cases of fallback, the type of callback will be EMV_CALLBACK_MSR. When this is received, MSR data must be collected and returned to IDTechSDK::IDT_NEO2::emv_callbackResponseMSR() to complete the transaction.

For LCD display messages, the callback type will be EMV_CALLBACK_TYPE_LCD. To evaluate what kind of LCD message, you get EMV_LCD_DISPLAY_MODE from IDTechSDK::EMV_Callback::lcd_displayMode: 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_LANGUAGE_SELECT: Language selection, response required with selected language index # 16 - LCD_DISPLAY_MODE_CLEAR_SCREEN: Request to clear LCD screen of information

If the mode is LCD_DISPLAY_MODE_MESSAGE or LCD_DISPLAY_MODE_CLEAR_SCREEN, these do not pause the EMV transaction. These two modes are for displaying a message (no response required), or for clearing the screen.

If the mode is LCD_DISPLAY_MODE_MENU, LCD_DISPLAY_MODE_PROMPT, or LCD_DISPLAY_MODE_LANGUAGE_SELECT, the provided message must be displayed, and then the EMV transaction pauses until a response is sent to IDTechSDK::IDT_NEO2::emv_callbackResponseLCD().

The message to display is byte[] lcd_messages. This contains either Message String, or a Message ID according to LCD Foreign Language Mapping Table (Foreign Language Mapping Table).

Chapter 12

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
5F28	Issuer Country Code
5F2A	Transaction Currency Code (Default: 08 40)
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 Selection
6F	File Control Information (FCI) Template
61	Application Template
62	File Control Parameters (FCP) 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)
82	Application Interchange Profile (AIP)

Tag	Description
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	Authorization Response 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
99	Transaction Personal Identification Number (PIN) Data
98	Transaction Certificate (TC) Hash Value
9A	Transaction Date (YYMMDD)
9A	Transaction Date
9B	Transaction Status Information
9B	Transaction Status Information
9C	Transaction Type
9C	Transaction Type
9D	Directory Definition File (DDF) Name
9F01	Acquirer Identifier
9F02	Amount, Authorized (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 (Default: 00 02)
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

Tag	Description
9F16	Merchant Identifier
9F17	Personal Identification Number (PIN) Try Counter
9F18	Issuer Script Identifier
9F19	Deleted (see 9F49)
9F1A	Terminal Country Code
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 (HHMMSS)
9F22	Certification Authority Public Key Index
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 (see below)
9F34	Cardholder Verification Method (CVM) Results
9F35	Terminal Type (see below)
9F36	Application Transaction Counter (ATC)
9F37	Unpredictable Number
9F38	Processing Options Data Object List (PDOL)
9F39	POS Entry Mode (Default: 07)
9F3A	Amount, Reference Currency
9F3B	Application Reference Currency
9F3C	Transaction Reference Currency Code
9F3D	Transaction Reference Currency Exponent
9F40	Additional Terminal Capabilities (see below)
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
9F47	Integrated Circuit Card (ICC) Public Key Exponent
9F48	Integrated Circuit Card (ICC) 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
9F4E	Merchant Name and Location

Tag	Description
9F4F	Log Format
9F50	Offline Accumulator Balance
9F51	Application Currency Code
9F52	Application Default Action (ADA)
9F53	Transaction Category Code
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
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
9F60	CVC3 (Track1)
9F61	CVC3 (Track2)
9F62	PCVC3 (Track1)
9F64	NATC (Track1)
9F65	PCVC3 (Track2)
9F66	PUNATC (Track2)
9F67	NATC (Track2)
9F68	Card Additional Processes
9F69	UDOL
9F6A	Unpredictable Number (Numeric)
9F6B	Track 2 Data
9F6C	Card Transaction Qualifiers (CTQ)
9F6D	Mag-stripe Application Version Number (Reader)
9F6E	Third Party Data
9F6E	Terminal Transaction Capabilities
9F6F	DS Slot Management Control
9F70	Protected Data Envelope 1
9F71	Protected Data Envelope 2
9F72	Protected Data Envelope 3
9F73	Protected Data Envelope 4
9F74	Protected Data Envelope 5
9F75	Unprotected Data Envelope 1
9F76	Unprotected Data Envelope 2
9F77	Unprotected Data Envelope 3
9F78	Unprotected Data Envelope 4
9F79	Unprotected Data Envelope 5
9F7A	VLP Terminal Support Indicator
9F7B	VLP Terminal Transaction Limit
9F7C	Customer Exclusive Data (CED)
9F7D	DS Summary 1

Tag	Description
9F7F	DS Unpredictable Number
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
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
DF01	ApplePay VAS Protocol
DF02	ApplePay VAS Failure Report
DF10	Terminal Languages Supported
DF10	Multi Language (Default: “enfr”)
DF11	Enable Transaction Logging
DF13	Terminal Action Code - Default
DF14	Terminal Action Code - Denial
DF15	Terminal Action Code - Online
DF17	Threshold Value for Biased Random Selection
DF18	Target Percentage to be Used for Random Selection
DF19	Maximum Target Percentage to be used for Biased Random Selection
DF1F	Last 4 digits of Primary Account Number (PAN)
DF21	Issuer Script Results
DF22	Force Online (1-Enable, 0-Disable)
DF25	Default DDOL (1-Enable, 0-Disable)
DF26	Revocation List Support (Default: Enable - 1)
DF27	Exception File Support (Default: Disable - 0)
DF28	Default TDOL
DF29	Terminal Capabilities - CVM Required
DF2A	Threshold Value for Biased Random Selection(Interac)

Tag	Description
DF2B	Maximum Target Percentage for Biased Random Selection (Interac)
DF2C	Target Percentage for Random Selection(Interac)
DF30	Track Data Source
DF31	DD Card Track 1
DF32	DD Card Track 2
DF33	Interac Receipt Required
DF34	TTK Customer - Firmware Version
DF40	Message to be displayed by EMV Kernel on “PIN Try Limit Exceeded” condition
DF41	Message to be displayed by EMV Kernel on “Last PIN Try” condition
DF42	Message to be displayed by EMV Kernel on “Please Try Again” condition
DF43	Message to be displayed by EMV Kernel on “Call Your Bank” condition
DF45	GMEDS Secret Keys
DF46	GMAD MIDs
DF47	ISIS Read Cmd Data
DF48	ISIS Write Data
DF49	ISIS Transaction Data
DF4A	TTK Customer - Current KSN of Data encryption Key
DF4B	TTK Customer - MSR all track data
DF4C	TTK Customer - Masked PAN
DF4D	TTK Customer - Additional POS Info
DF4E	Polling Options
DF4F	TTK Customer - Fallback Reason
DF50	Special Flow
DF51	Amex Terminal Capability
DF52	Transaction CVM
DF55	RID
DF56	Activate Trans for DESFireViVOComm Flows
DF57	Reader Primary Language
DF57	2nd usage: Remaining Candidates
DF58	Reader Secondary Language
DF5A	TLVExclusion List
DF5B	Terminal Entry Capability
DF5C	RF Deactivate Period
DF5D	D-PAS Issuer Script Response status
DF5E	Transaction Timing Information
DF5F	Encrypted PAN for remote PIN Pad
DF60	Product ID
DF61	Processor ID
DF61	CVMRequiredLimit_JCBScheme
DF62	Main Firmware Build ID
DF63	CB Enhanced DDA Indicator (same block as DF03)
DF64	CB Wave 2 CVM Requirements (same block as DF04)
DF65	Build ID Num (Cxx)
DF65	CB Display Offline Funds Indicator (same block as DF05)
DF65	Serial heartbeat Required
DF66	SVN Number
DF66	CB Terminal Type (same block as 9F35)
DF66	Display Unsupported Card
DF68	Enable/Disable STOP command processing
DF69	ConfigureProprietaryTags

Tag	Description
DF6A	Enable/Disable Comm Error Recovery
DF6C	Cubic FTP Phase 2 Mode Options
DF6D	Cubic Mode 3 Match AID
DF6E	Cubic Fixed Fare Amounts
DF6F	Cubic Timestamp Data
DF70	Loyalty Program ID
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
DF891B	Poll Mode
DF891C	Interac Retry Limit
DFDE04	MSR Encryption Option
DFEE0C	PPSE Terminate Flags
DFEE12	KID
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	Contact Terminal Configuration (see below)
DFEE1F	Issuer script device limit, Range: 0∼255 (Default: 128)
DFEE20	ICC Power on detect waiting time. (Unit: Sec) (Default: 60S)
DFEE21	ICC L1 waiting time. (Unit: Sec)(Default: 10 S)
DFEE22	Driver (Menu, Get PIN, Get MSR) Timeout. (Unit: Sec) (see below)
DFEE23	MSR Track Data
DFEE24	Force Acceptance (Default: 00)
DFEE25	ICC Response Code
DFEE26	Encryption StatusInformation
DFEE27	MSR Control
DFEF1A	TLV available
DFEF1A	Encrypted Sensitive Tags
DFEF1A	Auto Authenticate
DFEF20	MAC option in reponse data

Tag	Description
DFEF21	BIN
DFEF22	AID
DFEF23	HMAC
DFEF24	HMAC KSN
DFEF25	Output Data Format Select
DFEF26	MSR fallback
DFEF27	Online capability
DFEF28	Disable Encrypt ON
DFEF2C	Terminal AID List
DFEF2E	Terminal Transaction Log
DFEF2F	CUP configuration
DFEF30	White List
DFEF31	Black List
DFEF32	Auto-Switch
DFEF34	Antenna Detection Switch
DFEF35	Communications Watchdog Period
DFEF36	Media Control & Status Tracking
DFEF37	Interface Select
DFEF38	Timeout for Next Command
DFEF39	Network Indicate
DFEF3A	Reader Behavior Mode
DFEF3B	Autopoll Transaction Separation Interval
DFEF40	Ascii-code encryption Tag57 TLV
DFEF41	MAC Verification Data for SRED
DFEF42	MAC VerificationKSN for SRED
DFEF43	Local TZ/DST information.
DFEF44	CombinationOptions
DFEF45	Removal Timeout
DFEF46	ACT Pass Response DOL
DFEF47	CDA Hash Input
DFEF48	Indicate - retrieve transaction resultagain due to Output RAM isNotenough.
DFEF49	Outcome Parameter Set
DFE↩ F4A	User Interface Request Data
DFEF4B	MSR Equivalent Data Option
DFEF4C	MSR Equivalent Data Track Lengths
DFEF4D	MSR Equivalent Data
DFEF4E	ACT MSD Response DOL
DFEF4F	ACT Decline Response DOL
DFEF50	Terminal Interchange Profile (JCB)
DFEF51	Bypass EMV Completion Output
DFEF52	Re-FallBack times
DFEF53	Dynamic Reader Limits
DFEF54	SmartTap AID Index
DFEF55	Kernel Specific Features
DFEF56	Retry Limit
DFEF57	PPSE Terminate Flags
DFEF59	Terminal Data Setting - Default Amount
DFEF5A	Terminal Data Setting - Tags to Return
DFE↩ F5B	Mask for Tag5A

Tag	Description
DFEF5C	Mask for Tag56
DFEF5D	Mask for Tag57
DFEF5E	Mask for Tag9F6B
DFEF5F	Mask forTagFFEE13
DFEF60	Mask forTagFFEE14
DFEF61	Error Code
DFEF62	Allow MSR Swipe data from ICC Card
DFEF63	Tags To Read Yet
DFEF64	Referral Timeout
DFEF6E	USB-KB Output Data Postfix
DFEF6F	Inter-character Delay for USB-KB Interface
DFEF70	PISCES dual interface interference prevention mechanism fine-tune parameters.
DFEF71	Waiting ICC insert time
DFEF72	Pre-poll card mechanism control in ACT cmd & config setting
DFEF73	Transaction Message Type
DFEF74	Reference amplitude value
DFEF75	Reference delta value
DFEF76	Transaction Interface Type to activate
DFEF77	Timeout for waiting next command
DFEF78	EMV contact L2 display messages option
DFEF79	PIN block format (when TDES)
DFEF7A	Enable Apple Pay Check
DFEF7B	Apple Pay Status
DFEF7C	Track Bit Encoding
DFEF7D	Re-power on times
DFEF7E	Fallback response code list
FF69	ViVOpay Proprietary Tag List
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
FFE3	Selection Features - GR 1.2.10
FFE4	Group Number / Fallback Group
FFE5	Max AID Length
FFE6	AID Disabled

Tag	Description
FFE7	Interface Support
FFE8	Exclude from Processing
FFE9	Kernel ID Transaction Type Group List
FFEA	Default Kernel ID
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
FFEE05	M/Chip3 Intermediate Message Marker
FFEE06	ApplePay VAS Container
FFEE07	Encrypted Sensitive Tags
FFEE08	Masked Tags
FFEE0A	BIN Range
FFEE0B	AID Range
FFEE0C	White List
FFEE10	ViVOpay MChip Group Tag
FFEE11	ViVOpay Discover Group Tag
FFEE12	KID
FFEE12	Cash Reader Risk Record
FFEE13	Track 1 Data
FFEE13	Cashback Reader Risk Record
FFEE14	Track 2 Data
FFEE14	DRL Record 1
FFEE15	DRL Record 2
FFEE16	DRL Record 3
FFEE17	DRL Record 4
FFEE18	Tags To Write Yet Before GenAC
FFEE19	Tags To Write Yet After GenAC
FFEE1A	Terminal App DET Data
FFEE1C	Unpredictable Number Range
FFEE1D	Sensitive Data Mask
FFE↩ E1E	Group 0 Initialize Flag
FFE↩ E1F	Error Code Table
FFEE20	Restart Deactivation Time
FFF0	Specific Features Switch
FFF1	Terminal Contactless Transaction Limit
FFF2	Terminal IFD
FFF3	Application Capability
FFF4	Visa Reader Risk Flags
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

9F33 Terminal Capabilities

```
Byte 1
```

```
b8  b7  b6  b5  b4  b3  b2  b1  Meaning
1   x   x   x   x   x   x   x   Manual key entry
x   1   x   x   x   x   x   x   Magnetic stripe
x   x   1   x   x   x   x   x   IC with contacts
x   x   x   0   x   x   x   x   RFU
x   x   x   x   0   x   x   x   RFU
x   x   x   x   x   0   x   x   RFU
x   x   x   x   x   x   0   x   RFU
x   x   x   x   x   x   x   0   RFU

Byte 2
b8  b7  b6  b5  b4  b3  b2  b1  Meaning
1   x   x   x   x   x   x   x   Plaintext PIN for IC verification
x   1   x   x   x   x   X   x   Enciphered PIN for online verification
x   x   1   x   x   x   X   x   Signature(paper)
x   x   x   1   x   x   X   x   Enciphered PIN for offline verification
x   x   x   x   1   x   X   x   No CVM Required
x   x   x   x   x   0   x   x   RFU
x   x   x   x   x   x   0   x   RFU
x   x   x   x   x   x   X   0   RFU

Byte 3
b8  b7  b6  b5  b4  b3  b2  b1  Meaning
1   x   x   x   x   x   x   x   SDA
X   1   x   x   x   x   x   x   DDA
X   x   1   x   x   x   x   x   Card capture
x   x   x   0   x   x   x   x   RFU
x   x   x   x   1   x   x   X   CDA
x   x   x   x   x   0   x   x   RFU
x   x   x   x   x   x   0   x   RFU
x   x   x   x   X   X   x   0   RFU
```

9F40 Additional Terminal Capabilities

```
Byte 1
b1  b2  b3  b4  b5  b6  b7  b8  Meaning
1   x   x   x   x   x   x   x   Cash
x   1   x   x   x   x   x   x   Goods
x   x   1   x   x   x   x   x   Services
x   x   x   1   x   x   x   x   Cashback
x   x   x   x   1   x   x   x   Inquiry
x   x   x   x   x   1   x   x   Transfer
x   x   x   x   x   x   1   x   Payment
x   x   x   x   x   x   x   1   Administrative

Byte 2
b8  b7  b6  b5  b4  b3  b2  b1  Meaning
1   x   x   x   x   x   x   x   Cash Deposit
x   0   x   x   x   x   x   x   RFU
x   x   0   x   x   x   x   x   RFU
x   x   x   0   x   x   x   x   RFU
x   x   x   x   0   x   x   x   RFU
x   x   x   x   x   0   x   x   RFU
x   x   x   x   x   x   0   x   RFU
x   x   x   x   x   x   x   0   RFU

Byte 3
b8  b7  b6  b5  b4  b3  b2  b1  Meaning
1   x   x   x   x   x   x   x   Numeric keys
x   1   x   x   x   x   x   x   Alphabetic and special characters keys
x   x   1   x   x   x   x   x   Command keys
x   x   x   1   x   x   x   x   Function Keys
x   x   x   x   0   x   x   x   RFU
x   x   x   x   x   0   x   x   RFU
x   x   x   x   x   x   0   x   RFU
x   x   x   x   x   x   x   0   RFU

Byte 4
b8  b7  b6  b5  b4  b3  b2  b1  Meaning
1   x   x   x   x   x   x   x   Print, attendant
x   1   x   x   x   x   x   x   Print, cardholder
x   x   1   x   x   x   x   x   Display, attendant
x   x   x   1   x   x   x   x   Display, cardholder
x   x   x   x   0   x   x   x   RFU
x   x   x   x   x   0   x   x   RFU
x   x   x   x   x   x   1   x   Code table 10
x   x   x   x   x   x   x   1   Code table 9

Byte 5
b8  b7  b6  b5  b4  b3  b2  b1  Meaning
1   x   x   x   x   x   x   x   Code table 8
x   1   x   x   x   x   x   x   Code table 7
x   x   1   x   x   x   x   x   Code table 6
x   x   x   1   x   x   x   x   Code table 5
x   x   x   x   1   x   x   x   Code table 4
x   x   x   x   x   1   x   x   Code table 3
```

```
x   x   x   x   x   x   1   x   Code table 2
x   x   x   x   x   x   x   1   Code table 1
```

9F35 Terminal Type

Environment	Financial Institution	Merchant	Cardholder
Attended			
Online only	11	21	
Offline with online capability	12	22	
Offline only	13	23	
Unattended			
Online only	14	24	34
Offline with online capability	15	25	35
Offline only	16	26	36

DFEE1E Contact Terminal Configuration (Default: F0 DC 3C F0 C2 9E 94 00)

```
Byte 1
b8  b7  b6  b5  b4  b3  b2  b1  Meaning
1   x   x   x   x   x   x   x   Key Pad support
x   1   x   x   x   x   x   x   LCD support
x   x   1   x   x   x   x   x   PIN Pad support
x   x   x   1   x   x   x   x   Print Support
x   x   x   x   0   x   x   x   RFU
x   x   x   x   x   0   x   x   RFU
x   x   x   x   x   x   0   x   RFU
x   x   x   x   x   x   X   0   RFU

Byte 2
b8  b7  b6  b5  b4  b3  b2  b1  Meaning
1   x   x   x   x   x   x   x   PSE support
x   1   x   x   x   x   x   x   Cardholder confirmation
x   x   1   x   x   x   x   x   Preferred display order
x   x   x   1   x   x   x   x   Multi language
x   x   x   x   1   x   x   x   EMV language selection method
x   x   x   x   x   1   x   x   Default DDOL
x   x   x   x   x   x   0   x   RFU
x   x   x   x   x   x   x   0   RFU

Byte 3
b8  b7  b6  b5  b4  b3  b2  b1  Meaning
0   x   x   x   x   x   x   x   RFU
(Revocation of Issuer Public Key Certificate (DF26))
x   1   x   x   x   x   x   x   Manual action when CA PK loading fails
x   x   1   x   x   x   x   x   CA PK verified with check sum
x   x   x   1   x   x   x   x   Bypass PIN Entry
x   x   x   x   1   x   x   x   Subsequent bypass PIN Entry
x   x   x   x   x   1   x   x   Get data for pin try counter
x   x   x   x   x   x   0   x   RFU
x   x   x   x   x   x   x   0   RFU

Byte 4
b8  b7  b6  b5  b4  b3  b2  b1  Meaning
1   x   x   x   x   x   x   x   Amount before CVM processing
x   1   x   x   x   x   x   x   Floor limit checking
x   x   1   x   x   x   x   x   Random transaction selection
x   x   x   1   x   x   x   x   Velocity checking
x   x   x   x   0   x   x   x   RFU
(Transaction Log (DF11))
x   x   x   x   x   0   x   x   RFU
(Exception File (DF27))
x   x   x   x   x   x   0   x   RFU
x   x   x   x   x   x   x   0   RFU

Byte 5
b8  b7  b6  b5  b4  b3  b2  b1  Meaning
1   x   x   x   x   x   x   X   Terminal action code support
x   1   x   x   x   x   x   x   Terminal action code can be change
x   x   1   x   x   x   x   x   Terminal action code can be deleted or disable
x   x   x   1   x   x   x   x   Default Action code processing before 1st GAC
x   x   x   x   1   x   x   x   Default Action code processing after 1st GAC
x   x   x   x   x   1   x   x   TAC/IAC default process when unable to go online (Skipped)
x   x   x   x   x   x   1   x   TAC/IAC default process when unable to go online (Normal)
x   x   x   x   x   x   x   0   RFU
```

```
Byte 6
b8  b7  b6  b5  b4  b3  b2  b1  Meaning
1   x   x   x   x   x   x   x   Forced Online support
x   1   x   x   x   x   x   x   Forced acceptance support
x   x   1   x   x   x   x   x   Advices support
x   x   x   1   x   x   x   x   Issuer referrals support
X   x   x   x   1   x   x   x   Batch data capture
x   x   x   x   x   1   x   x   Online data capture
X   x   x   x   x   x   1   x   Default TDOL
X   x   x   x   x   x   x   0   RFU


Byte 7
b8  b7  b6  b5  b4  b3  b2  b1  Meaning
1   x   x   x   x   x   x   x   amount and pin entered on the same keypad
x   1   x   x   x   x   x   x   ICC/Magstripe reader combined
x   x   1   x   x   x   x   x   Magstripe read first
x   x   x   1   x   x   x   x   Support account type selection
x   x   x   x   1   x   x   x   On fly script processing
x   x   x   x   x   1   x   x   Internal date management
x   x   x   x   x   x   1   x   Reversal Mode
(1)Unable go online
(2) ARC Error
0: (3) Online Approved but reader not approved.
1: (3) Online Approved but card response AAC.
x   x   x   x   x   x   x   0   RFU

Byte 8
b8  b7  b6  b5  b4  b3  b2  b1  Meaning
x   x   x   x   x   x   x   x   RFU
```

DFEE22 Driver (Menu, Get PIN, Get MSR) Timeout. (Unit: Sec)

```
Byte1: Timeout for Menu. (Default: 30 S)
Byte2: Timeout for Get PIN. (Default: 60 S)
Byte3: Timeout for Get MSR. (Default: 60 S)
```

Chapter 13

Namespace Index

13.1 Packages

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

Chapter 14

Class Index

14.1 Class List

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

Chapter 15

Namespace Documentation

15.1 IDTechSDK Namespace Reference

Classes

- class EMV_Callback
- class IDT_L100
- class IDT_NEO2
- 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_DISP↩**
  LAY_MODE_PROMPT = 2, **EMV_LCD_DISPLAY_MODE_MESSAGE** = 3,
  EMV_LCD_DISPLAY_MODE_LANGUAGE_SELECT = 8, **EMV_LCD_DISPLAY_MODE_CLEAR_SCRE↩**
  EN = 16, **EMV_LCD_DISPLAY_MODE_AR_MESSAGE2** = 0Xf2, **EMV_LCD_DISPLAY_MODE_AR_ME↩**
  SSAGE3 = 0Xf3,
  EMV_LCD_DISPLAY_MODE_AR_MESSAGE4 = 0Xf4 }
- enum CTLS_APPLICATION {
  CTLS_APPLICATION_NONE = 0, **CTLS_APPLICATION_MASTERCARD** = 1, **CTLS_APPLICATION_VI↩**
  SA = 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_D↩**
  IAGNOSTIC = 10, **CTLS_APPLICATION_HID** = 11,
  CTLS_APPLICATION_MSR_SWIPE = 12, **CTLS_APPLICATION_RESERVED** = 13, **CTLS_APPLICATI↩**
  ON_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_M↩**
  KSK =2, **EMV_PIN_MODE_OFFLINE** =3,
  EMV_PIN_MODE_POG = 0x10, **EMV_PIN_MODE_MCPOG** = 0x11 }
- enum EMV_RESULT_CODE {
  EMV_RESULT_CODE_UNKNOWN = -1, **EMV_RESULT_CODE_APPROVED_OFFLINE** = 0, **EMV_RES↩**
  ULT_CODE_DECLINED_OFFLINE = 1, **EMV_RESULT_CODE_APPROVED** = 2,
  EMV_RESULT_CODE_DECLINED = 3, **EMV_RESULT_CODE_GO_ONLINE** = 4, **EMV_RESULT_COD↩**

E_CALL_YOUR_BANK = 5, **EMV_RESULT_CODE_NOT_ACCEPTED** = 6,
EMV_RESULT_CODE_FALLBACK_TO_MSR = 7, **EMV_RESULT_CODE_TIMEOUT** = 8, **EMV_RESUL↩T_CODE_GO_ONLINE_CTLS** = 9, **EMV_RESULT_CODE_FAILED** = 10,
EMV_RESULT_CODE_AUTHENTICATE_TRANSACTION = 0x0010, **EMV_RESULT_CODE_TRANSAC↩TION_CANCELED** = 0x0012, **EMV_RESULT_CODE_SWIPE_NON_ICC** = 0x11, **EMV_RESULT_CODE↩_CTLS_TWO_CARDS** = 0x7A,
EMV_RESULT_CODE_CTLS_TERMINATE = 0x7E, **EMV_RESULT_CODE_CTLS_TERMINATE_TRY_↩ANOTHER** = 0x7D, **EMV_RESULT_CODE_MSR_SWIPE_CAPTURED** = 0x80, **EMV_RESULT_CODE_↩REQUEST_ONLINE_PIN** = 0x81,
EMV_RESULT_CODE_REQUEST_SIGNATURE = 0x82, **EMV_RESULT_CODE_FALLBACK_TO_CON↩TACT** = 0x83, **EMV_RESULT_CODE_FALLBACK_TO_OTHER** = 0x84, **EMV_RESULT_CODE_REVER↩SAL_REQUIRED** = 0x85,
EMV_RESULT_CODE_ADVISE_REQUIRED = 0x86, **EMV_RESULT_CODE_ADVISE_REVERSAL_RE↩QUIRED** = 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_FAILED** = 0x1003, **EMV_RESULT_CODE_MEMOR↩Y_NOT_ENOUGH** = 0x2001,
EMV_RESULT_CODE_SMARTCARD_OK = 0x3001, **EMV_RESULT_CODE_SMARTCARD_FAIL** = 0x3002, **EMV_RESULT_CODE_SMARTCARD_INIT_FAILED** = 0x3003, **EMV_RESULT_CODE_FAL↩LBACK_SITUATION** = 0x3004,
EMV_RESULT_CODE_SMARTCARD_ABSENT = 0x3005, **EMV_RESULT_CODE_SMARTCARD_TIME↩OUT** = 0x3006, **EMV_RESULT_CODE_MSR_CARD_ERROR** = 0x3007, **EMV_RESULT_CODE_MSR_C↩ARD_READ_ERROR** = 0x3012,
EMV_RESULT_CODE_PARSING_TAGS_FAILED = 0x5001, **EMV_RESULT_CODE_CARD_DATA_↩ELEMENT_DUPLICATE** = 0x5002, **EMV_RESULT_CODE_DATA_FORMAT_INCORRECT** = 0x5003, **EMV_RESULT_CODE_APP_NO_TERM** = 0x5004,
EMV_RESULT_CODE_APP_NO_MATCHING = 0x5005, **EMV_RESULT_CODE_MANDATORY_OBJEC↩T_MISSING** = 0x5006, **EMV_RESULT_CODE_APP_SELECTION_RETRY** = 0x5007, **EMV_RESULT_C↩ODE_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_O↩UT_OF_RANGE** = 0x5012,
EMV_RESULT_CODE_SFI_OUT_OF_RANGE = 0x5013, **EMV_RESULT_CODE_AFL_INCORRECT** = 0x5014, **EMV_RESULT_CODE_EXP_DATE_INCORRECT** = 0x5015, **EMV_RESULT_CODE_EFF_DAT↩E_INCORRECT** = 0x5016,
EMV_RESULT_CODE_ISS_COD_TBL_OUT_OF_RANGE = 0x5017, **EMV_RESULT_CODE_CRYPT↩OGRAM_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_P↩RESENCED** = 0x5028,
EMV_RESULT_CODE_ARC_INVALID = 0x5029, **EMV_RESULT_CODE_COMM_NO_ONLINE** = 0x5030, **EMV_RESULT_CODE_TRAN_TYPE_INCORRECT** = 0x5031, **EMV_RESULT_CODE_APP_NO_SUPP↩ORT** = 0x5032,
EMV_RESULT_CODE_APP_NOT_SELECT = 0x5033, **EMV_RESULT_CODE_LANG_NOT_SELECT** = 0x5034, **EMV_RESULT_CODE_TERM_DATA_NOT_PRESENCED** = 0x5035, **EMV_RESULT_CODE_C↩VM_TYPE_UNKNOWN** = 0x6001,
EMV_RESULT_CODE_CVM_AIP_NOT_SUPPORTED = 0x6002, **EMV_RESULT_CODE_CVM_TAG_8E↩_MISSING** = 0x6003, **EMV_RESULT_CODE_CVM_TAG_8E_FORMAT_ERROR** = 0x6004, **EMV_RESU↩LT_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_RESU↩LT_CODE_UNKONWN** = 0xffff }

15.1.1 Enumeration Type Documentation

15.1.1.1 enum IDTechSDK.CTLS_APPLICATION [strong]

Define CTLS_APPLICATION.

15.1.1.2 enum IDTechSDK.EMV_CALLBACK_TYPE [strong]

Define EMV_CALLBACK_TYPES.

15.1.1.3 enum IDTechSDK.EMV_LCD_DISPLAY_MODE [strong]

Define EMV_LCD_DISPLAY_MODE.

15.1.1.4 enum IDTechSDK.EMV_PIN_MODE [strong]

Define EMV_PIN_MODE.

15.1.1.5 enum IDTechSDK.EMV_RESULT_CODE [strong]

Define EMV_PIN_MODE.

Chapter 16

Class Documentation

16.1 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

16.1.1 Detailed Description

Class for LCD Message

16.1.2 Member Data Documentation

16.1.2.1 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

16.1.2.2 byte [ ] IDTechSDK.EMV_Callback.language

Message Language

2 Bytes

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

16.1.2.3 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

16.1.2.4 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

16.1.2.5 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.

16.1.2.6 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.

16.1.2.7 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

16.1.2.8 bool IDTechSDK.EMV_Callback.maskEntry

Mask Entry

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

16.1.2.9 int IDTechSDK.EMV_Callback.msr_displayMessage

MSR Message

Message to display during swipe request

16.1.2.10 int IDTechSDK.EMV_Callback.msr_swipeTimeout

Swipe Timeout

Timeout value waiting for MSR Swipe

16.1.2.11 int IDTechSDK.EMV_Callback.pin_entryInterval

PIN Entry Interval Timeout value of interval between input each PIN

16.1.2.12 int IDTechSDK.EMV_Callback.pin_entryStartTimeout

PIN Entry Start Timeout

Timeout value waiting for PIN entry to start

16.1.2.13 byte [ ] IDTechSDK.EMV_Callback.pin_KSN

PIN KSN

Pairing DUKPT KSN

16.1.2.14 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

16.1.2.15 byte [ ] IDTechSDK.EMV_Callback.pin_truncatedPAN

Truncated PAN

Truncated PAN

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

- Source_Windows/IDT_Transactions.cs

16.2 IDTechSDK.IDT_L100 Class Reference

Public Member Functions

- RETURN_CODE device_getFirmwareVersion (ref string response)
- RETURN_CODE device_SymmetricRKI (int type)
- RETURN_CODE device_updateDeviceFirmware (byte[] firmwareData)
- RETURN_CODE device_sendPAE (string command, ref string response, int timeout, string ip="")

- 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 config_getModelNumber (ref string response)
- RETURN_CODE config_getSerialNumber (ref string response)
- bool config_setCmdTimeOutDuration (int newTimeOut)
- RETURN_CODE pin_getEncryptedPIN (int keyType, string PAN, string message, int timeout)
- RETURN_CODE pin_getManualPanEntry (bool csc, bool ADR, bool ZIP)
- RETURN_CODE pin_getFunctionKey ()
- RETURN_CODE pin_sendBeep (int frequency, int duration)
- RETURN_CODE device_rebootDevice ()
- RETURN_CODE pin_setKeypressCapture (bool showKeyValue)
- RETURN_CODE pin_cancelPINEntry ()
- RETURN_CODE device_getKeyStatus (ref byte[] status)
- RETURN_CODE pin_promptForKeyInput (int messageID, int languageID, bool maskInput, int minLen, int maxLen)
- RETURN_CODE pin_promptForKeyInputEnc (byte[] message, bool maskInput, int minLen, int maxLen)
- RETURN_CODE pin_promptForAmountInputEnc (byte[] message, int minLen, int maxLen)
- RETURN_CODE pin_promptForAmountInput (int messageID, int languageID, int minLen, int maxLen)
- RETURN_CODE device_setSleepModeTime (int time)
- RETURN_CODE device_startRKI ()
- RETURN_CODE device_enterStopMode ()
- RETURN_CODE device_setDateTime ()
- RETURN_CODE device_getDateTime (ref byte[] dateTime)
- RETURN_CODE lcd_clearDisplay (int lineNumber)
- RETURN_CODE lcd_clearAllLines ()
- RETURN_CODE lcd_savePrompt (int promptNumber, string prompt)
- RETURN_CODE lcd_displayPrompt (int promptNumber, int lineNumber)
- RETURN_CODE lcd_displayMessage (int lineNumber, string message)
- RETURN_CODE config_setEthernetMACAddress (byte[] address)
- RETURN_CODE config_getEthernetMACAddress (ref byte[] address)
- RETURN_CODE config_getNetworkConfiguration (ref bool isStatic, ref string address, ref string subnet, ref string gateway, ref string dns)
- RETURN_CODE config_setNetworkConfiguration (bool isStatic, string address, string subnet, string gateway, string dns)
- RETURN_CODE lcd_enableBacklight (bool enable)
- RETURN_CODE lcd_getBacklightStatus (ref bool enabled)
- RETURN_CODE config_setBaudRate (int baud)
- RETURN_CODE config_getBaudRate (ref int baud)

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 bool closeSerialPort ()
- static bool closeUSB ()
- 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_L100 SharedController [get]

16.2.1 Member Function Documentation

16.2.1.1 static bool IDTechSDK.IDT_L100.closeSerialPort ( ) [static]

Close Serial Port Interface

Instructs SDK to close the Serial Port if connected to L100

Returns

bool TRUE=successful, FALSE=failure

16.2.1.2 static bool IDTechSDK.IDT_L100.closeUSB ( ) [static]

Close USB

Instructs SDK to close the USB if connected to L100

Returns

bool TRUE=successful, FALSE=failure

16.2.1.3 RETURN_CODE IDTechSDK.IDT_L100.config_getBaudRate (ref int *baud*)

Get Baud Rate

Gets the buad rate for RS-232 communication.

Parameters

baud	• 2 = 2400 • 3 = 4800 • 4 = 9600 • 6 = 19200 • 7 = 38400 • 9 = 115200

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

16.2.1.4 RETURN_CODE IDTechSDK.IDT_L100.config_getEthernetMACAddress (ref byte[] *address*)

Get Device Ethernet MAC Address

Parameters

address	6-byte MAC Address

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

16.2.1.5 RETURN_CODE IDTechSDK.IDT_L100.config_getModelNumber (ref string *response*)

Polls device for Model Number

Parameters

response	Returns Model Number

Returns

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

16.2.1.6 RETURN_CODE IDTechSDK.IDT_L100.config_getNetworkConfiguration (ref bool *isStatic,* ref string *address,* ref string *subnet,* ref string *gateway,* ref string *dns*)

Get Device Network Configuration

Parameters

isStatic	TRUE = Static IP, FALSE = DHCP
address	Device IP Address as string. Example "192.168.1.15"
subnet	Device Subnet as string. Example "255.255.255.0"
gateway	Device Gateway address as a string. Example "8.8.8.8"
dns	Device DNS address as string. Example "192.168.1.22"

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

16.2.1.7 RETURN_CODE IDTechSDK.IDT_L100.config_getSerialNumber (ref string *response*)

Polls device for Serial Number

Parameters

response	Returns Serial Number

Returns

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

16.2.1.8 RETURN_CODE IDTechSDK.IDT_L100.config_setBaudRate (int *baud*)

Set Baud Rate

Sets the buad rate for RS-232 communication.

Parameters

baud	• 2 = 2400 • 3 = 4800 • 4 = 9600 • 6 = 19200 • 7 = 38400 • 9 = 115200

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

16.2.1.9 bool IDTechSDK.IDT_L100.config_setCmdTimeOutDuration (int *newTimeOut*)

Command Acknowledgement Timout

Sets the amount of seconds to wait for an {ACK} to a command before a timeout. Responses should normally be received under one second. Default is 3 seconds

Parameters

newTimeOut	Timout value. Valid range 1 - 60 seconds

Returns

Success flag. Determines if value was set and in range.

16.2.1.10 RETURN_CODE IDTechSDK.IDT_L100.config_setEthernetMACAddress (byte[] *address*)

Set Device Ethernet MAC Address

Parameters

address	6-byte MAC Address

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

16.2.1.11 RETURN_CODE IDTechSDK.IDT_L100.config_setNetworkConfiguration (bool *isStatic,* string *address,* string *subnet,* string *gateway,* string *dns*)

Set Device Network Configuration

Parameters

isStatic	TRUE = Static IP, FALSE = DHCP
address	Device IP Address as string. Example "192.168.1.15"
subnet	Device Subnet as string. Example "255.255.255.0"
gateway	Device Gateway address as a string. Example "8.8.8.8"
dns	Device DNS address as string. Example "192.168.1.22"

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

16.2.1.12 RETURN_CODE IDTechSDK.IDT_L100.device_enterStopMode ( )

Enter Stop Mode

Set device enter to stio mode. In stop mode, LCD display and backlight is off. Stop mode reduces power consumption to the lowest possible level. A unit in stop mode can only be woken up by a physical key press.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

16.2.1.13 RETURN_CODE IDTechSDK.IDT_L100.device_getDateTime (ref byte[] *dateTime*)

Get Date Time

Gets current system date and time of the device.

Parameters

dateTime	The date time returned as follows: • byte 0: Year 00-99 • byte 1: Month 01-12 • byte 2: Date 01-31 • byte 3: Hour 00-23 • byte 4: Minute 00-59 • byte 5: Second 00-59

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

16.2.1.14 RETURN_CODE IDTechSDK.IDT_L100.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()

16.2.1.15 RETURN_CODE IDTechSDK.IDT_L100.device_getKeyStatus (ref byte[] *status*)

Get Key Status

Gets the status of loaded keys

Parameters

status	byte 0: PIN DUKPT Key, 1 Exists, 0 None, 0xFF STOP byte 1: PIN Master Key, 1 Exists, 0 None byte 2: PIN Session Key, 1 Exists, 0 None byte 3: Account/MSR DUKPT Key, Does not support, always 0 byte 4: Account/ICC DUKPT Key, Does not support, always 0 byte 5: Admin DUKPT Key, 1 Exists, 0 None, 0xFF STOP

Returns

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

16.2.1.16 RETURN_CODE IDTechSDK.IDT_L100.device_rebootDevice ( )

Reboot Device

Executes a command to restart the device.

- Card data is cleared, resetting card status bits.
- Response data of the previous command is cleared.
- Resetting firmware.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

16.2.1.17 RETURN_CODE IDTechSDK.IDT_L100.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

16.2.1.18 RETURN_CODE IDTechSDK.IDT_L100.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

16.2.1.19 RETURN_CODE IDTechSDK.IDT_L100.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()

16.2.1.20 RETURN_CODE IDTechSDK.IDT_L100.device_setDateTime ( )

Set Date Time

Set current system date and time to the device.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

16.2.1.21 RETURN_CODE IDTechSDK.IDT_L100.device_setSleepModeTime (int *time*)

Set Sleep Mode Timer

Set device enter to sleep mode after the given time. In sleep mode, LCD display and backlight is off. Sleep mode reduces power consumption to the lowest possible level. A unit in Sleep mode can only be woken up by a physical key press.

Parameters

time	Enter sleep time value, in second.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

16.2.1.22 RETURN_CODE IDTechSDK.IDT_L100.device_startRKI ( )

Start Remote Key Injection

Starts a remote key injection request with IDTech RKI servers.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

16.2.1.23 RETURN_CODE IDTechSDK.IDT_L100.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

16.2.1.24 RETURN_CODE IDTechSDK.IDT_L100.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;
    }
}
```

16.2.1.25 static int IDTechSDK.IDT_L100.getCommandTimeout ( ) [static]

Get Command Timeout

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

Return values

time	Time

16.2.1.26 RETURN_CODE IDTechSDK.IDT_L100.lcd_clearAllLines ( )

Clear LCD Display

Clears all lines of the LCD Display.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

16.2.1.27 RETURN_CODE IDTechSDK.IDT_L100.lcd_clearDisplay (int *lineNumber*)

Clear LCD Display Line

Clears the line number of the LCD Display.

Parameters

lineNumber	Line number to clear (1-4)

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

16.2.1.28 RETURN_CODE IDTechSDK.IDT_L100.lcd_displayMessage (int *lineNumber,* string *message*)

Display Message on Line

Displays a message on a display line.

Parameters

lineNumber	Line number to display message on (1-4)
message	Message to display

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

16.2.1.29 RETURN_CODE IDTechSDK.IDT_L100.lcd_displayPrompt (int *promptNumber,* int *lineNumber*)

Display Prompt on Line

Displays a message prompt from L100 memory.

Parameters

promptNumber	Prompt number (0-9)
lineNumber	Line number to display message prompt (1-4)

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

16.2.1.30 RETURN_CODE IDTechSDK.IDT_L100.lcd_enableBacklight (bool *enable*)

Enable/Disable LCD Backlight

Turns on/off the LCD back lighting.

Parameters

enable	TRUE = turn ON backlight, FALSE = turn OFF backlight

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

16.2.1.31 RETURN_CODE IDTechSDK.IDT_L100.lcd_getBacklightStatus (ref bool *enabled*)

Get Backlight Status

Returns the status of the LCD back lighting.

Parameters

enabled	TRUE = Backlight is ON, FALSE = Backlight is OFF

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

16.2.1.32 static void IDTechSDK.IDT_L100.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

16.2.1.33 RETURN_CODE IDTechSDK.IDT_L100.lcd_savePrompt (int *promptNumber,* string *prompt*)

Save Prompt

Saves a message prompt to L100 memory.

Parameters

promptNumber	Prompt number (0-9)
prompt	Prompt string (up to 20 characters)

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

16.2.1.34 RETURN_CODE IDTechSDK.IDT_L100.pin_cancelPINEntry ( )

Cancel PIN Entry

Cancel “Get Function Key” & “Get Encrypted PIN” & “Get Numeric” & “Get Amount”

Returns

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

16.2.1.35 RETURN_CODE IDTechSDK.IDT_L100.pin_getEncryptedPIN (int *keyType,* string *PAN,* string *message,* int *timeout*)

Get Encrypted PIN

Requests PIN Entry

Parameters

keyType	• 0x00- MKSK-TDES: External Plaintext PAN • 0x01- DUKPT-TDES: External Plaintext PAN • 0x10 MKSK-TDES: External Ciphertext PAN • 0x11 DUKPT-TDES: External Ciphertext PAN
PAN	Account Number
message	Message to display = timeout

Returns

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

16.2.1.36 RETURN_CODE IDTechSDK.IDT_L100.pin_getFunctionKey ( )

Get Function Key

Captures a function key

- Backspace = B
- Cancel = C
- Enter = E
- * = *
- # = #
- Help = ?
- Function Key 1 = F1
- Function Key 2 = F2
- Function Key 3 = F3

Timeout hard wired 3 minutes

Returns

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

16.2.1.37 RETURN_CODE IDTechSDK.IDT_L100.pin_getManualPanEntry (bool *csc,* bool *ADR,* bool *ZIP*)

Get Manual Pan Entry

prompt the user to manually enter a card PAN and Expiry Date (and optionally CSC) from the keypad and return it to the POS.

Parameters

csc	Request CSS
ADR	Request Address
ZIP	Request Zip

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

16.2.1.38 RETURN_CODE IDTechSDK.IDT_L100.pin_promptForAmountInput (int *messageID,* int *languageID,* int *minLen,* int *maxLen*)

Prompt for Amount Input

Prompts for amount input using the secure message according to the following table

Msg Id	English Prompt	Portuguese Prompt	Spanish Prompt	French Prompt
1	ENTER	ENTER	INGRESE	ENTREZ
2	REENTER	RE-INTRODUZIR	REINGRESE	RE-ENTREZ
3	ENTER YOUR	INTRODUZIR O SEU	INGRESE SU	ENTREZ VOTRE

Msg Id	English Prompt	Portuguese Prompt	Spanish Prompt	French Prompt
4	REENTER YOUR	RE-INTRODUZIR O SEU	REINGRESE SU	RE-ENTREZ VOTRE
5	PLEASE ENTER	POR FAVOR DIGITE	POR FAVOR INGRESE	SVP ENTREZ
6	PLEASE REENTER	POR FAVO REENTRAR	POR FAVO REINGRESE	SVP RE-ENTREZ
7	PO NUMBER	NÚMERO PO	NUMERO PO	No COMMANDE
8	DRIVER ID	LICENÇA	LICENCIA	ID CONDUCTEUR
9	ODOMETER	ODOMETER	ODOMETRO	ODOMETRE
10	ID NUMBER	NÚMERO ID	NUMERO ID	No IDENT
11	EQUIP CODE	EQUIP CODE	CODIGO EQUIP	CODE EQUIPEMENT
12	DRIVERS ID	DRIVER ID	ID CONDUCTOR	ID CONDUCTEUR
13	JOB NUMBER	EMP NÚMERO	NUMERO EMP	No TRAVAIL
14	WORK ORDER	TRABALHO ORDEM	ORDEN TRABAJO	FICHE TRAVAIL
15	VEHICLE ID	ID VEÍCULO	ID VEHICULO	ID VEHICULE
16	ENTER DRIVER	ENTER DRIVER	INGRESE CONDUCTOR	ENTR CONDUCTEUR
17	ENTER DEPT	ENTER DEPT	INGRESE DEPT	ENTR DEPARTEMNT
18	ENTER PHONE	ADICIONAR PHONE	INGRESE TELEFONO	ENTR No TELEPH
19	ENTER ROUTE	ROUTE ADD	INGRESE RUTA	ENTREZ ROUTE
20	ENTER FLEET	ENTER FROTA	INGRESE FLOTA	ENTREZ PARC AUTO
21	ENTER JOB ID	ENTER JOB ID	INGRESE ID TRABAJO	ENTR ID TRAVAIL
22	ROUTE NUMBER	NÚMERO PATH	RUTA NUMERO	No ROUTE
23	ENTER USER ID	ENTER USER ID	INGRESE ID USUARIO	ID UTILISATEUR
24	FLEET NUMBER	NÚMERO DE FROTA	FLOTA NUMERO	No PARC AUTO
25	ENTER PRODUCT	ADICIONAR PRODUTO	INGRESE PRODUCTO	ENTREZ PRODUIT
26	DRIVER NUMBER	NÚMERO DRIVER	CONDUCTOR NUMERO	No CONDUCTEUR
27	ENTER LICENSE	ENTER LICENÇA	INGRESE LICENCIA	ENTREZ PERMIS
28	ENTER FLEET NO	ENTER NRO FROTA	INGRESE NRO FLOTA	ENT No PARC AUTO
29	ENTER CAR WASH	WASH ENTER	INGRESE LAVADO	ENTREZ LAVE-AUTO
30	ENTER VEHICLE	ENTER VEÍCULO	INGRESE VEHICULO	ENTREZ VEHICULE
31	ENTER TRAILER	TRAILER ENTER	INGRESE TRAILER	ENTREZ REMORQUE
32	ENTER ODOMETER	ENTER ODOMETER	INGRESE ODOMETRO	ENTREZ ODOMETRE
33	DRIVER LICENSE	CARTEIRA DE MOTORISTA	LICENCIA CONDUCTOR	PERMIS CONDUIRE
34	ENTER CUSTOMER	ENTER CLIENTE	INGRESE CLIENTE	ENTREZ CLIENT
35	VEHICLE NUMBER	NÚMERO DO VEÍCULO	VEHICULO NUMERO	No VEHICULE
36	ENTER CUST DATA	ENTER CLIENTE INFO	INGRESE INFO CLIENTE	INFO CLIENT
37	REENTER DRIVID	REENTRAR DRIVER ID	REINGRESE ID CHOFER	RE-ENTR ID COND
38	ENTER USER DATA	ENTER INFO USUÁRIO	INGRESE INFO USUARIO	INFO UTILISATEUR
39	ENTER CUST CODE	ENTER CODE. CLIENTE	INGRESE COD. CLIENTE	ENTR CODE CLIENT

Msg Id	English Prompt	Portuguese Prompt	Spanish Prompt	French Prompt
40	ENTER EMPLOYEE	ENTER FUNCIONÁRIO	INGRESE EMPLEADO	ENTREZ EMPLOYE
41	ENTER ID NUMBER	ENTER NÚMERO ID	INGRESE NUMERO ID	ENTREZ No ID
42	ENTER DRIVER ID	ENTER ID DRIVER	INGRESE ID CONDUCTOR	No CONDUCTEUR
43	ENTER FLEET PIN	ENTER PIN FROTA	INGRESE PIN DE FLOTA	NIP PARC AUTO
44	ODOMETER NUMBER	NÚMERO ODOMETER	ODOMETRO NUMERO	No ODOMETRE
45	ENTER DRIVER LIC	ENTER DRIVER LIC	INGRESE LIC CONDUCTOR	PERMIS CONDUIRE
46	ENTER TRAILER NO	NRO TRAILER ENTER	INGRESE NRO TRAILER	ENT No REMORQUE
47	REENTER VEHICLE	REENTRAR VEÍCULO	REINGRESE VEHICULO	RE-ENTR VEHICULE
48	ENTER VEHICLE ID	ENTER VEÍCULO ID	INGRESE ID VEHICULO	ENTR ID VEHICULE
49	ENTER BIRTH DATE	INSERIR DATA NAC	INGRESE FECHA NAC	ENT DT NAISSANCE
50	ENTER DOB MMDDYY	ENTER FDN MMDDYY	INGRESE FDN MMDDAA	NAISSANCE MMJJAA
51	ENTER FLEET DATA	ENTER FROTA INFO	INGRESE INFO DE FLOTA	INFO PARC AUTO
52	ENTER REFERENCE	ENTER REFERÊNCIA	INGRESE REFERENCIA	ENTREZ REFERENCE
53	ENTER AUTH NUMBR	ENTER NÚMERO AUT	INGRESE NUMERO AUT	No AUTORISATION
54	ENTER HUB NUMBER	ENTER HUB NRO	INGRESE NRO HUB	ENTREZ No NOYAU
55	ENTER HUBOMETER	MEDIDA PARA ENTRAR HUB	INGRESE MEDIDO DE HUB	COMPTEUR NOYAU
56	ENTER TRAILER ID	TRAILER ENTER ID	INGRESE ID TRAILER	ENT ID REMORQUE
57	ODOMETER READING	QUILOMETRAGEM	LECTURA ODOMETRO	LECTURE ODOMETRE
58	REENTER ODOMETER	REENTRAR ODOMETER	REINGRESE ODOMETRO	RE-ENT ODOMETRE
59	REENTER DRIV. ID	REENTRAR DRIVER ID	REINGRESE ID CHOFER	RE-ENT ID CONDUC
60	ENTER CUSTOMER ID	ENTER CLIENTE ID	INGRESE ID CLIENTE	ENTREZ ID CLIENT
61	ENTER CUST. ID	ENTER CLIENTE ID	INGRESE ID CLIENTE	ENTREZ ID CLIENT
62	ENTER ROUTE NUM	ENTER NUM ROUTE	INGRESE NUM RUTA	ENT No ROUTE
63	ENTER FLEET NUM	FROTA ENTER NUM	INGRESE NUM FLOTA	ENT No PARC AUTO
64	FLEET PIN	FROTA PIN	PIN DE FLOTA	NIP PARC AUTO
65	DRIVER #	DRIVER #	CONDUCTOR #	CONDUCTEUR
66	ENTER DRIVER #	ENTER DRIVER #	INGRESE CONDUCTOR #	ENT # CONDUCTEUR
67	VEHICLE #	VEÍCULO #	VEHICULO #	# VEHICULE
68	ENTER VEHICLE #	ENTER VEÍCULO #	INGRESE VEHICULO #	ENT # VEHICULE

Msg Id	English Prompt	Portuguese Prompt	Spanish Prompt	French Prompt
69	JOB #	TRABALHO #	TRABAJO #	# TRAVAIL
70	ENTER JOB #	ENTER JOB #	INGRESE TRABAJO #	ENTREZ # TRAVAIL
71	DEPT NUMBER	NÚMERO DEPT	NUMERO DEPTO	No DEPARTEMENT
72	DEPARTMENT #	DEPARTAMENTO #	DEPARTAMENTO #	DEPARTEMENT
73	ENTER DEPT #	ENTER DEPT #	INGRESE DEPTO #	ENT# DEPARTEME↩ NT
74	LICENSE NUMBER	NÚMERO DE LICE↩ NÇA	NUMERO LICENCIA	No PERMIS
75	LICENSE #	LICENÇA #	LICENCIA #	# PERMIS
76	ENTER LICENSE #	ENTER LICENÇA #	INGRESE LICENCIA #	ENTREZ # PERMIS
77	DATA	INFO	INFO	INFO
78	ENTER DATA	ENTER INFO	INGRESE INFO	ENTREZ INFO
79	CUSTOMER DATA	CLIENTE INFO	INFO CLIENTE	INFO CLIENT
80	ID #	ID #	ID #	# ID
81	ENTER ID #	ENTER ID #	INGRESE ID #	ENTREZ # ID
82	USER ID	USER ID	ID USUARIO	ID UTILISATEUR
83	ROUTE #	ROUTE #	RUTA #	# ROUTE
84	ENTER ROUTE #	ADD ROUTE #	INGRESE RUTA #	ENTREZ # ROUTE
85	ENTER CARD NUM	ENTER NÚMERO DE CARTÃO	INGRESE NUM TAR↩ JETA	ENTREZ NO CARTE
86	EXP DATE(YYMM)	VALIDADE VAL (AA↩ MM)	FECHA EXP (AAMM)	DATE EXPIR(AAMM)
87	PHONE NUMBER	TELEFONE	NUMERO TELEFONO	NO TEL
88	CVV START DATE	CVV DATA DE INÍCIO	CVV FECHA INICIO	CVV DATE DE DEB↩ UT
89	ISSUE NUMBER	NÚMERO DE EMIS↩ SÃO	NUMERO DE EMISI↩ ON	NO DEMISSION
90	START DATE (MMYY)	DATA DE INÍCIO (A↩ AMM)	FECHA INICIO (AA↩ MM)	DATE DE DEBUT-A↩ AMM

```
@param messageID  Message (1-90)
@param languageID 0=English Prompt, 1=Portuguese Prompt, 2=Spanish Prompt, 3=French Prompt
@param minLen  Minimum input length.  Cannot be less than 1
@param maxLen Maximum input length.  Cannot be greater than 15

Timeout is hardwired to 3 minutes


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

16.2.1.39 RETURN_CODE IDTechSDK.IDT_L100.pin_promptForAmountInputEnc (byte[ ] *message,* int *minLen,* int *maxLen*)

Prompt for Amount Input from Encrypted Message

Prompts for amount input using the secure message data

```
@param message  Encrypted Message
@param minLen  Minimum input length.  Cannot be less than 1
@param maxLen Maximum input length.  Cannot be greater than 15

Timeout is hardwired to 3 minutes
```

Returns

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

16.2.1.40 RETURN_CODE IDTechSDK.IDT_L100.pin_promptForKeyInput (int *messageID,* int *languageID,* bool *maskInput,* int *minLen,* int *maxLen*)

Prompt for Key Input

Prompts for a numeric key using the secure message according to the following table

Msg Id	English Prompt	Portuguese Prompt	Spanish Prompt	French Prompt
1	ENTER	ENTER	INGRESE	ENTREZ
2	REENTER	RE-INTRODUZIR	REINGRESE	RE-ENTREZ
3	ENTER YOUR	INTRODUZIR O SEU	INGRESE SU	ENTREZ VOTRE
4	REENTER YOUR	RE-INTRODUZIR O SEU	REINGRESE SU	RE-ENTREZ VOTRE
5	PLEASE ENTER	POR FAVOR DIGITE	POR FAVOR INGRESE	SVP ENTREZ
6	PLEASE REENTER	POR FAVO REENTRAR	POR FAVO REINGRESE	SVP RE-ENTREZ
7	PO NUMBER	NÚMERO PO	NUMERO PO	No COMMANDE
8	DRIVER ID	LICENÇA	LICENCIA	ID CONDUCTEUR
9	ODOMETER	ODOMETER	ODOMETRO	ODOMETRE
10	ID NUMBER	NÚMERO ID	NUMERO ID	No IDENT
11	EQUIP CODE	EQUIP CODE	CODIGO EQUIP	CODE EQUIPEMENT
12	DRIVERS ID	DRIVER ID	ID CONDUCTOR	ID CONDUCTEUR
13	JOB NUMBER	EMP NÚMERO	NUMERO EMP	No TRAVAIL
14	WORK ORDER	TRABALHO ORDEM	ORDEN TRABAJO	FICHE TRAVAIL
15	VEHICLE ID	ID VEÍCULO	ID VEHICULO	ID VEHICULE
16	ENTER DRIVER	ENTER DRIVER	INGRESE CONDUCTOR	ENTR CONDUCTEUR
17	ENTER DEPT	ENTER DEPT	INGRESE DEPT	ENTR DEPARTEMNT
18	ENTER PHONE	ADICIONAR PHONE	INGRESE TELEFONO	ENTR No TELEPH
19	ENTER ROUTE	ROUTE ADD	INGRESE RUTA	ENTREZ ROUTE
20	ENTER FLEET	ENTER FROTA	INGRESE FLOTA	ENTREZ PARC AUTO
21	ENTER JOB ID	ENTER JOB ID	INGRESE ID TRABAJO	ENTR ID TRAVAIL
22	ROUTE NUMBER	NÚMERO PATH	RUTA NUMERO	No ROUTE
23	ENTER USER ID	ENTER USER ID	INGRESE ID USUARIO	ID UTILISATEUR
24	FLEET NUMBER	NÚMERO DE FROTA	FLOTA NUMERO	No PARC AUTO
25	ENTER PRODUCT	ADICIONAR PRODUTO	INGRESE PRODUCTO	ENTREZ PRODUIT
26	DRIVER NUMBER	NÚMERO DRIVER	CONDUCTOR NUMERO	No CONDUCTEUR
27	ENTER LICENSE	ENTER LICENÇA	INGRESE LICENCIA	ENTREZ PERMIS
28	ENTER FLEET NO	ENTER NRO FROTA	INGRESE NRO FLOTA	ENT No PARC AUTO
29	ENTER CAR WASH	WASH ENTER	INGRESE LAVADO	ENTREZ LAVE-AUTO
30	ENTER VEHICLE	ENTER VEÍCULO	INGRESE VEHICULO	ENTREZ VEHICULE
31	ENTER TRAILER	TRAILER ENTER	INGRESE TRAILER	ENTREZ REMORQUE
32	ENTER ODOMETER	ENTER ODOMETER	INGRESE ODOMETRO	ENTREZ ODOMETRE
33	DRIVER LICENSE	CARTEIRA DE MOTORISTA	LICENCIA CONDUCTOR	PERMIS CONDUIRE
34	ENTER CUSTOMER	ENTER CLIENTE	INGRESE CLIENTE	ENTREZ CLIENT
35	VEHICLE NUMBER	NÚMERO DO VEÍCULO	VEHICULO NUMERO	No VEHICULE

Msg Id	English Prompt	Portuguese Prompt	Spanish Prompt	French Prompt
36	ENTER CUST DATA	ENTER CLIENTE INFO	INGRESE INFO CLIENTE	INFO CLIENT
37	REENTER DRIVID	REENTRAR DRIVER ID	REINGRESE ID CHOFER	RE-ENTR ID COND
38	ENTER USER DATA	ENTER INFO USUÁRIO	INGRESE INFO USUARIO	INFO UTILISATEUR
39	ENTER CUST CODE	ENTER CODE. CLIENTE	INGRESE COD. CLIENTE	ENTR CODE CLIENT
40	ENTER EMPLOYEE	ENTER FUNCIONÁRIO	INGRESE EMPLEADO	ENTREZ EMPLOYE
41	ENTER ID NUMBER	ENTER NÚMERO ID	INGRESE NUMERO ID	ENTREZ No ID
42	ENTER DRIVER ID	ENTER ID DRIVER	INGRESE ID CONDUCTOR	No CONDUCTEUR
43	ENTER FLEET PIN	ENTER PIN FROTA	INGRESE PIN DE FLOTA	NIP PARC AUTO
44	ODOMETER NUMBER	NÚMERO ODOMETER	ODOMETRO NUMERO	No ODOMETRE
45	ENTER DRIVER LIC	ENTER DRIVER LIC	INGRESE LIC CONDUCTOR	PERMIS CONDUIRE
46	ENTER TRAILER NO	NRO TRAILER ENTER	INGRESE NRO TRAILER	ENT No REMORQUE
47	REENTER VEHICLE	REENTRAR VEÍCULO	REINGRESE VEHICULO	RE-ENTR VEHICULE
48	ENTER VEHICLE ID	ENTER VEÍCULO ID	INGRESE ID VEHICULO	ENTR ID VEHICULE
49	ENTER BIRTH DATE	INSERIR DATA NAC	INGRESE FECHA NAC	ENT DT NAISSANCE
50	ENTER DOB MMDDYY	ENTER FDN MMDDYY	INGRESE FDN MMDDAA	NAISSANCE MMJJAA
51	ENTER FLEET DATA	ENTER FROTA INFO	INGRESE INFO DE FLOTA	INFO PARC AUTO
52	ENTER REFERENCE	ENTER REFERÊNCIA	INGRESE REFERENCIA	ENTREZ REFERENCE
53	ENTER AUTH NUMBR	ENTER NÚMERO AUT	INGRESE NUMERO AUT	No AUTORISATION
54	ENTER HUB NUMBER	ENTER HUB NRO	INGRESE NRO HUB	ENTREZ No NOYAU
55	ENTER HUBOMETER	MEDIDA PARA ENTRAR HUB	INGRESE MEDIDO DE HUB	COMPTEUR NOYAU
56	ENTER TRAILER ID	TRAILER ENTER ID	INGRESE ID TRAILER	ENT ID REMORQUE
57	ODOMETER READING	QUILOMETRAGEM	LECTURA ODOMETRO	LECTURE ODOMETRE
58	REENTER ODOMETER	REENTRAR ODOMETER	REINGRESE ODOMETRO	RE-ENT ODOMETRE
59	REENTER DRIV. ID	REENTRAR DRIVER ID	REINGRESE ID CHOFER	RE-ENT ID CONDUC
60	ENTER CUSTOMER ID	ENTER CLIENTE ID	INGRESE ID CLIENTE	ENTREZ ID CLIENT
61	ENTER CUST. ID	ENTER CLIENTE ID	INGRESE ID CLIENTE	ENTREZ ID CLIENT
62	ENTER ROUTE NUM	ENTER NUM ROUTE	INGRESE NUM RUTA	ENT No ROUTE
63	ENTER FLEET NUM	FROTA ENTER NUM	INGRESE NUM FLOTA	ENT No PARC AUTO

Msg Id	English Prompt	Portuguese Prompt	Spanish Prompt	French Prompt
64	FLEET PIN	FROTA PIN	PIN DE FLOTA	NIP PARC AUTO
65	DRIVER #	DRIVER #	CONDUCTOR #	CONDUCTEUR
66	ENTER DRIVER #	ENTER DRIVER #	INGRESE CONDUC↩ TOR #	ENT # CONDUCTEUR
67	VEHICLE #	VEÍCULO #	VEHICULO #	# VEHICULE
68	ENTER VEHICLE #	ENTER VEÍCULO #	INGRESE VEHICULO #	ENT # VEHICULE
69	JOB #	TRABALHO #	TRABAJO #	# TRAVAIL
70	ENTER JOB #	ENTER JOB #	INGRESE TRABAJO #	ENTREZ # TRAVAIL
71	DEPT NUMBER	NÚMERO DEPT	NUMERO DEPTO	No DEPARTEMENT
72	DEPARTMENT #	DEPARTAMENTO #	DEPARTAMENTO #	DEPARTEMENT
73	ENTER DEPT #	ENTER DEPT #	INGRESE DEPTO #	ENT# DEPARTEME↩ NT
74	LICENSE NUMBER	NÚMERO DE LICE↩ NÇA	NUMERO LICENCIA	No PERMIS
75	LICENSE #	LICENÇA #	LICENCIA #	# PERMIS
76	ENTER LICENSE #	ENTER LICENÇA #	INGRESE LICENCIA #	ENTREZ # PERMIS
77	DATA	INFO	INFO	INFO
78	ENTER DATA	ENTER INFO	INGRESE INFO	ENTREZ INFO
79	CUSTOMER DATA	CLIENTE INFO	INFO CLIENTE	INFO CLIENT
80	ID #	ID #	ID #	# ID
81	ENTER ID #	ENTER ID #	INGRESE ID #	ENTREZ # ID
82	USER ID	USER ID	ID USUARIO	ID UTILISATEUR
83	ROUTE #	ROUTE #	RUTA #	# ROUTE
84	ENTER ROUTE #	ADD ROUTE #	INGRESE RUTA #	ENTREZ # ROUTE
85	ENTER CARD NUM	ENTER NÚMERO DE CARTÃO	INGRESE NUM TAR↩ JETA	ENTREZ NO CARTE
86	EXP DATE(YYMM)	VALIDADE VAL (AA↩ MM)	FECHA EXP (AAMM)	DATE EXPIR(AAMM)
87	PHONE NUMBER	TELEFONE	NUMERO TELEFONO	NO TEL
88	CVV START DATE	CVV DATA DE INÍCIO	CVV FECHA INICIO	CVV DATE DE DEB↩ UT
89	ISSUE NUMBER	NÚMERO DE EMIS↩ SÃO	NUMERO DE EMISI↩ ON	NO DEMISSION
90	START DATE (MMYY)	DATA DE INÍCIO (A↩ AMM)	FECHA INICIO (AA↩ MM)	DATE DE DEBUT-A↩ AMM

```
        @param messageID  Message (1-90)
        @param languageID 0=English Prompt, 1=Portuguese Prompt, 2=Spanish Prompt, 3=French Prompt
        @param maskInput  TRUE = entry is masked with '*', FALSE = entry is displayed on keypad
        @param minLen  Minimum input length.  Cannot be less than 1
        @param maxLen Maximum input length.  Cannot be greater than 16

    Timeout is hardwired to 3 minutes

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

16.2.1.41 RETURN_CODE IDTechSDK.IDT_L100.pin_promptForKeyInputEnc (byte[] *message,* bool *maskInput,* int *minLen,* int *maxLen*)

Prompt for Key Input from Encrypted Message

Prompts for a numeric key using the secure encrypted message data

Parameters

message	Encrytped Secure Message
maskInput	TRUE = entry is masked with '*', FALSE = entry is displayed on keypad
minLen	Minimum input length. Cannot be less than 1
maxLen	Maximum input length. Cannot be greater than 16

Timeout is hardwired to 3 minutes

Returns

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

16.2.1.42 RETURN_CODE IDTechSDK.IDT_L100.pin_sendBeep (int *frequency,* int *duration*)

Send Beep

Executes a beep request.

Parameters

frequency	Frequency, range 200-20000Hz
duration	Duration, range 16-65535ms

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

16.2.1.43 RETURN_CODE IDTechSDK.IDT_L100.pin_setKeypressCapture (bool *showKeyValue*)

Set Keypress Capture Mode

If TRUE, each keypress will be broadcast with its value 0-9, B, C, E. Function completes after timeout, C, or E are received If FALSE, each keypress will generate a generic keypress notification, with final results being returned after E is pressed, or C/timeout encountered before then.

Parameters

showKeyValue	TRUE = broadcast each keypress value, FALSE = only broadcast a key was pressed, not it's value.

Returns

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

16.2.1.44 static String IDTechSDK.IDT_L100.SDK_Version ( ) [static]

SDK Version

- All Devices

Returns the current version of SDK

Returns

Framework version

16.2.1.45 static void IDTechSDK.IDT_L100.setCallback (CallBack *my_Callback*) [static]

Set Callback

Sets the class callback

16.2.1.46 static void IDTechSDK.IDT_L100.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

16.2.1.47 static void IDTechSDK.IDT_L100.setCommandTimeout (int *milliseconds*) [static]

Set Command Timeout

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

Parameters

milliseconds	Time

16.2.1.48 static bool IDTechSDK.IDT_L100.useSerialPort (int *port*) [static]

Use Serial Port Interface

Instructs SDK to attempt to use the Serial Port for communication with L100 using default baud rate

Parameters

port	Serial Port to use. Example COM1 = 1.

Returns

bool TRUE=successful, FALSE=failure

16.2.1.49 static bool IDTechSDK.IDT_L100.useSerialPort (int *port,* int *baud*) [static]

Use Serial Port Interface with baud rate L100

Parameters

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

Returns

bool TRUE=successful, FALSE=failure

16.2.1.50 static bool IDTechSDK.IDT_L100.useUSB ( ) [static]

Use USB Interface

Instructs SDK to attempt to use USB for communication with IDT_L100

16.2.2 Property Documentation

16.2.2.1 IDT_L100 IDTechSDK.IDT_L100.SharedController [static],[get]

Singleton Instance

Establishes an singleton instance of IDT_L100 class.

Returns

Instance of IDT_L100

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

- Source_Windows/IDT_L100.cs

16.3 IDTechSDK.IDT_NEO2 Class Reference

Public Member Functions

- bool ip_connectToSocket (string IP, bool isSecure=false)
- void ip_autoConnectToSocket ()
- void ip_monitorSocketConnectionStatus (bool enable, bool monitorConnect, int interval, int retryCount)
- RETURN_CODE device_getBatteryVoltage (ref string voltage)
- RETURN_CODE config_getSerialNumber (ref string response, string ip="")
- RETURN_CODE device_terminalInfo (ref byte[] tlv)
- RETURN_CODE config_setWhiteList (byte[] data, string ip="")
- RETURN_CODE config_setTrackFormat (byte option, string ip="")
- RETURN_CODE config_getTrackFormat (ref byte option, string ip="")
- RETURN_CODE config_setMasking (byte prePAN, byte postPAN, byte asciiMask, byte hexMask, bool maskExp, string ip="")
- RETURN_CODE config_getMasking (ref byte prePAN, ref byte postPAN, ref byte asciiMask, ref byte hex↩ Mask, ref bool maskExp, string ip="")
- RETURN_CODE config_getWhiteList (ref Dictionary< string, string > data, string ip="")
- RETURN_CODE config_getWiFiMACAddress (ref byte[] address, string ip="")
- RETURN_CODE config_getBLEMACAddress (ref byte[] address, string ip="")
- RETURN_CODE config_getEthernetMACAddress (ref byte[] address, string ip="")

- RETURN_CODE config_getNetworkConfiguration (ref bool isStatic, ref string address, ref string subnet, ref string gateway, ref string dns)
- RETURN_CODE device_getDeviceTime (ref DateTime time, string ip="")
- RETURN_CODE config_setBluetoothParameters (string name, string oldPW, string newPW)
- RETURN_CODE config_setSwipeandDone (byte swipeVal, byte doneVal, byte delay)
- RETURN_CODE config_getSwipeandDone (ref byte swipeVal, ref byte doneVal, ref byte delay)
- RETURN_CODE device_disBlueLED ()
- RETURN_CODE device_enaBlueLED (byte[] dataCmd)
- RETURN_CODE device_onYellowLED ()
- RETURN_CODE device_offYellowLED ()
- RETURN_CODE device_buzzerOnOff ()
- RETURN_CODE device_getLightSensorVal (ref UInt16 lightVal, string ip="")
- RETURN_CODE device_setSelfCheckTime (byte hour, byte minutes, string ip="")
- RETURN_CODE device_logRead (DeviceLogCallback callback, string ip="")
- RETURN_CODE device_logClear (string ip="")
- RETURN_CODE device_logEnable (bool enable, string ip="")
- RETURN_CODE device_extendedErrorCondition (bool enable, string ip="")
- RETURN_CODE device_getSelfCheckTime (ref byte hour, ref byte minutes, string ip="")
- RETURN_CODE config_setNetworkConfiguration (bool isStatic, string address, string subnet, string gateway, string dns)
- RETURN_CODE config_setWifiConfig (string mode, string ssid, string password, string ip, string netMask, string gateway)
- RETURN_CODE config_getWifiConfig (ref byte mode, ref string ssid, ref string password, ref string ip, ref string netMask, ref string gateway)
- RETURN_CODE device_getFirmwareVersion (ref string response, string ip="")
- RETURN_CODE device_getBootloaderVersion (ref string response, string ip="")
- RETURN_CODE config_setWirelessWorkMode (string mode)
- RETURN_CODE config_getWirelessWorkMode (ref byte mode)
- RETURN_CODE device_getRT1050FirmwareVersion (ref string response, string ip="")
- RETURN_CODE device_sendPAE (string command, ref string response, int timeout, string ip="")
- RETURN_CODE device_setTransArmorEncryption (byte[] cert)
- RETURN_CODE device_getTransArmorID (ref string TID)
- RETURN_CODE device_setTransArmorID (string TID)
- RETURN_CODE device_retrieveAIDList (ref byte[][] response)
- RETURN_CODE device_rebootDevice (string ip="")
- RETURN_CODE device_sendDataCommand (string cmd, bool calcLRC, ref byte[] response, string ip="")
- RETURN_CODE device_sendDataCommand_ext (string cmd, bool calcLRC, ref byte[] response, int timeout, bool noResponse, string ip="")
- RETURN_CODE msr_setMSRTrack (int val, string ip)
- RETURN_CODE msr_getMSRTrack (ref int val, string ip)
- RETURN_CODE msr_startMSRSwipe (int timeout, string ip="")
- RETURN_CODE msr_startMSRSwipe (int timeout, string ip, SwipeCallback swipeCallback, TimeoutCallback timeoutCallback, FailureCallback failureCallback)
- RETURN_CODE msr_cancelMSRSwipe (string ip="")
- RETURN_CODE emv_cancelTransaction ()
- RETURN_CODE ctls_cancelTransaction (string ip="")
- RETURN_CODE emv_startTransaction (double amount, double amtOther, int exponent, int type, int timeout, byte[] tags, bool forceOnline, bool isFastEMV=false)
- RETURN_CODE emv_activateTransaction (int timeout, byte[] tags, bool forceOnline, bool isFastEMV=false)
- RETURN_CODE emv_authenticateTransaction (byte[] updatedTLV)
- RETURN_CODE ctls_startTransaction (double amount, double amtOther, int exponent, int type, int timeout, byte[] tags, bool forceOnline, bool isFastEMV=false)
- RETURN_CODE ctls_activateTransaction (int timeout, byte[] tags, bool forceOnline, bool isFastEMV=false)
- RETURN_CODE ctls_updateBalance (byte statusCode, byte[] authCode, byte[] date, byte[] time)

- RETURN_CODE device_startTransaction (double amount, double amtOther, int exponent, int type, int timeout, byte[] tags, bool isFastEMV=false, string ip="")
- RETURN_CODE device_activateTransaction (int timeout, byte[] tags, bool isFastEMV=false)
- RETURN_CODE emv_completeTransaction (bool commError, byte[] authCode, byte[] iad, byte[] tlvScripts, byte[] tlv)
- RETURN_CODE emv_callbackResponseLCD (EMV_LCD_DISPLAY_MODE type, byte selection)
- RETURN_CODE emv_callbackResponsePIN (EMV_PIN_MODE type, byte[] KSN, byte[] PIN)
- RETURN_CODE emv_callbackResponseMSR (byte[] MSR)
- RETURN_CODE emv_getEMVKernelCheckValue (ref string response)
- RETURN_CODE emv_getEMVConfigurationCheckValue (ref string response)
- RETURN_CODE emv_getEMVKernelVersion (ref string response)
- RETURN_CODE emv_retrieveTransactionResult (byte[] tags, ref IDTTransactionData tlv)
- RETURN_CODE emv_removeApplicationData (byte[] AID)
- RETURN_CODE ctls_removeApplicationData (byte[] AID)
- RETURN_CODE emv_removeAllApplicationData ()
- RETURN_CODE emv_removeCAPK (byte[] capk)
- RETURN_CODE ctls_removeCAPK (byte[] capk)
- RETURN_CODE emv_removeAllCAPK ()
- RETURN_CODE ctls_removeAllCAPK ()
- RETURN_CODE emv_removeCRL (byte[] crlList)
- RETURN_CODE emv_removeAllCRL ()
- RETURN_CODE emv_retrieveApplicationData (byte[] AID, ref byte[] tlv)
- RETURN_CODE ctls_retrieveApplicationData (byte[] AID, ref byte[] tlv)
- RETURN_CODE emv_retrieveAIDList (ref byte[][] response)
- RETURN_CODE ctls_retrieveAIDList (ref byte[][] response)
- RETURN_CODE emv_retrieveCAPK (byte[] capk, ref byte[] key)
- RETURN_CODE ctls_retrieveCAPK (byte[] capk, ref byte[] key)
- RETURN_CODE emv_retrieveCAPKList (ref byte[] keys)
- RETURN_CODE ctls_retrieveCAPKList (ref byte[] keys)
- RETURN_CODE emv_retrieveCRLList (ref byte[] list)
- RETURN_CODE ctls_retrieveTerminalData (ref byte[] tlv)
- RETURN_CODE device_retrieveTerminalData (ref byte[] tlv)
- RETURN_CODE emv_retrieveTerminalData (ref byte[] tlv)
- RETURN_CODE emv_removeTerminalData ()
- RETURN_CODE emv_setApplicationData (byte[] name, byte[] tlv)
- RETURN_CODE emv_setApplicationData (byte[] tlv)
- RETURN_CODE ctls_setApplicationData (byte[] tlv)
- RETURN_CODE ctls_setDefaultConfiguration ()
- RETURN_CODE ctls_setConfigurationGroup (byte[] tlv)
- RETURN_CODE emv_setCAPK (byte[] key)
- RETURN_CODE ctls_setCAPK (byte[] key)
- RETURN_CODE emv_setCRL (byte[] list)
- RETURN_CODE emv_setTerminalData (byte[] tlv)
- RETURN_CODE ctls_setTerminalData (byte[] tlv)
- RETURN_CODE device_setTerminalData (byte[] tlv)
- RETURN_CODE device_sendVivoCommandP3 (byte command, byte subCommand, byte[] data, ref byte[] response, string ip="")
- RETURN_CODE device_sendVivoCommandP3_ext (byte command, byte subCommand, byte[] data, ref byte[] response, int timeout, bool noResponse, string ip="")
- RETURN_CODE device_sendVivoCommandP4 (byte command, byte subCommand, byte[] data, ref byte[] response, string ip="")
- RETURN_CODE device_sendVivoCommandP4_ext (byte command, byte subCommand, byte[] data, ref byte[] response, int timeout, bool noResponse, string ip="")
- RETURN_CODE emv_setTerminalMajorConfiguration (int configuration)
- RETURN_CODE emv_getTerminalMajorConfiguration (ref int configuration)

- RETURN_CODE device_lowPowerMode (bool stopMode, bool wakeOnTrans)
- RETURN_CODE device_pingDevice (string ip="")
- RETURN_CODE device_buzzer ()
- RETURN_CODE device_cancelTransaction ()
- RETURN_CODE device_resetTransaction ()
- RETURN_CODE device_controlLED (byte indexLED, byte control, string ip="")
- RETURN_CODE device_getProductType (ref byte[] type)
- RETURN_CODE device_getProcessorType (ref byte[] type)
- RETURN_CODE getHardwareInfor (ref string ascii)
- RETURN_CODE getUIDofMCU (ref string uid)
- RETURN_CODE getModuleVer (ref string moduleVer)
- RETURN_CODE getUsbBootLoader (ref string bootLoader)
- RETURN_CODE getRemoteKeyInjectionTO (ref int timeout)
- RETURN_CODE getCashTranRiskPara (ref byte[] tlv)
- RETURN_CODE getDrlReaderRiskPara (byte index, ref byte[] tlv)
- RETURN_CODE getMsrSecurePar (bool b0, bool b1, bool b2, bool b3, ref byte[] tlv)
- RETURN_CODE getWhiteList (ref byte[] list)
- RETURN_CODE emv_resetConfigurationGroup (int group)
- RETURN_CODE device_getMerchantRecord (int index, ref byte[] record)
- RETURN_CODE device_resetConfigurationGroup (int group)
- RETURN_CODE ctls_resetConfigurationGroup (int group)
- 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 device_SymmetricRKI (int type, string ip="")
- RETURN_CODE ctls_getConfigurationGroup (int group, ref byte[] tlv)
- RETURN_CODE ctls_removeConfigurationGroup (int group)
- RETURN_CODE ctls_getAllConfigurationGroups (ref byte[][] response)
- RETURN_CODE device_setBurstMode (byte mode)
- RETURN_CODE device_setPollMode (byte mode)
- RETURN_CODE device_setMerchantRecord (int index, bool enabled, string merchantID, string merchant↩ URL)
- RETURN_CODE device_startRKI (bool isTest, string ip="")
- RETURN_CODE device_updateFirmwareType (string path, FIRMWARE_TYPE type, FirmwareUpdate↩ Callback callback, string ip)
- RETURN_CODE device_updateFirmwareIP (string path, int type, FirmwareUpdateCallback callback, string ip)
- RETURN_CODE icc_getICCReaderStatus (ref byte status)
- RETURN_CODE icc_powerOffICC ()
- RETURN_CODE icc_powerOnICC (ref byte[] ATR, byte interfaces)
- RETURN_CODE icc_exchangeAPDU (string c_APDU, ref byte[] response)
- RETURN_CODE lcd_displayMessage (int lineNumber, string message)
- RETURN_CODE lcd_clearAllLines ()
- RETURN_CODE lcd_showScreen (string screenName, string ip="")
- RETURN_CODE lcd_createScreen (string screenName, ref UInt16 screenID, string ip="")
- RETURN_CODE lcd_cloneScreen (string screenName, string cloneName, ref UInt16 cloneID, string ip="")
- RETURN_CODE lcd_destroyScreen (string screenName, string ip="")
- RETURN_CODE lcd_getActiveScreen (ref string screenName, string ip="")
- RETURN_CODE lcd_getButtonEvent (ref UInt16 screenID, ref UInt16 objectID, ref string screenName, ref string objectName, ref bool isLongPress, string ip="")
- RETURN_CODE lcd_addButton (string screenName, string buttonName, byte type, byte alignment, UInt16 xCord, UInt16 yCord, string label, ref lcdItem returnItem, buttonCallback callback, string ip="")

- RETURN_CODE lcd_addEthernet (string screenName, string objectName, byte alignment, UInt16 xCord, UInt16 yCord, ref lcdItem returnItem, string ip)
- RETURN_CODE lcd_addLED (string screenName, string objectName, byte alignment, UInt16 xCord, UInt16 yCord, ref lcdItem returnItem, byte[] LED, string ip="")
- RETURN_CODE lcd_addImage (string screenName, string objectName, byte alignment, UInt16 xCord, UInt16 yCord, string filename, ref lcdItem returnItem, string ip="")
- void lcd_setButtonCallback (string screenName, string buttonName, buttonCallback callback, string ip)
- void lcd_setPinInputCallback (SwipeCallback callback, string ip="")
- RETURN_CODE device_enterStandbyMode (string ip="")
- void lcd_setPinSwipeCallback (SwipeCallback callback, string ip="")
- void lcd_setPinFailureCallback (FailureCallback callback, string ip="")
- void lcd_setPinTimeoutCallback (TimeoutCallback callback, string ip="")
- void lcd_setPinCancelPromptCallback (CancelPromptCallback callback, string ip="")
- RETURN_CODE lcd_addText (string screenName, string objectName, byte alignment, UInt16 xCord, UInt16 yCord, UInt16 width, UInt16 height, byte fontID, byte[] color, string label, ref lcdItem returnItem, string ip="")
- RETURN_CODE lcd_updateLabel (string screenName, string objectName, string label, string ip="")
- RETURN_CODE lcd_updateColor (string screenName, string objectName, byte[] color, string ip="")
- RETURN_CODE lcd_updatePosition (string screenName, string objectName, byte alignment, UInt16 new_xCord, UInt16 new_yCord, string ip="")
- RETURN_CODE lcd_removeItem (string screenName, string objectName, string ip="")
- RETURN_CODE lcd_storeScreenInfo (string ip="")
- RETURN_CODE lcd_loadScreenInfo (string ip="")
- RETURN_CODE lcd_clearScreenInfo (string ip="")
- RETURN_CODE lcd_getAllScreens (ref byte screenNumbers, ref Dictionary< UInt16, string > returnScreens, string ip="")
- RETURN_CODE lcd_getAllObjects (string screenName, ref byte objectNumbers, ref Dictionary< UInt16, string > returnObjects, string ip="")
- RETURN_CODE lcd_queryScreenbyName (string screenName, ref byte result, string ip="")
- RETURN_CODE lcd_queryObjectbyName (string objectName, ref byte objectNumbers, ref List< string > returnItems, string ip="")
- RETURN_CODE lcd_queryScreenbyID (UInt16 screenID, ref byte result, ref string screenName, string ip="")
- RETURN_CODE lcd_queryObjectbyID (UInt16 objectID, ref byte objectNumbers, ref List< string > returnItems, string ip="")
- RETURN_CODE lcd_setBacklight (byte backlightVal, string ip="")
- RETURN_CODE device_enablePassThrough (bool enablePassThrough)
- RETURN_CODE device_enableL100PassThrough (bool enablePassThrough)
- string device_getResponseCodeString (RETURN_CODE eCode)
- RETURN_CODE device_getTransactionResults (ref IDTTransactionData results)
- RETURN_CODE pin_getFunctionKey (int timeout, string ip="")
- RETURN_CODE pin_getFunctionKey (int timeout, string ip, SwipeCallback inputCallback, FailureCallback failureCallback, TimeoutCallback timeoutCallback, CancelPromptCallback cancelPromptCallback)
- RETURN_CODE pin_capturePin (int timeout, int type, string PAN, int minPIN, int maxPIN, string message, string ip="")
- RETURN_CODE pin_capturePin (int timeout, int type, string PAN, int minPIN, int maxPIN, string message, string ip, SwipeCallback inputCallback, FailureCallback failureCallback, TimeoutCallback timeoutCallback, CancelPromptCallback cancelPromptCallback)
- RETURN_CODE pin_sendBeep (string ip="")
- RETURN_CODE pin_getPanEntry (bool csc, bool expDate, bool ADR, bool ZIP, bool mod10, UInt16 timeout, bool encPANOnly=false, string ip="")
- RETURN_CODE pin_getPanEntry (bool csc, bool expDate, bool ADR, bool ZIP, bool mod10, UInt16 timeout, bool encPANOnly, string ip, SwipeCallback swipeCallback, TimeoutCallback timeoutCallback, CancelPromptCallback cancelPromptCallback)
- RETURN_CODE pin_promptForInput (int messageID, short timeout, string ip=null)
- RETURN_CODE pin_promptForInput (int messageID, short timeout, string ip, SwipeCallback inputCallback, SwipeCallback swipeCallback, FailureCallback failureCallback, TimeoutCallback timeoutCallback, CancelPromptCallback cancelPromptCallback)

- RETURN_CODE pin_promptForNumericKeyWithSwipe (short timeout, byte function, int minLen, int maxLen, string line1, string line2, byte[] signature, string ip="")
- RETURN_CODE pin_promptForNumericKeyWithSwipe (short timeout, byte function, int minLen, int max↩ Len, string line1, string line2, byte[] signature, string ip, SwipeCallback inputCallback, SwipeCallback swipe↩ Callback, FailureCallback failureCallback, TimeoutCallback timeoutCallback, CancelPromptCallback cancel↩ PromptCallback)
- RETURN_CODE pin_promptForAmount (int timeout, int minLen, int maxLen, string message, byte[] signature, string ip="")
- RETURN_CODE pin_promptForAmount (int timeout, int minLen, int maxLen, string message, byte[] signature, string ip, SwipeCallback inputCallback, FailureCallback failureCallback, TimeoutCallback timeout↩ Callback, CancelPromptCallback cancelPromptCallback)
- RETURN_CODE pin_cancelPINEntry (string ip="")
- void device_listenForNotifications (bool enable, string ip="")
- RETURN_CODE device_deleteDirectory (string filename, string ip="")
- RETURN_CODE device_listDirectory (string directoryName, bool recursive, bool onSD, ref string directory, string ip="")
- RETURN_CODE device_transferFile (string fileName, byte[] file, string ip="")
- RETURN_CODE device_deleteFile (string filename, string ip="")
- 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="")
- RETURN_CODE lcd_startScreenSaver (string name, string ip="")
- RETURN_CODE lcd_playAudio (string name, int type, string ip="")
- RETURN_CODE lcd_stopAudio (string ip="")
- RETURN_CODE lcd_startScanQR (ushort timeout, string ip="")
- RETURN_CODE lcd_stopScanQR (string ip="")
- RETURN_CODE lcd_startCameraCapture (ushort timeout, string ip="")
- RETURN_CODE lcd_stopCameraCapture (string ip="")

Static Public Member Functions

- static bool useSerialPort (int port)
- static bool useSerialPort (int port, int baud)
- static int getCommandTimeout ()
- static void setCommandTimeout (int milliseconds)
- static void **hasUI** (bool val)
- static bool ip_switchToSocket (string IP)
- static bool closeSocket (string IP)
- static List< string > ip_getSocketList ()
- static string getlastErrorString (string ip="")
- static bool ip_isConnected (string ip, int attempts=1, bool isSecure=false)
- static bool useUSB ()
- static void setCallback (CallBack my_Callback)
- static void setLongPressCallback (longPressCallback callback, string ip="")
- static void setCallbackIP (CallBackIP my_Callback, string ip="")
- static void setCallback (IntPtr my_Callback, SynchronizationContext context)
- static String SDK_Version ()
- static void emv_autoAuthenticate (bool authenticate)
- static void emv_autoAuthenticate (bool authenticate, byte[] tags)

- static void emv_allowFallback (bool allow)
- static RETURN_CODE device_updateFirmwareType (FIRMWARE_TYPE type, byte[] firmwareData, string ip="")
- static RETURN_CODE device_updateDeviceFirmware (byte[] firmwareData)
- static RETURN_CODE device_wakeDevice (string macAddress="", string ipAddress="")
- static RETURN_CODE lcd_clearDisplay ()
- static void lcd_retrieveMessage (DisplayMessages.DISPLAY_MESSAGE_LANGUAGE lang, DisplayMessages.DISPLAY_MESSAGE_IDENTIFIER id, ref string line1, ref string line2)

Static Public Attributes

- static bool **bypassUSBCheck** = false

Properties

- static IDT_NEO2 SharedController `[get]`

16.3.1 Detailed Description

Class for VP6300 reader

16.3.2 Member Function Documentation

16.3.2.1 static bool IDTechSDK.IDT_NEO2.closeSocket (string *IP*) `[static]`

Close Socket

Instructs SDK to attempt to use close existing TCP/IP socket used for communication with IDT_NEO2

Parameters

IP	Valid established IP address of the existing device. Must match original IP string exactly. Example: Connect to device as 192.168.1.155:50#Device_1. You must use that whole string, not just 192.168.1.155, or 192.168.1.155:50.

Return values

success	TRUE = socket closed, FALSE = socket not found

16.3.2.2 RETURN_CODE IDTechSDK.IDT_NEO2.config_getBLEMACAddress (ref byte[] *address*, string *ip* = " ")

Get Device BLE MAC Address

Parameters

address	6-byte MAC Address
ip	Optional: The callback function will only be applicable to the provided IP.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

16.3.2.3 RETURN_CODE IDTechSDK.IDT_NEO2.config_getEthernetMACAddress (ref byte[] *address,* string *ip* = " ")

Get Device Ethernet MAC Address

Parameters

address	6-byte MAC Address
ip	Optional: The callback function will only be applicable to the provided IP.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

16.3.2.4 RETURN_CODE IDTechSDK.IDT_NEO2.config_getMasking (ref byte *prePAN,* ref byte *postPAN,* ref byte *asciiMask,* ref byte *hexMask,* ref bool *maskExp,* string *ip* = " ")

Get Masking

Parameters

prePAN	the number of pre-PAN characters to display in the clear. Valid values 0-6. Default 4.
postPAN	the number of post-PAN characters to display in the clear. Valid values 0-6. Default 4.
asciiMask	Mask character for ASCII output masked data. Valid values 0x20-0x7E. Default 0x2A (∗) param hexMask Mask character for compressed numeric masked data. Valid values are 0x0A = 0x0F. Default 0x0C param maskExp TRUE = mask expiration data, FALSE = display expiration date in the clear.
ip	Optional: The callback function will only be applicable to the provided IP.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

16.3.2.5 RETURN_CODE IDTechSDK.IDT_NEO2.config_getNetworkConfiguration (ref bool *isStatic,* ref string *address,* ref string *subnet,* ref string *gateway,* ref string *dns*)

Get Device Network Configuration

Parameters

isStatic	TRUE = Static IP, FALSE = DHCP
address	Device IP Address as string. Example "192.168.1.15"
subnet	Device Subnet as string. Example "255.255.255.0"
gateway	Device Gateway address as a string. Example "8.8.8.8"
dns	Device DNS address as string. Example "192.168.1.22"

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

16.3.2.6 RETURN_CODE IDTechSDK.IDT_NEO2.config_getSerialNumber (ref string *response,* string *ip* = " ")

Polls device for Serial Number

Parameters

response	Returns Serial Number
ip	Optional: The callback function will only be applicable to the provided IP.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

16.3.2.7 RETURN_CODE IDTechSDK.IDT_NEO2.config_getSwipeandDone (ref byte *swipeVal,* ref byte *doneVal,* ref byte *delay*)

Get Swipe Button and Done Button Configuration

Read the button configuration from the ViVOpay Vendi reader.

Parameters

swipe	Value of Swipe Button. The value set to 01h, the Swipe Card switch is enabled. If value is set to 00h the Swipe Card switch is disabled.
done	Value of Done Button. If value is set to 01h, the Done switch is enabled. If value is set to 00h, the DONE switch is disabled.
delay	The Delay is an unsigned delay value in seconds. This should probably not be set to values larger than 30 seconds

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

16.3.2.8 RETURN_CODE IDTechSDK.IDT_NEO2.config_getTrackFormat (ref byte *option,* string *ip* = " ")

Get Track Format

Parameters

option	Format Options • 0 = No start/end sentinels, No LRC • 1 = Include start/end sentinels, No LRC • 2 = Include start/end sentinels, Include LRC
ip	Optional: The callback function will only be applicable to the provided IP.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

16.3.2.9 RETURN_CODE IDTechSDK.IDT_NEO2.config_getWhiteList (ref Dictionary< string, string > *data,* string *ip* = " ")

Get White List

Parameters

data	The white list data returned as Dictionary. Key = start bin range, Value = end bin range. When no range, Value = empty string.
ip	Optional: The callback function will only be applicable to the provided IP.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

16.3.2.10 RETURN_CODE IDTechSDK.IDT_NEO2.config_getWifiConfig (ref byte *mode,* ref string *ssid,* ref string *password,* ref string *ip,* ref string *netMask,* ref string *gateway*)

Get Wifi Configuration

Return all the network configurations.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

16.3.2.11 RETURN_CODE IDTechSDK.IDT_NEO2.config_getWiFiMACAddress (ref byte[ ] *address,* string *ip* = " ")

Get Device WiFi MAC Address

Parameters

address	6-byte MAC Address
ip	Optional: The callback function will only be applicable to the provided IP.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

16.3.2.12 RETURN_CODE IDTechSDK.IDT_NEO2.config_getWirelessWorkMode (ref byte *mode*)

Get Wireless Work Mode

Return Wireless Work Mode.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

16.3.2.13 RETURN_CODE IDTechSDK.IDT_NEO2.config_setBluetoothParameters (string *name,* string *oldPW,* string *newPW*)

Set BluetoothParameters

Sets the name and password for the BLE module.

Sending null to all three parameters resets the default password to 123456

Parameters

name	Device name, 1-25 characters
oldPW	Old password, as a six character string, example "123456"
newPW	New password, as a six character string, example "654321"

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

16.3.2.14 RETURN_CODE IDTechSDK.IDT_NEO2.config_setMasking (byte *prePAN,* byte *postPAN,* byte *asciiMask,* byte *hexMask,* bool *maskExp,* string *ip =* " ")

Set Masking

Parameters

prePAN	the number of pre-PAN characters to display in the clear. Valid values 0-6. Default 4.
postPAN	the number of post-PAN characters to display in the clear. Valid values 0-6. Default 4.
asciiMask	Mask character for ASCII output masked data. Valid values 0x20-0x7E. Default 0x2A (∗) param hexMask Mask character for compressed numeric masked data. Valid values are 0x0A = 0x0F. Default 0x0C param maskExp TRUE = mask expiration data, FALSE = display expiration date in the clear.
ip	Optional: The callback function will only be applicable to the provided IP.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

16.3.2.15 RETURN_CODE IDTechSDK.IDT_NEO2.config_setNetworkConfiguration (bool *isStatic,* string *address,* string *subnet,* string *gateway,* string *dns*)

Set Device Network Configuration

Parameters

isStatic	TRUE = Static IP, FALSE = DHCP
address	Device IP Address as string. Example "192.168.1.15"
subnet	Device Subnet as string. Example "255.255.255.0"
gateway	Device Gateway address as a string. Example "8.8.8.8"
dns	Device DNS address as string. Example "192.168.1.22"

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

16.3.2.16 RETURN_CODE IDTechSDK.IDT_NEO2.config_setSwipeandDone (byte *swipeVal,* byte *doneVal,* byte *delay*)

Set Swipe Button and Done Button Configuration

Configure the buttons on the ViVOpay Vendi reader. Both the SWIPE and DONE buttons can be independently disabled with this command. This command also sets the TAP disable time for when the SWIPE button is pressed. When the SWIPE button is enabled, the contactless reader is turned off for the programmed delay time so that a false read does not occur when the user wishes to swipe a dual contactless/MagStripe card.

Parameters

swipe	Value of Swipe Button. The value set to 01h, the Swipe Card switch is enabled. If value is set to 00h the Swipe Card switch is disabled.
done	Value of Done Button. If value is set to 01h, the Done switch is enabled. If value is set to 00h, the DONE switch is disabled.
delay	The Delay is an unsigned delay value in seconds. This should probably not be set to values larger than 30 seconds

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

16.3.2.17 RETURN_CODE IDTechSDK.IDT_NEO2.config_setTrackFormat (byte *option,* string *ip* = " ")

Set Track Format

Parameters

option	Format Options • 0 = No start/end sentinels, No LRC • 1 = Include start/end sentinels, No LRC • 2 = Include start/end sentinels, Include LRC
ip	Optional: The callback function will only be applicable to the provided IP.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

16.3.2.18 RETURN_CODE IDTechSDK.IDT_NEO2.config_setWhiteList (byte[] *data,* string *ip* = " ")

Set White List

Parameters

data	The signed white list data
ip	Optional: The callback function will only be applicable to the provided IP.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

16.3.2.19 RETURN_CODE IDTechSDK.IDT_NEO2.config_setWifiConfig (string *mode,* string *ssid,* string *password,* string *ip,* string *netMask,* string *gateway*)

Set Wifi Configuration

Set wifi configuration. Configuration will be saved in flash.

Parameters

mode	WiFi work mode 30h – Set NULL Mode, WiFi RF will be disabled 31h - Set Station Mode Default） 32h – Set SoftAP Mode 33h – Set SoftAP+Station Mode（Default）
ssid	SSID of the target AP as string.
password	Password of the target AP as string.
ip	Device IP Address as string.
netMask	Device Subnet as string.
gateway	Device Gateway address as a string.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

16.3.2.20 RETURN_CODE IDTechSDK.IDT_NEO2.config_setWirelessWorkMode (string *mode*)

Set Wireless Work Mode

Set wireless work mode(Wi-Fi/BLE). Configuration will be saved in flash.

Parameters

mode	Wireless work mode 30h – Set Wi-Fi TCP Server Mode 31h - Set Wi-Fi SSL Server Mode 32h – Set BLE Server Mode

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

16.3.2.21 RETURN_CODE IDTechSDK.IDT_NEO2.ctls_activateTransaction (int *timeout,* byte[ ] *tags,* bool *forceOnline,* bool *isFastEMV* = `false`)

Start a CTLS Transaction Request

Authorizes the CTLS transaction for an CTLS card

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

Parameters

forceOnline	TRUE = do not allow offline approval, FALSE = allow ICC to approve offline if terminal capable
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

16.3.2.22 RETURN_CODE IDTechSDK.IDT_NEO2.ctls_cancelTransaction (string *ip* = " ")

Cancel Transaction

Cancels the currently executing EMV or CTLS transaction.

Returns

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

16.3.2.23 RETURN_CODE IDTechSDK.IDT_NEO2.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()

16.3.2.24 RETURN_CODE IDTechSDK.IDT_NEO2.ctls_getConfigurationGroup (int *group,* ref byte[] *tlv*)

Get Configuration Group

Retrieves the Configuration for the specified Group. Group 0 = terminal settings.

Parameters

group	Configuration Group
tlv	return data

Returns

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

16.3.2.25 RETURN_CODE IDTechSDK.IDT_NEO2.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()

16.3.2.26 RETURN_CODE IDTechSDK.IDT_NEO2.ctls_removeAllCAPK ( )

Remove All Certificate Authority Public Key

Removes all the CAPK for CTLS

Returns

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

16.3.2.27 RETURN_CODE IDTechSDK.IDT_NEO2.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()

16.3.2.28 RETURN_CODE IDTechSDK.IDT_NEO2.ctls_removeCAPK (byte[] *capk*)

Remove Certificate Authority Public Key

Removes the CAPK for CTLS 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()

16.3.2.29 RETURN_CODE IDTechSDK.IDT_NEO2.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_NEO2::device_getResponseCodeString:()

16.3.2.30 RETURN_CODE IDTechSDK.IDT_NEO2.ctls_resetConfigurationGroup (int *group*)

Reset Configuration Group

This command allows resetting a dataset to its default configuration. If the file exists, it will be overwritten. If not, it will be created.

Parameters

group	Configuration Group

Return values

RETURN_CODE	Return codes listed as typedef enum in IDTCommon:RETURN_CODE. Values can be parsed with IDT_VP8800::device_getResponseCodeString:()

16.3.2.31 RETURN_CODE IDTechSDK.IDT_NEO2.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 TLV data objects: DFEE2D (group name) followed by 9F06 (AID), and DFEE4F (Interface 01 = CTLS, 02 = CONTACT)

Returns

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

16.3.2.32 RETURN_CODE IDTechSDK.IDT_NEO2.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()

16.3.2.33 RETURN_CODE IDTechSDK.IDT_NEO2.ctls_retrieveCAPK (byte[] *capk,* ref byte[] *key*)

Retrieve Certificate Authority Public Key

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

Parameters

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

Parameters

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()

16.3.2.34 RETURN_CODE IDTechSDK.IDT_NEO2.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()

16.3.2.35 RETURN_CODE IDTechSDK.IDT_NEO2.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_getConfiguraitonGroup(0).

Parameters

tlv	Response returned as a TLV

Returns

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

16.3.2.36 RETURN_CODE IDTechSDK.IDT_NEO2.ctls_setApplicationData (byte[ ] *tlv*)

Set Application Data by AID

Sets the Application Data as specified by TLV data

Parameters

tlv	Application data in TLV format The first tag of the TLV data must be the group number (DFEE2D). Group 0 = System, all other Groups = User The second tag of the TLV data must be the AID (9F06) If tag DFEE4F is included, it must have a value of 0x01 to be a CTLS AID

Example valid TLV, for Group #2, AID a0000000045010: "dfee2d01029f0607a0000000045010dfee4b0101dfee2e0110dfee4c0101dfee

Tags:

- DFEE2D : Group Number. Mandatory First Tag. 0 = System, 1-255 = User
- 9F06 : AID. Mandatory Second Tag
- DFEE4B : Partial AID Matching. 01 = allowed, 00 = Disabled. Mandatory for Visa
- DFEE4C : Application Flow, System: Never, User: Mandatory – 0x01 = MasterCard – 0x02 = AMEX – 0x03 = MasterCard w/Strip Application – 0x06 = Visa – 0x0D = Discover – 0x0E = JCB – 0x15 = Reserved – 0x16 = Reserved – 0x17 = Reserved
- DFEE4D = PPSE Disable, Optional
- DFEE2E = Max AID Length, Mandatory if DFEE4B included. Visa must be set to 16
- DFEE2F = Disable System Aid (no effect on User AID)
- DFEE4F = Interface Support. 01 = CTLS, 02 = Contact. If missing, defaults to CTLS

Returns

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

16.3.2.37 RETURN_CODE IDTechSDK.IDT_NEO2.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()

16.3.2.38 RETURN_CODE IDTechSDK.IDT_NEO2.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 (DFEE2D). A second tag must exist

Returns

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

16.3.2.39 RETURN_CODE IDTechSDK.IDT_NEO2.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()

16.3.2.40 RETURN_CODE IDTechSDK.IDT_NEO2.ctls_setTerminalData (byte[ ] *tlv*)

Set Terminal Data

Sets the Terminal Data for CTLS as specified by the TLV. The terminal global data is group 0. 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_NEO2::device_getResponseCodeString:()

16.3.2.41 RETURN_CODE IDTechSDK.IDT_NEO2.ctls_startTransaction (double *amount,* double *amtOther,* int *exponent,* int *type,* int *timeout,* byte[ ] *tags,* bool *forceOnline,* bool *isFastEMV* = `false`)

Start a CTLS Transaction Request

Authorizes the CTLS transaction for an CTLS card

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)
exponent	Number of characters after decimile point
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.
forceOnline	TRUE = do not allow offline approval, FALSE = allow ICC to approve offline if terminal capable
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

16.3.2.42 RETURN_CODE IDTechSDK.IDT_NEO2.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

16.3.2.43 RETURN_CODE IDTechSDK.IDT_NEO2.device_activateTransaction (int *timeout*, byte[ ] *tags*, bool *isFastEMV* = false)

Start a Transaction Request

Authorizes the transaction CTLS, MSR or Contact EMV

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

16.3.2.44 RETURN_CODE IDTechSDK.IDT_NEO2.device_buzzer ( )

Buzzer Device

Buzzer the reader. If success can hear one beep. Otherwise, returns timeout.

Returns

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

16.3.2.45 RETURN_CODE IDTechSDK.IDT_NEO2.device_buzzerOnOff ( )

Buzzer On/Off

Cause the buzzer to beep once.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

16.3.2.46 RETURN_CODE IDTechSDK.IDT_NEO2.device_cancelTransaction ( )

Cancel Transaction

Cancels the currently executing device transaction.

Returns

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

16.3.2.47 RETURN_CODE IDTechSDK.IDT_NEO2.device_controlLED (byte *indexLED,* byte *control,* string *ip* = " ")

Control LED

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

Parameters

indexLED	description as follows: 00h: LED 0 (Power LED) 01h: LED 1 02h: LED 2 03h: LED 3 10h: Single Tri-Color LED (Unipay III used) FFh: All 4 LEDs + Single Tri-Color LED Where the LEDs are numbered 0, 1, 2, 3 counting from the left. Note: If you are using pass-through mode to control the Power LED (LED 0), it is your responsibility to make sure that it behaves correctly.
control	description as follows: 00h: LED Off (LED 0∼4 + Tri-Color LED) 01h: LED On (LED 0∼4) 02h: Green Color (Tri-Color LED) 03h: Red Color (Tri-Color LED) 04h: Amber Color(Tri-Color LED)
ip	Optional: The callback function will only be applicable to the provided IP.

Returns

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

16.3.2.48 RETURN_CODE IDTechSDK.IDT_NEO2.device_controlUserInterface (byte[ ] *values*)

Control User Interface

Controls the User Interface: Display, Beep, LED

Parameters

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()

16.3.2.49 RETURN_CODE IDTechSDK.IDT_NEO2.device_deleteDirectory (string *filename,* string *ip =* " ")

Delete Directory

This command deletes an empty directory.

Parameters

filename	Complete path and file name of the directory you want to delete. You do not need to specify the root directory. Indicate subdirectories with a forward slash (/).
ip	Optional: IP address to execute command on (for IP connected devices)

Returns

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

16.3.2.50 RETURN_CODE IDTechSDK.IDT_NEO2.device_deleteFile (string *filename,* string *ip =* " ")

Delete File

This command deletes a file or group of files.

Parameters

filename	Complete path and file name of the file you want to delete. You do not need to specify the root directory. Indicate subdirectories with a forward slash (/).
ip	Optional: IP address to execute command on (for IP connected devices)

Returns

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

16.3.2.51 RETURN_CODE IDTechSDK.IDT_NEO2.device_disBlueLED ( )

Disable Blue LED Sequence

Stop the blue LEDs on the ViVOpay Vendi reader from flashing in left to right sequence and turn the LEDs off, and contactless function is disable at the same time.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

16.3.2.52 RETURN_CODE IDTechSDK.IDT_NEO2.device_enableL100PassThrough (bool *enablePassThrough*)

Enable L100 Pass Through

Enables Pass Through Mode for direct communication to L100 hook up to NEO II device

Parameters

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

Returns

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

16.3.2.53 RETURN_CODE IDTechSDK.IDT_NEO2.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()

16.3.2.54 RETURN_CODE IDTechSDK.IDT_NEO2.device_enaBlueLED (byte[ ] *dataCmd*)

Control the blue LED behavior on the Vendi reader.

Control the blue LED behavior on the Vendi reader.

Parameters

dataCmd	LED control. Minumum 4 bytes. Maximum 25 bytes. First byte is cycle. Next three bytes are sequence. Then sequence can repeat up to 8 times (24 bytes total for sequence) Byte 0 = Cycles (0 = Cycle once 1 = Repeat) Byte 1 = LED State Bitmap Bit - Description 8 = Left blue LED, 0 = off, 1 = on 7 = Center Blue LED, 0 = off, 1 = on 6 = Right Blue LED0 = off, 1 = on 5 = Yellow LED, 0 = off, 1 = on 4 = Reserved for future use 3 = Reserved for future use 2 = Reserved for future use 1 = Reserved for future use Byte2∼3 = Duration (Given in multiples of 10 millisecond) If cycle equals 1, more pairs would be after Byte 3.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

16.3.2.55 RETURN_CODE IDTechSDK.IDT_NEO2.device_enterStandbyMode (string *ip* = " ")

Enter Standby Mode

Puts unit into low power stand by mode

Parameters

ip	Optional: IP address to execute command on (for IP connected devices)

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

16.3.2.56 RETURN_CODE IDTechSDK.IDT_NEO2.device_extendedErrorCondition (bool *enable,* string *ip* = " ")

Enable/disable Extended Error Condition

Enables/disables extended error condition for commands 02-40, 61-xx, 62-xx, 83-41 when error is 0xD0A or 0xD0B String can be retrieved with getLastErrorString

Parameters

enable	TRUE = enable log, FALSE = disable log
ip	Optional: IP address to execute command on (for IP connected devices)

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

16.3.2.57 RETURN_CODE IDTechSDK.IDT_NEO2.device_getBatteryVoltage (ref string *voltage*)

Polls device for Battery Voltage

Parameters

voltage	Returns Battery Voltage string representing millivolts.

Return values

RETURN_CODE	• 0x0000: Success: no error - RETURN_CODE_DO_SUCCESS • 0x0001: Disconnect: no response from reader - RETURN_CODE_ERR_DISCONNECT • 0x0002: Invalid Response: invalid response data - RETURN_CODE_ERR_CMD_RESPONSE • 0x0003: Timeout: time out for task or CMD - RETURN_CODE_ERR_TIMEDOUT • 0x0004: Invalid Parameter: wrong parameter - RETURN_CODE_ERR_INVALID_PARAMETER • 0x0005: MSR Busy: SDK is doing MSR or ICC task - RETURN_CODE_SDK_BUSY_MSR • 0x0006: PINPad Busy: SDK is doing PINPad task - RETURN_CODE_SDK_BUSY_PINPAD • 0x0007: Unknown: Unknown error - RETURN_CODE_ERR_OTHER • 0x0100 through 0xFFFF refer to errorCode.getErrorString()

16.3.2.58 RETURN_CODE IDTechSDK.IDT_NEO2.device_getBootloaderVersion (ref string *response,* string *ip* = " ")

Polls device for Bootloader Version

Parameters

response	Response returned of Bootloader Version
ip	Optional: IP address to execute command on (for IP connected devices)

Returns

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

16.3.2.59 RETURN_CODE IDTechSDK.IDT_NEO2.device_getDeviceTime (ref DateTime *time,* string *ip =* " ")

Get Device Time

Parameters

time	Device Time

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

16.3.2.60 RETURN_CODE IDTechSDK.IDT_NEO2.device_getFirmwareVersion (ref string *response,* string *ip =* " ")

Polls device for Firmware Version

Parameters

response	Response returned of Firmware Version
ip	Optional: IP address to execute command on (for IP connected devices)

Returns

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

16.3.2.61 RETURN_CODE IDTechSDK.IDT_NEO2.device_getLightSensorVal (ref UInt16 *lightVal,* string *ip =* " ")

Get Light Sensor Value

Parameters

lightVal	Value of the light sensor
ip	Optional: IP address to execute command on (for IP connected devices)

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

16.3.2.62 RETURN_CODE IDTechSDK.IDT_NEO2.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()

16.3.2.63 RETURN_CODE IDTechSDK.IDT_NEO2.device_getProcessorType (ref byte[] *type*)

Get Processor Type Returns a processor type TLV

Parameters

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

Processor Type | Description

45 00 | ARM7/ LPC21xx 4D 00 | ARM Cortex-M4/ K21 Family 4E 00 | ARM Cortex-M4/ K81 Family

16.3.2.64 RETURN_CODE IDTechSDK.IDT_NEO2.device_getProductType (ref byte[] *type*)

Get Product Type Returns a "product type" value in a proprietary TLV

Parameters

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

Product Type | Description

42 37 00 | ViVOpay 5000 43 33 00 | ViVOpay 4500 43 35 00 | ViVOpay Vend 43 36 00 | Vendi (NEO) 43 37 00 | ViVOpay Kiosk1 (ATM1) 43 38 00 | Kiosk2 43 39 00 | Kiosk3 (NEO) 55 31 00 | UniPay 1.5 (NEO) 55 33 00 | UniPay III (NEO) 55 33 31 | VP3300, VP3300 OEM (NEO) (iBase/Cake same code) 55 33 32 | VP3300E(NEO) 55 33 33 | VP3300C(NEO) 55 33 34 | BTPay Mini (NEO) (UniPayIII + BLE) 56 31 00 | VP3600 56 32 00 | VP5200 56 33 00 | VP5300 56 34 00 | VP6300 56 35 00 | VP6800 56 36 00 | VP8300 56 37 00 | VP8310 56 38 00 | VP8800 56 39 00 | VP8810 56 40 00 | VP9000 44 30 00 | QX120 44 31 00 | Mx8Series 44 32 00 | NETs 44 33 00 | Magtek 44 35 00 | ICP

16.3.2.65 string IDTechSDK.IDT_NEO2.device_getResponseCodeString (RETURN_CODE *eCode*)

Get the description of response result.

Parameters

eCode	the response result.

Return values

the	string for description of response result

- case 0: "no error, beginning task";
- case 1: "no response from reader";
- case 2: "invalid response data";
- case 3: "time out for task or CMD";
- case 4: "wrong parameter";
- case 5: "SDK is doing MSR or ICC task";
- case 6: "SDK is doing PINPad task";
- case 7: "SDK is doing CTLS task";
- case 8: "SDK is doing EMV task";
- case 9: "SDK is doing Other task";
- case 10: "err response or data";
- case 11: "no reader attached";
- case 12: "mono audio is enabled";
- case 13: "did connection";
- case 14: "audio volume is too low";
- case 15: "task or CMD be canceled";
- case 16: "UF wrong string format";
- case 17: "UF file not found";
- case 18: "UF wrong file format";
- case 19: "Attempt to contact online host failed";
- case 20: "Attempt to perform RKI failed";
- case 0x300: "Key Type(TDES) of Session Key is not same as the related Master Key.";
- case 0x400: "Related Key was not loaded.";
- case 0x500: "Key Same.";
- case 0x501: "Key is all zero";
- case 0x502: "TR-31 format error";
- case 0x702: "PAN is Error Key.";

- case 0x705: "No Internal MSR PAN (or Internal MSR PAN is erased timeout)";
- case 0X0C01: "Incorrect Frame Tag";
- case 0X0C02: "Incorrect Frame Type";
- case 0X0C03: "Unknown Frame Type";
- case 0X0C04: "Unknown Command";
- case 0X0C05: "Unknown Sub-Command";
- case 0X0C06: "CRC Error";
- case 0X0C07: "Failed";
- case 0X0C08: "Timeout";
- case 0X0C0A: "Incorrect Parameter";
- case 0X0C0B: "Command Not Supported";
- case 0X0C0C: "Sub-Command Not Supported";
- case 0X0C0D: "Parameter Not Supported / Status Abort Command";
- case 0X0C0F: "Sub-Command Not Allowed";
- case 0X0D01: "Incorrect Header Tag";
- case 0X0D02: "Unknown Command";
- case 0X0D03: "Unknown Sub-Command";
- case 0X0D04: "CRC Error in Frame";
- case 0X0D05: "Incorrect Parameter";
- case 0X0D06: "Parameter Not Supported";
- case 0X0D07: "Mal-formatted Data";
- case 0X0D08: "Timeout";
- case 0X0D0A: "Failed / NACK";
- case 0X0D0B: "Command not Allowed";
- case 0X0D0C: "Sub-Command not Allowed";
- case 0X0D0D: "Buffer Overflow (Data Length too large for reader buffer)";
- case 0X0D0E: "User Interface Event";
- case 0X0D11: "Communication type not supported, VT-1, burst, etc.";
- case 0X0D12: "Secure interface is not functional or is in an intermediate state.";
- case 0X0D13: "Data field is not mod 8";
- case 0X0D14: "Pad 0x80 not found where expected";
- case 0X0D15: "Specified key type is invalid";
- case 0X0D1: "Could not retrieve key from the SAM(InitSecureComm)";
- case 0X0D17: "Hash code problem";
- case 0X0D18: "Could not store the key into the SAM(InstallKey)";
- case 0X0D19: "Frame is too large";

- case 0X0D1A: "Unit powered up in authentication state but POS must resend the InitSecureComm command";
- case 0X0D1B: "The EEPROM may not be initialized because SecCommInterface does not make sense";
- case 0X0D1C: "Problem encoding APDU";
- case 0X0D20: "Unsupported Index(ILM) SAM Transceiver error – problem communicating with the SAM(Key Mgr)";
- case 0X0D2: "Unexpected Sequence Counter in multiple frames for single bitmap(ILM) Length error in data returned from the SAM(Key Mgr)";
- case 0X0D22: "Improper bit map(ILM)";
- case 0X0D23: "Request Online Authorization";
- case 0X0D24: "ViVOCard3 raw data read successful";
- case 0X0D25: "Message index not available(ILM) ViVOcomm activate transaction card type(ViVOcomm)";
- case 0X0D26: "Version Information Mismatch(ILM)";
- case 0X0D27: "Not sending commands in correct index message index(ILM)";
- case 0X0D28: "Time out or next expected message not received(ILM)";
- case 0X0D29: "ILM languages not available for viewing(ILM)";
- case 0X0D2A: "Other language not supported(ILM)";
- case 0X0D41: "Unknown Error from SAM";
- case 0X0D42: "Invalid data detected by SAM";
- case 0X0D43: "Incomplete data detected by SAM";
- case 0X0D44: "Reserved";
- case 0X0D45: "Invalid key hash algorithm";
- case 0X0D46: "Invalid key encryption algorithm";
- case 0X0D47: "Invalid modulus length";
- case 0X0D48: "Invalid exponent";
- case 0X0D49: "Key already exists";
- case 0X0D4A: "No space for new RID";
- case 0X0D4B: "Key not found";
- case 0X0D4C: "Crypto not responding";
- case 0X0D4D: "Crypto communication error";
- case 0X0D4E: "Module-specific error for Key Manager";
- case 0X0D4F: "All key slots are full (maximum number of keys has been installed)";
- case 0X0D50: "Auto-Switch OK";
- case 0X0D51: "Auto-Switch failed";
- case 0X0D90: "Account DUKPT Key not exist";
- case 0X0D91: "Account DUKPT Key KSN exausted";
- case 0x0D00: "This Key had been loaded.";

- case 0x0E00: "Base Time was loaded.";
- case 0x0F00: "Encryption Or Decryption Failed.";
- case 0x1000: "Battery Low Warning (It is High Priority Response while Battery is Low.)";
- case 0x1800: "Send “Cancel Command” after send “Get Encrypted PIN” &”Get Numeric “& “Get Amount””;
- case 0x1900: "Press “Cancel” key after send “Get Encrypted PIN” &”Get Numeric “& “Get Amount””;
- case 0x30FF: "Security Chip is not connect";
- case 0x3000: "Security Chip is deactivation & Device is In Removal Legally State.";
- case 0x3101: "Security Chip is activation & Device is In Removal Legally State.";
- case 0x5500: "No Admin DUKPT Key.";
- case 0x5501: "Admin DUKPT Key STOP.";
- case 0x5502: "Admin DUKPT Key KSN is Error.";
- case 0x5503: "Get Authentication Code1 Failed.";
- case 0x5504: "Validate Authentication Code Error.";
- case 0x5505: "Encrypt or Decrypt data failed.";
- case 0x5506: "Not Support the New Key Type.";
- case 0x5507: "New Key Index is Error.";
- case 0x5508: "Step Error.";
- case 0x5509: "KSN Error";
- case 0x550A: "MAC Error.";
- case 0x550B: "Key Usage Error.";
- case 0x550C: "Mode Of Use Error.";
- case 0x550F: "Other Error.";
- case 0x6000: "Save or Config Failed / Or Read Config Error.";
- case 0x6200: "No Serial Number.";
- case 0x6900: "Invalid Command - Protocol is right, but task ID is invalid.";
- case 0x6A01: "Unsupported Command – Protocol and task ID are right, but command is invalid – In this State";
- case 0x6A00: "Unsupported Command - Protocol and task ID are right, but command is invalid.";
- case 0x6B00: "Unknown parameter in command - Protocol task ID and command are right, but parameter is invalid.";
- case 0x6C00: "Unknown parameter in command – Protocol task ID and command are right, but length is out of the requirement.";
- case 0x7200: "Device is suspend (MKSK suspend or press password suspend).";
- case 0x7300: "PIN DUKPT is STOP (21 bit 1).";
- case 0x7400: "Device is Busy.";
- case 0xE100: "Can not enter sleep mode";
- case 0xE200: "File has existed";

- case 0xE300: "File has not existed";
- case 0xE313: "IO line low -- Card error after session start";
- case 0xE400: "Open File Error";
- case 0xE500: "SmartCard Error";
- case 0xE600: "Get MSR Card data is error";
- case 0xE700: "Command time out";
- case 0xE800: "File read or write is error";
- case 0xE900: "Active 1850 error!";
- case 0xEA00: "Load bootloader error";
- case 0xEF00: "Protocol Error- STX or ETX or check error.";
- case 0xEB00: "Picture is not exist";
- case 0x2C02: "No Microprocessor ICC seated";
- case 0x2C06: "no card seated to request ATR";
- case 0x2D01: "Card Not Supported,";
- case 0x2D03: "Card Not Supported, wants CRC";
- case 0x690D: "Command not supported on reader without ICC support";
- case 0x8100: "ICC error time out on power-up";
- case 0x8200: "invalid TS character received - Wrong operation step";
- case 0x8300: "Decode MSR Error";
- case 0x8400: "TriMagII no Response";
- case 0x8500: "No Swipe MSR Card";
- case 0x8510: "No Financial Card";
- case 0x8600: "Unsupported F, D, or combination of F and D";
- case 0x8700: "protocol not supported EMV TD1 out of range";
- case 0x8800: "power not at proper level";
- case 0x8900: "ATR length too long";
- case 0x8B01: "EMV invalid TA1 byte value";
- case 0x8B02: "EMV TB1 required";
- case 0x8B03: "EMV Unsupported TB1 only 00 allowed";
- case 0x8B04: "EMV Card Error, invalid BWI or CWI";
- case 0x8B06: "EMV TB2 not allowed in ATR";
- case 0x8B07: "EMV TC2 out of range";
- case 0x8B08: "EMV TC2 out of range";
- case 0x8B09: "per EMV96 TA3 must be $>$ 0xF";
- case 0x8B10: "ICC error on power-up";
- case 0x8B11: "EMV T=1 then TB3 required";

- case 0x8B12: "Card Error, invalid BWI or CWI";
- case 0x8B13: "Card Error, invalid BWI or CWI";
- case 0x8B17: "EMV TC1/TB3 conflict-";
- case 0x8B20: "EMV TD2 out of range must be T=1";
- case 0x8C00: "TCK error";
- case 0xA304: "connector has no voltage setting";
- case 0xA305: "ICC error on power-up invalid (SBLK(IFSD) exchange";
- case 0xE301: "ICC error after session start";
- case 0xFF00: "Request to go online";
- case 0xFF01: "EMV: Accept the offline transaction";
- case 0xFF02: "EMV: Decline the offline transaction";
- case 0xFF03: "EMV: Accept the online transaction";
- case 0xFF04: "EMV: Decline the online transaction";
- case 0xFF05: "EMV: Application may fallback to magstripe technology";
- case 0xFF06: "EMV: ICC detected tah the conditions of use are not satisfied";
- case 0xFF07: "EMV: ICC didn't accept transaction";
- case 0xFF08: "EMV: Transaction was cancelled";
- case 0xFF09: "EMV: Application was not selected by kernel or ICC format error or ICC missing data error";
- case 0xFF0A: "EMV: Transaction is terminated";
- case 0xFF0B: "EMV: Other EMV Error";
- case 0xFFFF: "NO RESPONSE";
- case 0xF002: "ICC communication timeout";
- case 0xF003: "ICC communication Error";
- case 0xF00F: "ICC Card Seated and Highest Priority, disable MSR work request";
- case 0xF200: "AID List / Application Data is not exist";
- case 0xF201: "Terminal Data is not exist";
- case 0xF202: "TLV format is error";
- case 0xF203: "AID List is full";
- case 0xF204: "Any CA Key is not exist";
- case 0xF205: "CA Key RID is not exist";
- case 0xF206: "CA Key Index it not exist";
- case 0xF207: "CA Key is full";
- case 0xF208: "CA Key Hash Value is Error";
- case 0xF209: "Transaction format error";
- case 0xF20A: "The command will not be processing";
- case 0xF20B: "CRL is not exist";

- case 0xF20C: "CRL number exceed max number";
- case 0xF20D: "Amount,Other Amount,Trasaction Type are missing";
- case 0xF20E: "The Identification of algorithm is mistake";
- case 0xF20F: "No Financial Card";
- case 0xF210: "In Encrypt Result state, TLV total Length is greater than Max Length";
- case 0x1001: "INVALID ARG";
- case 0x1002: "FILE_OPEN_FAILED";
- case 0x1003: "FILE OPERATION_FAILED";
- case 0x2001: "MEMORY_NOT_ENOUGH";
- case 0x3002: "SMARTCARD_FAIL";
- case 0x3003: "SMARTCARD_INIT_FAILED";
- case 0x3004: "FALLBACK_SITUATION";
- case 0x3005: "SMARTCARD_ABSENT";
- case 0x3006: "SMARTCARD_TIMEOUT";
- case 0x5001: "EMV_PARSING_TAGS_FAILED";
- case 0x5002: "EMV_DUPLICATE_CARD_DATA_ELEMENT";
- case 0x5003: "EMV_DATA_FORMAT_INCORRECT";
- case 0x5004: "EMV_NO_TERM_APP";
- case 0x5005: "EMV_NO_MATCHING_APP";
- case 0x5006: "EMV_MISSING_MANDATORY_OBJECT";
- case 0x5007: "EMV_APP_SELECTION_RETRY";
- case 0x5008: "EMV_GET_AMOUNT_ERROR";
- case 0x5009: "EMV_CARD_REJECTED";
- case 0x5010: "EMV_AIP_NOT_RECEIVED";
- case 0x5011: "EMV_AFL_NOT_RECEIVED";
- case 0x5012: "EMV_AFL_LEN_OUT_OF_RANGE";
- case 0x5013: "EMV_SFI_OUT_OF_RANGE";
- case 0x5014: "EMV_AFL_INCORRECT";
- case 0x5015: "EMV_EXP_DATE_INCORRECT";
- case 0x5016: "EMV_EFF_DATE_INCORRECT";
- case 0x5017: "EMV_ISS_COD_TBL_OUT_OF_RANGE";
- case 0x5018: "EMV_CRYPTOGRAM_TYPE_INCORRECT";
- case 0x5019: "EMV_PSE_NOT_SUPPORTED_BY_CARD";
- case 0x5020: "EMV_USER_SELECTED_LANGUAGE";
- case 0x5021: "EMV_SERVICE_NOT_ALLOWED";
- case 0x5022: "EMV_NO_TAG_FOUND";

- case 0x5023: "EMV_CARD_BLOCKED";
- case 0x5024: "EMV_LEN_INCORRECT";
- case 0x5025: "CARD_COM_ERROR";
- case 0x5026: "EMV_TSC_NOT_INCREASED";
- case 0x5027: "EMV_HASH_INCORRECT";
- case 0x5028: "EMV_NO_ARC";
- case 0x5029: "EMV_INVALID_ARC";
- case 0x5030: "EMV_NO_ONLINE_COMM";
- case 0x5031: "TRAN_TYPE_INCORRECT";
- case 0x5032: "EMV_APP_NO_SUPPORT";
- case 0x5033: "EMV_APP_NOT_SELECT";
- case 0x5034: "EMV_LANG_NOT_SELECT";
- case 0x5035: "EMV_NO_TERM_DATA";
- case 0x6001: "CVM_TYPE_UNKNOWN";
- case 0x6002: "CVM_AIP_NOT_SUPPORTED";
- case 0x6003: "CVM_TAG_8E_MISSING";
- case 0x6004: "CVM_TAG_8E_FORMAT_ERROR";
- case 0x6005: "CVM_CODE_IS_NOT_SUPPORTED";
- case 0x6006: "CVM_COND_CODE_IS_NOT_SUPPORTED";
- case 0x6007: "NO_MORE_CVM";
- case 0x6008: "PIN_BYPASSED_BEFORE";
- case 0x7001: "PK_BUFFER_SIZE_TOO_BIG";
- case 0x7002: "PK_FILE_WRITE_ERROR";
- case 0x7003: "PK_HASH_ERROR";
- case 0x8001: "NO_CARD_HOLDER_CONFIRMATION";
- case 0x8002: "GET_ONLINE_PIN";
- case 0xD000: "Data not exist";
- case 0xD001: "Data access error";
- case 0xD100: "RID not exist";
- case 0xD101: "RID existed";
- case 0xD102: "Index not exist";
- case 0xD200: "Maximum exceeded";
- case 0xD201: "Hash error";
- case 0xD205: "System Busy";
- case 0x0E01: "Unable to go online";
- case 0x0E02: "Technical Issue";

- case 0x0E03: "Declined";
- case 0x0E04: "Issuer Referral transaction";
- case 0x0F01: "Decline the online transaction";
- case 0x0F02: "Request to go online";
- case 0x0F03: "Transaction is terminated";
- case 0x0F05: "Application was not selected by kernel or ICC format error or ICC missing data error";
- case 0x0F07: "ICC didn't accept transaction";
- case 0x0F0A: "Application may fallback to magstripe technology";
- case 0x0F0C: "Transaction was cancelled";
- case 0x0F0D: "Timeout";
- case 0x0F0F: "Other EMV Error";
- case 0x0F10: "Accept the offline transaction";
- case 0x0F11: "Decline the offline transaction";
- case 0x0F21: "ICC detected tah the conditions of use are not satisfied";
- case 0x0F22: "No app were found on card matching terminal configuration";
- case 0x0F23: "Terminal file does not exist";
- case 0x0F24: "CAPK file does not exist";
- case 0x0F25: "CRL Entry does not exist";
- case 0x0FFE: "code when blocking is disabled";
- case 0x0FFF: "code when command is not applicable on the selected device";
- case 0xF005: "ICC Encrypted C-APDU Data Structure Length Error Or Format Error.";
- case 0xBBE0: "CM100 Success";
- case 0xBBE1: "CM100 Parameter Error";
- case 0xBBE2: "CM100 Low Output Buffer";
- case 0xBBE3: "CM100 Card Not Found";
- case 0xBBE4: "CM100 Collision Card Exists";
- case 0xBBE5: "CM100 Too Many Cards Exist";
- case 0xBBE6: "CM100 Saved Data Does Not Exist";
- case 0xBBE8: "CM100 No Data Available";
- case 0xBBE9: "CM100 Invalid CID Returned";
- case 0xBBEA: "CM100 Invalid Card Exists";
- case 0xBBEC: "CM100 Command Unsupported";
- case 0xBBED: "CM100 Error In Command Process";
- case 0xBBEE: "CM100 Invalid Command";
- case 0X9031: "Unknown command";
- case 0X9032: "Wrong parameter (such as the length of the command is incorrect)";

- case 0X9038: "Wait (the command couldn't be finished in BWT)";
- case 0X9039: "Busy (a previously command has not been finished)";
- case 0X903A: "Number of retries over limit";
- case 0X9040: "Invalid Manufacturing system data";
- case 0X9041: "Not authenticated";
- case 0X9042: "Invalid Master DUKPT Key";
- case 0X9043: "Invalid MAC Key";
- case 0X9044: "Reserved for future use";
- case 0X9045: "Reserved for future use";
- case 0X9046: "Invalid DATA DUKPT Key";
- case 0X9047: "Invalid PIN Pairing DUKPT Key";
- case 0X9048: "Invalid DATA Pairing DUKPT Key";
- case 0X9049: "No nonce generated";
- case 0X9949: "No GUID available. Perform getVersion first.";
- case 0X9950: "MAC Calculation unsuccessful. Check BDK value.";
- case 0X904A: "Not ready";
- case 0X904B: "Not MAC data";
- case 0X9050: "Invalid Certificate";
- case 0X9051: "Duplicate key detected";
- case 0X9052: "AT checks failed";
- case 0X9053: "TR34 checks failed";
- case 0X9054: "TR31 checks failed";
- case 0X9055: "MAC checks failed";
- case 0X9056: "Firmware download failed";
- case 0X9060: "Log is full";
- case 0X9061: "Removal sensor unengaged";
- case 0X9062: "Any hardware problems";
- case 0X9070: "ICC communication timeout";
- case 0X9071: "ICC data error (such check sum error)";
- case 0X9072: "Smart Card not powered up";

16.3.2.66 RETURN_CODE IDTechSDK.IDT_NEO2.device_getRT1050FirmwareVersion (ref string *response,* string *ip* = " ")

Get RT1050 Firmware Version

Parameters

response	Response returned of Firmware Version
ip	Optional: IP address to execute command on (for IP connected devices)

Returns

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

16.3.2.67 RETURN_CODE IDTechSDK.IDT_NEO2.device_getSelfCheckTime (ref byte *hour,* ref byte *minutes,* string *ip* = " ")

Get Self-Check Time

Gest the specific time for 24hrs self-check in Coordinated Universal Time

Parameters

Hours,00-23h	
Minutes	00-59h
ip	Optional: IP address to execute command on (for IP connected devices)

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

16.3.2.68 RETURN_CODE IDTechSDK.IDT_NEO2.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

16.3.2.69 RETURN_CODE IDTechSDK.IDT_NEO2.device_getTransArmorID (ref string *TID*)

Get TransArmor ID

Parameters

TID	TransArmor ID

Returns

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

16.3.2.70 RETURN_CODE IDTechSDK.IDT_NEO2.device_listDirectory (string *directoryName,* bool *recursive,* bool *onSD,* ref string *directory,* string *ip* = " ")

List Directory

This command retrieves a directory listing of user accessible files from the reader.

Parameters

directoryName	Directory Name. If null, root directory is listed
recursive	Included sub-directories
onSD	TRUE = use flash storage
directory	The returned directory information
ip	Optional: IP address to execute command on (for IP connected devices)

Returns

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

16.3.2.71 void IDTechSDK.IDT_NEO2.device_listenForNotifications (bool *enable,* string *ip =* " ")

Listen for Notifications

Instructs SDK to listen for unsolicited data

Parameters

enable	TRUE = Listen, FALSE = Don't Listen

16.3.2.72 RETURN_CODE IDTechSDK.IDT_NEO2.device_logClear (string *ip =* " ")

Clear Log

Instructs device to delete all log data

Parameters

ip	Optional: IP address to execute command on (for IP connected devices)

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

16.3.2.73 RETURN_CODE IDTechSDK.IDT_NEO2.device_logEnable (bool *enable,* string *ip =* " ")

Enable Log

Instructs device to enable log

Parameters

enable	TRUE = enable log, FALSE = disable log
ip	Optional: IP address to execute command on (for IP connected devices)

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

16.3.2.74 RETURN_CODE IDTechSDK.IDT_NEO2.device_logRead (DeviceLogCallback *callback,* string *ip* = " ")

Read Log

Instructs device to output all log data

Parameters

callback	DeviceLogCallback that will receive all log entries. If null, will be sent to MessageCallback, DeviceState.LogEvent
ip	Optional: IP address to execute command on (for IP connected devices)

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

16.3.2.75 RETURN_CODE IDTechSDK.IDT_NEO2.device_lowPowerMode (bool *stopMode,* bool *wakeOnTrans*)

Enter Low Power Mode

Puts terminal in sleep mor stop mode, with the option to wak on swipe/tap

Parameters

stopMode	TRUE = Stop Mode (POR required), FALSE = Sleep Mode (resume from last instruction)
wakeOnTrans	TRUE = Swipe/Tap will wake from low power, FALSE = will not wake on power from swipe/tap

Return values

RETURN_CODE	Return codes listed as typedef enum in IDTCommon:RETURN_CODE. Values can be parsed with IDT_NEO2::device_getResponseCodeString:()

16.3.2.76 RETURN_CODE IDTechSDK.IDT_NEO2.device_offYellowLED ( )

Turn Off Yellow LED

Turn off the ViVOpay Vendi reader yellow LED. This LED is located below the three blue LEDs.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

16.3.2.77 RETURN_CODE IDTechSDK.IDT_NEO2.device_onYellowLED ( )

Turn On Yellow LED

Turn On the ViVOpay Vendi reader yellow LED. This LED is located below the three blue LEDs.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

16.3.2.78 RETURN_CODE IDTechSDK.IDT_NEO2.device_pingDevice (string *ip* = " ")

Ping Device

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

Parameters

ip	Optional: The callback function will only be applicable to the provided IP.

Returns

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

16.3.2.79 RETURN_CODE IDTechSDK.IDT_NEO2.device_rebootDevice (string *ip* = " ")

Reboot Device

Performs a reboot of the device

Parameters

ip	Optional: The callback function will only be applicable to the provided IP.

16.3.2.80 RETURN_CODE IDTechSDK.IDT_NEO2.device_resetConfigurationGroup (int *group*)

Reset Configuration Group

This command allows resetting a dataset to its default configuration. If the file exists, it will be overwritten. If not, it will be created.

Parameters

group	Configuration Group

Return values

RETURN_CODE	Return codes listed as typedef enum in IDTCommon:RETURN_CODE. Values can be parsed with IDT_VP8800::device_getResponseCodeString:()

16.3.2.81 RETURN_CODE IDTechSDK.IDT_NEO2.device_resetTransaction ( )

Reset Transaction

Abruptly terminates the currently executing device transaction. Does not produce any further transaction data or notifications after executing.

Returns

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

16.3.2.82 RETURN_CODE IDTechSDK.IDT_NEO2.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 2-tag TLV data objects: DFEE2D (group name) followed by 9F06 (AID)

Returns

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

16.3.2.83 RETURN_CODE IDTechSDK.IDT_NEO2.device_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_getConfiguraitonGroup(0).

Parameters

tlv	Response returned as a TLV

Returns

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

16.3.2.84 RETURN_CODE IDTechSDK.IDT_NEO2.device_sendDataCommand (string *cmd,* bool *calcLRC,* ref byte[] *response,* string *ip* = " ")

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
ip	Optional IP parameter

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

16.3.2.85 RETURN_CODE IDTechSDK.IDT_NEO2.device_sendDataCommand_ext (string *cmd,* bool *calcLRC,* ref byte[] *response,* int *timeout,* bool *noResponse,* string *ip* = " ")

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}'
ip	Optional IP parameter

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

16.3.2.86 RETURN_CODE IDTechSDK.IDT_NEO2.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()

16.3.2.87 RETURN_CODE IDTechSDK.IDT_NEO2.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()

16.3.2.88 RETURN_CODE IDTechSDK.IDT_NEO2.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()

16.3.2.89 RETURN_CODE IDTechSDK.IDT_NEO2.device_sendVivoCommandP3 (byte *command,* byte *subCommand,* byte[ ] *data,* ref byte[ ] *response,* string *ip* = " ")

Send Vivo Command Protocol 3

Sends a protocol 3 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()

16.3.2.90 RETURN_CODE IDTechSDK.IDT_NEO2.device_sendVivoCommandP3_ext (byte *command,* byte *subCommand,* byte[ ] *data,* ref byte[ ] *response,* int *timeout,* bool *noResponse,* string *ip* = " ")

Send Vivo Command Protocol 3 Extended

Sends a protocol 3 command to Vivo readers (IDG/NEO)

Parameters

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

Parameters

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()

16.3.2.91 RETURN_CODE IDTechSDK.IDT_NEO2.device_sendVivoCommandP4 (byte *command,* byte *subCommand,* byte[ ] *data,* ref byte[ ] *response,* string *ip =* " ")

Send Vivo Command Protocol 4

Sends a protocol 4 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()

16.3.2.92 RETURN_CODE IDTechSDK.IDT_NEO2.device_sendVivoCommandP4_ext (byte *command,* byte *subCommand,* byte[ ] *data,* ref byte[ ] *response,* int *timeout,* bool *noResponse,* string *ip =* " ")

Send Vivo Command Protocol 4 Extended

Sends a protocol 4 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()

16.3.2.93 RETURN_CODE IDTechSDK.IDT_NEO2.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()

16.3.2.94 RETURN_CODE IDTechSDK.IDT_NEO2.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()

16.3.2.95 RETURN_CODE IDTechSDK.IDT_NEO2.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()

16.3.2.96 RETURN_CODE IDTechSDK.IDT_NEO2.device_setSelfCheckTime (byte *hour,* byte *minutes,* string *ip* = " ")

Set Self-Check Time

Set a specific time for 24hrs self-check in Coordinated Universal Time

Parameters

Hours	00-23h
Minutes	00-59h
ip	Optional: IP address to execute command on (for IP connected devices)

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

16.3.2.97 RETURN_CODE IDTechSDK.IDT_NEO2.device_setTerminalData (byte[ ] *tlv*)

Set Terminal Data

Sets the Terminal Data for CTLS as specified by the TLV. The terminal global data is group 0. 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_NEO2::device_getResponseCodeString:()

16.3.2.98 RETURN_CODE IDTechSDK.IDT_NEO2.device_setTransArmorEncryption (byte[ ] *cert*)

Set TransArmor Encryption

Parameters

cert	Certificate in PEM format

Returns

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

16.3.2.99 RETURN_CODE IDTechSDK.IDT_NEO2.device_setTransArmorID (string *TID*)

Set TransArmor ID

Parameters

TID	TransArmor ID

Returns

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

16.3.2.100 RETURN_CODE IDTechSDK.IDT_NEO2.device_startRKI (bool *isTest,* string *ip* = " ")

Start Remote Key Injection

Starts a remote key injection request with IDTech RKI servers.

Parameters

isTest	TRUE = Demo Device, FALSE = Production Device
ip	Optional IP address

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

16.3.2.101 RETURN_CODE IDTechSDK.IDT_NEO2.device_startTransaction (double *amount,* double *amtOther,* int *exponent,* int *type,* int *timeout,* byte[ ] *tags,* bool *isFastEMV* = `false`, string *ip* = " ")

Start a Transaction Request

Authorizes the transaction CTLS, MSR or Contact EMV

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)
exponent	Number of characters after decimile point
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
ip	Optional IP parameter

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

16.3.2.102 RETURN_CODE IDTechSDK.IDT_NEO2.device_SymmetricRKI (int *type,* string *ip* = " ")

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
ip	Optional IP address

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

16.3.2.103 RETURN_CODE IDTechSDK.IDT_NEO2.device_terminalInfo (ref byte[ ] *tlv*)

Retrieve Terminal Info

Parameters

tlv	TLV Tags assigned as follows: • 0x01 = Date Time, 6 bytes Y M D H M S • 0x02 = Device Model Number • 0x03 = Firmware Version • 0x04 = hardware Version • 0x05 = Serial Number • 0x06 Last swipe UTC/RTC • 0x07 Life time total swipe • 0x08 Last Dip (ICC) UTC/RTC • 0x09 Lifetime Total Dip (ICC) • 0x0A Last Tap (CL) UTC/RTC • 0x0B Lifetime Total Tap (CL) • 0x0C Reboot Count • 0x0D Device Uptime since Last Reboot • 0x0E Device Lifetime Uptime • 0x0F Tamper Status

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

16.3.2.104 RETURN_CODE IDTechSDK.IDT_NEO2.device_transferFile (string *fileName,* byte[ ] *file,* string *ip* = " ")

Transfer File

This command transfers a data file to the reader.

Parameters

fileName	Filename. The data for this command is a ASCII string with the complete path and file name you want to create. You do not need to specify the root directory. Indicate subdirectories with a forward slash (/).
file	The data file.
ip	Optional: IP address to execute command on (for IP connected devices)

Returns

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

16.3.2.105 static RETURN_CODE IDTechSDK.IDT_NEO2.device_updateDeviceFirmware (byte[ ] *firmwareData*) [static]

Update K81 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;
    }
}
```

16.3.2.106 RETURN_CODE IDTechSDK.IDT_NEO2.device_updateFirmwareIP (string *path,* int *type,* FirmwareUpdateCallback *callback,* string *ip*)

Update Firmware

Updates the firmware over IP .

Parameters

path	Local filepath to the signed binary data of a firmware file provided by IDTech
type	0 = K81, 1 = 1050
callback	Callback to receive the status updates

Returns

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

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

- 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
- IP. Ip address of the device

Example code starting a firmware update

```
OpenFileDialog diag = new OpenFileDialog();

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

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

    RETURN_CODE rt = IDT_Device.SharedController.device_updateFirmwareIP(diag.FileName,0,callback,"
      10.12.34.96");
    if (rt == RETURN_CODE.RETURN_CODE_DO_SUCCESS)
    {

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

Example monitoring firmware update status / success

```
 private void callback( byte[] data, RETURN_CODE transactionResultCode, string IP)
{


            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;


}
```

16.3.2.107 static RETURN_CODE IDTechSDK.IDT_NEO2.device_updateFirmwareType (FIRMWARE_TYPE *type*, byte[] *firmwareData*, string *ip* = "") [static]

Update App Firmware

Updates the firmware

Parameters

type	FIRMWARE_TYPE. It can be Bootloader A, Bootloader B, 1050, K81, or Kernels 0-11.
firmwareData	Signed binary data of a firmware file provided by IDTech
ip	Optional ip address of device

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_updateFirmware(FIRMWARE_TYPE.FIRMWARE_TYPE_1050,
      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;
    }
}
```

16.3.2.108 RETURN_CODE IDTechSDK.IDT_NEO2.device_updateFirmwareType (string *path*, FIRMWARE_TYPE *type*, FirmwareUpdateCallback *callback*, string *ip*)

Update Firmware

Updates the firmware over IP .

Parameters

path	Local filepath to the signed binary data of a firmware file provided by IDTech
type	FIRMWARE_TYPE
callback	Callback to receive the status updates

Returns

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

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

- 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
- IP. Ip address of the device

Example code starting a firmware update

```
OpenFileDialog diag = new OpenFileDialog();

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

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

    RETURN_CODE rt = IDT_Device.SharedController.device_updateFirmwareIP(diag.FileName,0,callback,"
      10.12.34.96");
    if (rt == RETURN_CODE.RETURN_CODE_DO_SUCCESS)
    {

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

Example monitoring firmware update status / success

```
 private void callback( byte[] data, RETURN_CODE transactionResultCode, string IP)
{


            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;


}
```

16.3.2.109 static RETURN_CODE IDTechSDK.IDT_NEO2.device_wakeDevice (string *macAddress* = " ", string *ipAddress* = " ") [static]

Wake Device From Standby Mode

If an IP connected device, the device can be located by it's MAC Address, or IP Address (if the SDK has previously established a connection to that device) When a IP device is first connected, the MacAddress and IP Address are stored in a look up table. Only one of the two parameter are required for Wake on WAN. If IP address is provided, it will take priority over Mac Address.

Parameters

macAddress	Optional: Mac Address of IP connected device to wake
ipAddress	Optional: IP Address of IP connected device to wake

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

16.3.2.110 RETURN_CODE IDTechSDK.IDT_NEO2.emv_activateTransaction (int *timeout*, byte[] *tags*, bool *forceOnline*, bool *isFastEMV* = false)

Start EMV Transaction Request

Authorizes the EMV transaction for an ICC card

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
forceOnline	TRUE = do not allow offline approval, FALSE = allow ICC to approve offline if terminal capable Note: To request tags to be included in default response, use tag DFEE1A, and specify tag list. Example four tags 9F02, 9F36, 95, 9F37 to be included in response = DFEE1A079F029F369f9F37
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 device_getResponseCodeString

16.3.2.111 static void IDTechSDK.IDT_NEO2.emv_allowFallback (bool *allow*) [static]

Allow fallback for EMV transactions. Default is TRUE

Parameters

allow	TRUE = allow fallback, FALSE = don't allow fallback

16.3.2.112 RETURN_CODE IDTechSDK.IDT_NEO2.emv_authenticateTransaction (byte[] *updatedTLV*)

Authenticate EMV Transaction Request

Authenticates the EMV transaction for an ICC card. Execute this after receiving response with result code 0x10 to emv_startTransaction

The tags will be returned in the callback routine.

```
@param updatedTLV  TLV stream that can be used to update the following values:
 - 9F02: Amount
 - 9F03: Other amount
 - 9C: Transaction type
 - 5F57: Account type
In addition tag DFEE1A can be sent to specify tag list to include in results. Example four tags 9F02, 9F36, 95

  @return RETURN_CODE:  Values can be parsed with device_getResponseCodeString
```

16.3.2.113 static void IDTechSDK.IDT_NEO2.emv_autoAuthenticate (bool *authenticate*) [static]

Enables authenticate for EMV transactions. If a startEMVTranaction results in code 0x0010 (start transaction success), then emv_authenticateEMVTransaction can automatically execute if parameter is set to TRUE

Parameters

authenticate	TRUE = auto authenticate, FALSE = manually authenticate

16.3.2.114 static void IDTechSDK.IDT_NEO2.emv_autoAuthenticate (bool *authenticate,* byte[] *tags*) [static]

Enables authenticate for EMV transactions. If a startEMVTranaction results in code 0x0010 (start transaction success), then emv_authenticateEMVTransaction can automatically execute if parameter is set to TRUE

Parameters

authenticate	TRUE = auto authenticate, FALSE = manually authenticate
tags	Tags to pass during authentication stage;

16.3.2.115 RETURN_CODE IDTechSDK.IDT_NEO2.emv_callbackResponseLCD (EMV_LCD_DISPLAY_MODE *type,* byte *selection*)

Callback Response LCD Display

Provides menu selection responses to the kernel after a callback was received with DeviceState.EMVCallback, and callbackType = EMV_CALLBACK_TYPE.EMV_CALLBACK_TYPE_LCD, and lcd_displayMode = EMV_LCD_DI↩SPLAY_MODE_MENU, EMV_LCD_DISPLAY_MODE_PROMPT, or EMV_LCD_DISPLAY_MODE_LANGUAGE↩_SELECT

Parameters

type	If Cancel key pressed during menu selection, then value is EMV_LCD_DISPLAY_MODE_CANCEL. Otherwise, value can be EMV_LCD_DISPLAY_MODE_MENU, EMV_LCD_DISPLAY_MODE_PROMPT, or EMV_LCD_DISPLAY_MODE_LANGUAGE_SELECT
selection	If type = EMV_LCD_DISPLAY_MODE_MENU or EMV_LCD_DISPLAY_MODE_LANGUAGE_SELECT, provide the selection ID line number. Otherwise, if type = EMV_LCD_DISPLAY_MODE_PROMPT supply either 0x43 ('C') for Cancel, or 0x45 ('E') for Enter/accept

Returns

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

16.3.2.116 RETURN_CODE IDTechSDK.IDT_NEO2.emv_callbackResponseMSR (byte[] *MSR*)

Callback Response MSR Entry

Provides MSR information to kernel after a callback was received with DeviceState.EMVCallback, and callbackType = EMV_CALLBACK_MSR

Parameters

MSR	Swiped track data

Returns

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

16.3.2.117 RETURN_CODE IDTechSDK.IDT_NEO2.emv_callbackResponsePIN (EMV_PIN_MODE *type,* byte[] *KSN,* byte[] *PIN*)

Callback Response PIN Entry

Provides (or cancels) PIN entry information to kernel after a callback was received with DeviceState.EMVCallback, and callbackType = EMV_CALLBACK_TYPE_PINPAD

Parameters

type	If Cancel key pressed during PIN entry, then value is EMV_PIN_MODE_CANCEL. Otherwise, value can be EMV_PIN_MODE_ONLINE_DUKPT, EMV_PIN_MODE_ONLINE_MKSK, or EMV_PIN_MODE_OFFLINE
KSN	If enciphered PIN, this is either the PIN DUKPT Key (EMV_PIN_MODE_ONLINE_DUKPT) or PIN Session Key (EMV_PIN_MODE_ONLINE_MKSK), or PIN Pairing DUKPT key (EMV_PIN_MODE_OFFLINE)
PIN	If encipherd PIN, this is encrypted PIN block. If device does not implement pairing function, this is plaintext PIN

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

16.3.2.118 RETURN_CODE IDTechSDK.IDT_NEO2.emv_cancelTransaction ( )

Cancel Transaction

Cancels the currently executing EMV or CTLS transaction.

Returns

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

16.3.2.119 RETURN_CODE IDTechSDK.IDT_NEO2.emv_completeTransaction (bool *commError,* byte[ ] *authCode,* byte[ ] *iad,* byte[ ] *tlvScripts,* byte[ ] *tlv*)

Complete EMV Transaction Request

Completes the EMV transaction for an ICC card when online authorization request is received from emv_↩ authenticateTransaction

The tags will be returned in the callback routine.

Parameters

commError	Communication error with host. Set to TRUE if host was unreachable, or FALSE if host response received. If Communication error, authCode, iad, tlvScripts can be null.
authCode	Authorization code from host. Two bytes. Example 0x3030. (Tag value 8A). Required
iad	Issuer Authentication Data, if any. Example 0x11223344556677883030 (tag value 91).
tlvScripts	71/72 scripts, if any
tlv	Additional TVL data to return with transaction results (if any)

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

NOTE: There are three possible outcomes for Authorization Code: Approval, Refer To Bank, Decline. The kernel maps these three outcomes to valid authorization response codes using tag DFEE1B through 8 bytes: {2 bytes Approval Code}{2 bytes Referral Code}{2 bytes Decline Code}{2 bytes RFU} If your gateway uses "00" for Approval, "01" for Referral, and "05" for Decline, then DFEE1B 08 3030 3031 3035 0000 If you use an authorization code value that that is not defined in DFEE1B, the kernel will use the DECLINE value of DFEE1B by default.

16.3.2.120 RETURN_CODE IDTechSDK.IDT_NEO2.emv_getEMVConfigurationCheckValue (ref string *response*)

Polls device for EMV Configuration Check Value

Parameters

response	Response returned of Check Value of Configuration

Returns

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

16.3.2.121 RETURN_CODE IDTechSDK.IDT_NEO2.emv_getEMVKernelCheckValue (ref string *response*)

Polls device for EMV Kernel Check Value

Parameters

response	Response returned of Check Value of Kernel

Returns

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

16.3.2.122 RETURN_CODE IDTechSDK.IDT_NEO2.emv_getEMVKernelVersion (ref string *response*)

Polls device for EMV Kernel Version

Parameters

response	Response returned of Kernel Version

Returns

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

16.3.2.123 RETURN_CODE IDTechSDK.IDT_NEO2.emv_getTerminalMajorConfiguration (ref int *configuration*)

Get Terminal Major Configuration

Gets the Terminal Data Major Configuration setting

Parameters

configuration	The configuration that is set (1-5)

Return values

RETURN_CODE	Return codes listed as typedef enum in IDTCommon:RETURN_CODE. Values can be parsed with IDT_NEO2::device_getResponseCodeString:()

16.3.2.124 RETURN_CODE IDTechSDK.IDT_NEO2.emv_removeAllApplicationData ( )

Remove All Application Data

Removes all the Application Data for EMV Kernel

Returns

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

16.3.2.125 RETURN_CODE IDTechSDK.IDT_NEO2.emv_removeAllCAPK ( )

Remove All Certificate Authority Public Key

Removes all the CAPK for EMV Kernel

Returns

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

16.3.2.126 RETURN_CODE IDTechSDK.IDT_NEO2.emv_removeAllCRL ( )

Remove All Certificate Revocation List Entries

Removes all CRLEntry entries

Returns

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

16.3.2.127 RETURN_CODE IDTechSDK.IDT_NEO2.emv_removeApplicationData (byte[] *AID*)

Remove Application Data by AID

Removes the Application Data for EMV Kernel 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()

16.3.2.128 RETURN_CODE IDTechSDK.IDT_NEO2.emv_removeCAPK (byte[] *capk*)

Remove Certificate Authority Public Key

Removes the CAPK for EMV Kernel 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()

16.3.2.129 RETURN_CODE IDTechSDK.IDT_NEO2.emv_removeCRL (byte[] *crlList*)

Remove Certificate Revocation List Entries

Removes CRLEntries as specified by the RID and Index and serial number passed as 9 bytes

Parameters

crlList	containing the list of CRL to remove: [CRL1][CRL2]...[CRLn] where each [CRL] is 9 bytes: [5 bytes RID][1 byte CAPK Index][3 bytes serial number]

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

16.3.2.130 RETURN_CODE IDTechSDK.IDT_NEO2.emv_removeTerminalData ( )

Remove Terminal Data

Removes the Terminal Data

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

16.3.2.131 RETURN_CODE IDTechSDK.IDT_NEO2.emv_resetConfigurationGroup (int *group*)

Reset Configuration Group

This command allows resetting a dataset to its default configuration. If the file exists, it will be overwritten. If not, it will be created.

Parameters

group	Configuration Group

Return values

RETURN_CODE	Return codes listed as typedef enum in IDTCommon:RETURN_CODE. Values can be parsed with IDT_VP8800::device_getResponseCodeString:()

16.3.2.132 RETURN_CODE IDTechSDK.IDT_NEO2.emv_retrieveAIDList (ref byte *response[][]*)

Retrieve AID list

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

Parameters

response	array of 2-tag TLV data objects: DFEE2D (group name) followed by 9F06 (AID)

Returns

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

16.3.2.133 RETURN_CODE IDTechSDK.IDT_NEO2.emv_retrieveApplicationData (byte[] *AID,* ref byte[] *tlv*)

Retrieve Application Data by AID

Retrieves the Application Data for EMV Kernel 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()

16.3.2.134 RETURN_CODE IDTechSDK.IDT_NEO2.emv_retrieveCAPK (byte[] *capk,* ref byte[] *key*)

Retrieve Certificate Authority Public Key

Retrieves the CAPK for EMV Kernel 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()

16.3.2.135 RETURN_CODE IDTechSDK.IDT_NEO2.emv_retrieveCAPKList (ref byte[ ] *keys*)

Retrieve the Certificate Authority Public Key list

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

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()

16.3.2.136 RETURN_CODE IDTechSDK.IDT_NEO2.emv_retrieveCRLList (ref byte[ ] *list*)

Retrieve the Certificate Revocation List

Returns the CRL entries on the terminal.

Parameters

list	[CRL1][CRL2]...[CRLn], each CRL 9 bytes where CRL = 5 bytes RID + 1 byte index + 3 bytes serial number

Returns

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

16.3.2.137 RETURN_CODE IDTechSDK.IDT_NEO2.emv_retrieveTerminalData (ref byte[ ] *tlv*)

Retrieve Terminal Data

Retrieves the Terminal Data for EMV Kernel.

Parameters

tlv	Response returned as a TLV

Returns

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

16.3.2.138 RETURN_CODE IDTechSDK.IDT_NEO2.emv_retrieveTransactionResult (byte[ ] *tags,* ref **IDTTransactionData** *tlv*)

Retrieve Transaction Results

Retrieves specified EMV tags from the currently executing transaction.

Parameters

tags	Tags to be retrieved. Example 0x9F028A will retrieve tags 9F02 and 8A
tlv	All requested tags returned as unencrypted, encrypted and masked TLV data in IDDTransactionData object

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

16.3.2.139 RETURN_CODE IDTechSDK.IDT_NEO2.emv_setApplicationData (byte[ ] *name,* byte[ ] *tlv*)

Set Application Data by AID

Sets the Application Data as specified by TLV data

Parameters

tlv	Application data in TLV format

Tags:

- DFEE4B : Partial AID Matching. 01 = allowed, 00 = Disabled. Mandatory for Visa
- DFEE4C : Application Flow, System: Never, User: Mandatory – 0x01 = MasterCard – 0x02 = AMEX – 0x03 = MasterCard w/Strip Application – 0x06 = Visa – 0x0D = Discover – 0x0E = JCB – 0x15 = Reserved – 0x16 = Reserved – 0x17 = Reserved
- DFEE4D = PPSE Disable, Optional
- DFEE2E = Max AID Length, Mandatory if DFEE4B included. Visa must be set to 16
- DFEE2F = Disable System Aid (no effect on User AID). 0x80 = Disable, 0x00 = Enable
- DFEE4F = Interface Support. 01 = CTLS, 02 = Contact. If missing, defaults to CTLS

Returns

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

16.3.2.140 RETURN_CODE IDTechSDK.IDT_NEO2.emv_setApplicationData (byte[ ] *tlv*)

Set Application Data by AID

Sets the Application Data as specified by TLV data

Parameters

tlv	Application data in TLV format The first tag of the TLV data must be the group number (DFEE2D). The second tag of the TLV data must be the AID (9F06) Tag DFEE4F must be included, and it must have a value of 0x02 to be and EMV AID

Example valid TLV, for Group #8, AID a0000000045010: "dfee2d01089f0607a0000000045010dfee4b0101dfee2e0110dfee4c0101dfee

Tags:

- DFEE2D : Group Number. Mandatory First Tag. 0 = System, 1-255 = User
- 9F06 : AID. Mandatory Second Tag
- DFEE4B : Partial AID Matching. 01 = allowed, 00 = Disabled. Mandatory for Visa
- DFEE4C : Application Flow, System: Never, User: Mandatory – 0x01 = MasterCard – 0x02 = AMEX – 0x03 = MasterCard w/Strip Application – 0x06 = Visa – 0x0D = Discover – 0x0E = JCB – 0x15 = Reserved – 0x16 = Reserved – 0x17 = Reserved
- DFEE4D = PPSE Disable, Optional
- DFEE2E = Max AID Length, Mandatory if DFEE4B included. Visa must be set to 16
- DFEE2F = Disable System Aid (no effect on User AID). 0x80 = Disable, 0x00 = Enable
- DFEE4F = Interface Support. 01 = CTLS, 02 = Contact. If missing, defaults to CTLS

Returns

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

16.3.2.141 RETURN_CODE IDTechSDK.IDT_NEO2.emv_setCAPK (byte[ ] *key*)

Set Certificate Authority Public Key

Sets the CAPK for EMV Kernel 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()

16.3.2.142 RETURN_CODE IDTechSDK.IDT_NEO2.emv_setCRL (byte[ ] *list*)

Set Certificate Revocation List

Sets the CRL

Parameters

list	CRL Entries containing the RID, Index, and serial numbers to set [CRL1][CRL2]...[CRLn] where each [CRL] is 9 bytes: [5 bytes RID][1 byte CAPK Index][3 bytes serial number]

Returns

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

16.3.2.143 RETURN_CODE IDTechSDK.IDT_NEO2.emv_setTerminalData (byte[ ] *tlv*)

Set Terminal Data

Sets the Terminal Data for EMV Kernel

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_NEO2::device_getResponseCodeString:()

16.3.2.144 RETURN_CODE IDTechSDK.IDT_NEO2.emv_setTerminalMajorConfiguration (int *configuration*)

Set Terminal Major Configuration

Sets the Terminal Data Major Configuration setting

Parameters

configuration	The configuration to set (1-5)

Return values

RETURN_CODE	Return codes listed as typedef enum in IDTCommon:RETURN_CODE. Values can be parsed with IDT_NEO2::device_getResponseCodeString:()

16.3.2.145 RETURN_CODE IDTechSDK.IDT_NEO2.emv_startTransaction (double *amount,* double *amtOther,* int *exponent,* int *type,* int *timeout,* byte[] *tags,* bool *forceOnline,* bool *isFastEMV* = `false`)

Start EMV Transaction Request

Authorizes the EMV transaction for an ICC card

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)
exponent	Number of characters after decimile point
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.
forceOnline	TRUE = do not allow offline approval, FALSE = allow ICC to approve offline if terminal capable Note: To request tags to be included in default response, use tag DFEE1A, and specify tag list. Example four tags 9F02, 9F36, 95, 9F37 to be included in response = DFEE1A079F029F369f9F37
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 device_getResponseCodeString

16.3.2.146 RETURN_CODE IDTechSDK.IDT_NEO2.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()

16.3.2.147 RETURN_CODE IDTechSDK.IDT_NEO2.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()

16.3.2.148 RETURN_CODE IDTechSDK.IDT_NEO2.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()

16.3.2.149 RETURN_CODE IDTechSDK.IDT_NEO2.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()

16.3.2.150 RETURN_CODE IDTechSDK.IDT_NEO2.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()

16.3.2.151 RETURN_CODE IDTechSDK.IDT_NEO2.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

Parameters

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()

16.3.2.152 RETURN_CODE IDTechSDK.IDT_NEO2.getCashTranRiskPara (ref byte[] *tlv*)

Get Cash Transaction Reader Risk Parameters Returns the TTQ and reader risk parameters that will be used for cash transactions, if enabled.

Parameters

tlv	TLV Data Objects

Returns

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

16.3.2.153 static int IDTechSDK.IDT_NEO2.getCommandTimeout ( ) [static]

Get Command Timeout

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

Return values

time	Time

16.3.2.154 RETURN_CODE IDTechSDK.IDT_NEO2.getDrlReaderRiskPara (byte *index,* ref byte[] *tlv*)

Get DRL Reader Risk Parameters Get the Index, Application Program ID, and reader risk parameters for the DRL settings.

Parameters

index	DRL index (01-04)
tlv	TLV Data Objects

Returns

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

16.3.2.155 RETURN_CODE IDTechSDK.IDT_NEO2.getHardwareInfor (ref string *ascii*)

Get Hardware Information

Parameters

ascii	the ascii charactor **RETURN_CODE: Values can be parsed with errorCode.getErrorString()**

ASCII | Description

HW,VPVendi<CR><LF>K21F Rev9 | Vendi HW,VP3300 Audio Jack<CR><LF>K21F Rev9 | Unipay III HW,V↩PUnipay1.5<CR><LF>K21F Rev9 | Unipay 1.5 HW,VPUniPay1.5TTK<CR><LF>K21F Rev9 | UniPay 1.5 TTK HW,VP3300 USB<CR><LF>K21F Rev9 | VP3300 USB, VP3300 USB OEM (iBase/Cake same code) HW,V↩P3300 USB-E<CR><LF>K21F Rev9 | VP3300 USB-E HW,VP3300 USB-C<CR><LF>K21F Rev9 | VP3300 USB-C HW,VPVP3300 Bluetooth<CR><LF>K21F Rev9 | VP3300 Bluetooth HW,.VP6300<CR><LF>K81F.↩Rev4 | VP6300

16.3.2.156 static string IDTechSDK.IDT_NEO2.getlastErrorString (string *ip* = " ") [static]

Last Error String

Returns the last firmware reported error string to RETURN_CODE_P2_FAILED and RETURN_CODE_P2_COM↩MAND_NOT_ALLOWED if error reporting is enabled

Parameters

ip	IP address of device

Return values

string	Last Error String Message

16.3.2.157 RETURN_CODE IDTechSDK.IDT_NEO2.getModuleVer (ref string *moduleVer*)

Get Module Version Information Get the 16 byte UID of MCU

Parameters

uid	string UID

Returns

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

16.3.2.158 RETURN_CODE IDTechSDK.IDT_NEO2.getMsrSecurePar (bool *b0,* bool *b1,* bool *b2,* bool *b3,* ref byte[ ] *tlv*)

Get MSR Secure Parameters get parameters from flash setting

Parameters

b0,b1,b2,b3	Encryption Option Bit 0: T1 force encrypt Bit 1: T2 force encrypt Bit 2: T3 force encrypt Bit 3: T3 force encrypt when card type is 80
tlv	MSR Secure Parameters TVL objects

Returns

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

16.3.2.159 RETURN_CODE IDTechSDK.IDT_NEO2.getRemoteKeyInjectionTO (ref int *timeout*)

Get Remote Key Injection Timeout

Parameters

timeout	Timeout is in seconds, value scope is [120, 3600]

Returns

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

16.3.2.160 RETURN_CODE IDTechSDK.IDT_NEO2.getUIDofMCU (ref string *uid*)

Get UID of MCU

Parameters

moduleVer	module version information

Returns

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

16.3.2.161 RETURN_CODE IDTechSDK.IDT_NEO2.getUsbBootLoader (ref string *bootLoader*)

Get USB Boot Loader Version Get the version of the USB Boot Loader

Parameters

bootLoader	USB boot loader information

Returns

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

16.3.2.162 RETURN_CODE IDTechSDK.IDT_NEO2.getWhiteList (ref byte[] *list*)

Get White List Retrieve the White List

Parameters

list	the white list

Returns

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

16.3.2.163 RETURN_CODE IDTechSDK.IDT_NEO2.icc_exchangeAPDU (string *c_APDU,* ref byte[] *response*)

Exchange APDU

Sends an APDU packet to the ICC. If successful, response is returned in APDUResult class instance in response parameter.

Parameters

c_APDU	APDU data packet
response	Unencrypted/encrypted parsed APDU response

Returns

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

16.3.2.164 RETURN_CODE IDTechSDK.IDT_NEO2.icc_getICCReaderStatus (ref byte *status*)

Get Reader Status

Returns the reader status

Parameters

status	Pointer that will return with the ICCReaderStatus results. bit 0: 0 = ICC Power Not Ready, 1 = ICC Powered bit 1: 0 = Card not seated, 1 = card seated

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

16.3.2.165 RETURN_CODE IDTechSDK.IDT_NEO2.icc_powerOffICC ( )

Power Off ICC

Powers down the ICC

Returns

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

If Success, empty If Failure, ASCII encoded data of error string

16.3.2.166 RETURN_CODE IDTechSDK.IDT_NEO2.icc_powerOnICC (ref byte[] *ATR,* byte *interfaces*)

Power On ICC

Power up the currently selected microprocessor card in the ICC reader

Parameters

interfaces	For NEO2 devices allowed interfaces for which to get the ATR. 0x20h = retrieve last ATR received from PICC 0x21h = SAM1 (SRED version only) 0x22h = SAM2 (SRED version only) For other devices interfaces euquals to 0s

Returns

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

16.3.2.167 void IDTechSDK.IDT_NEO2.ip_autoConnectToSocket ( )

Auto Connect To Socket

Instructs SDK to attempt to attempt to connect to all the IP addresses as specified in NEO2_Devices.xml. The NEO2_Devices.xml must be located in the same root directory as IDTechSDK.dll. It is used to specify NEO2 devices for USB connectivity and IP connectivity. It is made up of DEVICE entries example as follows:

```
<Device>
   <Product>VP6800</Product>
   <Name>VP6800</Name>
   <VID>0x0acd</VID>
   <PID>0x4460</PID>
   <UsageID>0x0001</UsageID>
   <UsagePage>0xFF00</UsagePage>
   <BaseIP>10.12.34.98</BaseIP>
   <IP_Count>2</IP_Count>
   <IP_Port></IP_Port>
</Device>
```

- BaseIP = IP address of first device to look for
- IP_Count - # of consecutive IP address to search, up to an IP address of xx.xx.xx.255
- IP_Port - Optional. Defaults to port 1025

Multiple device connections can be defined by a single Device entry, if those devices will use consecutive IP addresses, or multiple device connections can be defined by multiple Device entries, each with unique IP.

Every IP connection attempt is reported to a callback as connection successful, or connection could not be established

16.3.2.168 bool IDTechSDK.IDT_NEO2.ip_connectToSocket (string *IP*, bool *isSecure* = `false`)

Connect to Socket

Instructs SDK to attempt to use TCP/IP for communication with IDT_NEO2

Parameters

IP	Valid IP address, with optional port and label. Examples • IP Address: "10.12.34.98" • IP Address + Port: "10.12.34.98:1024" • IP Address + Label: "10.12.34.98#Neo Device #1" • IP Address + Port + Label: "10.12.34.98:1025#Neo Device #1"
isSecure	TRUE = Use TLS 1.2

Return values

success	TRUE = device found, FALSE = device not found

NOTE: If the device is found, it will automatically become the currently selected SDK device unless there is another currently connected IP device the SDK is using. If you would like to change between multiple IP connected devices, use ip_switchToSocket instead

16.3.2.169 static List<string> IDTechSDK.IDT_NEO2.ip_getSocketList ( ) [static]

Get IP Socket List

Returns a list of all the established socket connections with the SDK. Each entry is the original full IP address specified.

Return values

list	Socket Connections

16.3.2.170 static bool IDTechSDK.IDT_NEO2.ip_isConnected (string *ip,* int *attempts* = 1, bool *isSecure* = false) [static]

Is IP Connected

Validates if IP channel is open and device responsive. If the channel is open and device not responsive, this will close the channel.

Parameters

IP	address to check
attempts	Number of connection attempts to try before returning result
isSecure	TRUE = TLS 1.2 connection

Return values

status	• TRUE = channel is open and device is responsive • FALSE = channel is open and device is unresponsive (channel will close) • FALSE = channel is closed

16.3.2.171 void IDTechSDK.IDT_NEO2.ip_monitorSocketConnectionStatus (bool *enable,* bool *monitorConnect,* int *interval,* int *retryCount*)

Monitor Socket Connection Status

Instructs SDK to monitor currently connected devices for network disconnect, and also to automatically re-connect when plugged back into the network. Disconnect monitors is done by pinging the SDK connected stream list on an interval, and if ping unsuccessful, that device stream is closed and removed from SDK device list, followed by a notification of the device disconnect.

If the option to monitor connection, then the NEO2_Devices.xml must contain the auto-connect information for the

SDK to run on the specified interval.

Parameters

enable	TRUE = enable polling for device disconnect and optional connect, at specified interval and retryCount
monitorConnect	If TRUE, an auto-connect attempt will be made at the specified interval
interval	Interval, in seconds, to monitor IP. Minimum value 5 seconds
retryCount	Number of retries trying to communicate before determining device not available and issue disconnect notification.

16.3.2.172 static bool IDTechSDK.IDT_NEO2.ip_switchToSocket (string *IP*) [static]

Switch to Socket

Instructs SDK to direct communication to the specified device (the device must have already established a connection through auto-connent or ip_connecToSocket)

Parameters

IP	Valid established IP address of the existing device. Must match original IP string exactly. Example: Connect to device as 192.168.1.155:50#Device_1. You must use that whole string, not just 192.168.1.155, or 192.168.1.155:50.

16.3.2.173 RETURN_CODE IDTechSDK.IDT_NEO2.lcd_addButton (string *screenName,* string *buttonName,* byte *type,* byte *alignment,* UInt16 *xCord,* UInt16 *yCord,* string *label,* ref lcdItem *returnItem,* buttonCallback *callback,* string *ip* = " ")

Add Button

Adds a button to a selected screen. Must execute lcd_createScreen first to establish a screen to draw on.

Parameters

screenName	Screen name that will be the target of add button
buttonName	Button name that will be the target of add button
type	Button Type • Large = 0x01 • Medium = 0x02 • Invisible = 0x03 (70px by 60 px)
alignment	Position for Button • 0 = Display object at the horizon center of specified y, while x ignored • 1 = Display object at specified x andy • 2 = Display object at center of screen, x, y are both ignored • 3 = Display object at left of the screen of specified y, while x ignored • 4 = Display object at right of the screen of specified y, while x ignored
xCord	x-coordinate for Button, range 0-271
yCord	y-coordinate for Button, range 0-479

Parameters

label	Label to show on the button. Required for Large/Medium buttons. Not used for Invisible buttons.
returnItem	The item includes screen ID, object ID, top-left x-coordinate, top-left y-coordinate, bottom-right x-coordinate, bottom-left y-coordinate, which are all assigned to the created button
callback	buttonCallback to receive button presses. Passing NULL will return button presses to main messageCallback.
ip	Optional: IP address to execute command on (for IP connected devices)

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

NOTE ON BUTTON PRESS EVENTS: A button press consists of uInt16 specifying the screen, and uInt16 specifying the button ID If buttonCallback is used, it will have the follow signature:
public delegate void buttonCallback(IDT_DEVICE_Types sender, UInt16 screenID, UInt16 itemID, string IP);
If buttonCallback is NULL, then the button presses will be passed to the messageCallback, with DeviceState.↩ ButtonEvent, and data = data[0], data[1] = uInt16 screenID, and data[2], data[3] = uint16 buttonID

16.3.2.174 RETURN_CODE IDTechSDK.IDT_NEO2.lcd_addEthernet (string *screenName,* string *objectName,* byte *alignment,* UInt16 *xCord,* UInt16 *yCord,* ref lcdItem *returnItem,* string *ip*)

Add Ethernet Settings

Adds an Ethernet settings to a selected screen. Must execute lcd_createScreen first to establish a screen to draw on.

Parameters

screenName	Screen name that will be the target of add widget
objectName	Object name that will be the target of add widget
alignment	Position for widget • 0 = Display object at the horizon center of specified y, while x ignored • 1 = Display object at specified x andy • 2 = Display object at center of screen, x, y are both ignored • 3 = Display object at left of the screen of specified y, while x ignored • 4 = Display object at right of the screen of specified y, while x ignored
xCord	x-coordinate for widget, range 0-271
yCord	y-coordinate for widget, range 0-479
returnItem	The item includes screen ID, object ID, top-left x-coordinate, top-left y-coordinate, bottom-right x-coordinate, bottom-left y-coordinate, which are all assigned to the created widget
ip	Optional: IP address to execute command on (for IP connected devices)

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

Item	Maximum can be created for each screen
Text Area	20
Large Button	8

Item	Maximum can be created for each screen
Medium Button	16
Invisible Button	16
Numeric Entry	1
Ethernet Setting	1
Led widget	1
image	20

16.3.2.175 RETURN_CODE IDTechSDK.IDT_NEO2.lcd_addImage (string *screenName,* string *objectName,* byte *alignment,* UInt16 *xCord,* UInt16 *yCord,* string *filename,* ref lcdItem *returnItem,* string *ip =* " ")

Add Image

Adds a image to a selected screen. Must execute lcd_createScreen first to establish a screen to draw on.

Parameters

screenName	Screen name that will be the target of add image
objectName	Object name that will be the target of add image
alignment	Position for Image • 0 = Display object at the horizon center of specified y, while x ignored • 1 = Display object at specified x andy • 2 = Display object at center of screen, x, y are both ignored • 3 = Display object at left of the screen of specified y, while x ignored • 4 = Display object at right of the screen of specified y, while x ignored
xCord	x-coordinate for Image, range 0-271
yCord	y-coordinate for Image, range 0-479
filename	Filename of the image. Must be available in device memory.
returnItem	The item includes screen ID, object ID, top-left x-coordinate, top-left y-coordinate, bottom-right x-coordinate, bottom-left y-coordinate, which are all assigned to the created image
ip	Optional: IP address to execute command on (for IP connected devices)

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

Item	Maximum can be created for each screen
Text Area	20
Large Button	8
Medium Button	16
Invisible Button	16
Numeric Entry	1
Ethernet Setting	1
Led widget	1
image	20

16.3.2.176 RETURN_CODE IDTechSDK.IDT_NEO2.lcd_addLED (string *screenName,* string *objectName,* byte *alignment,* UInt16 *xCord,* UInt16 *yCord,* ref lcdItem *returnItem,* byte[ ] *LED,* string *ip =* " ")

Add LED

Adds a LED widget to a selected screen. Must execute lcd_createScreen first to establish a screen to draw on.

Parameters

screenName	Screen name that will be the target of add LED
objectName	Object name that will be the target of add LED
alignment	Position for LED • 0 = Display object at the horizon center of specified y, while x ignored • 1 = Display object at specified x andy • 2 = Display object at center of screen, x, y are both ignored • 3 = Display object at left of the screen of specified y, while x ignored • 4 = Display object at right of the screen of specified y, while x ignored
xCord	x-coordinate for LED, range 0-271
yCord	y-coordinate for LED, range 0-479
returnItem	The item includes screen ID, object ID, top-left x-coordinate, top-left y-coordinate, bottom-right x-coordinate, bottom-left y-coordinate, which are all assigned to the created widget
LED	Must be 4 bytes, LED 0 = byte 0, LED 1 = byte 1, LED 2 = byte 2, LED 3 = byte 3 • Value 0 = LED OFF • Value 1 = LED Green • Value 2 = LED Yellow • Value 3 = LED Red
ip	Optional: IP address to execute command on (for IP connected devices)

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

Item	Maximum can be created for each screen
Text Area	20
Large Button	8
Medium Button	16
Invisible Button	16
Numeric Entry	1
Ethernet Setting	1
Led widget	1
image	20

16.3.2.177 RETURN_CODE IDTechSDK.IDT_NEO2.lcd_addText (string *screenName,* string *objectName,* byte *alignment,* UInt16 *xCord,* UInt16 *yCord,* UInt16 *width,* UInt16 *height,* byte *fontID,* byte[ ] *color,* string *label,* ref lcdItem *returnItem,* string *ip =* " ")

Add text

Adds a text component to a selected screen. Must execute lcd_createScreen first to establish a screen to draw on.

Parameters

screenName	Screen name that will be the target of add text
objectName	Object name that will be the target of add text
alignment	Position for Text • 0 = Display object at the horizon center of specified y, while x ignored • 1 = Display object at specified x andy • 2 = Display object at center of screen, x, y are both ignored • 3 = Display object at left of the screen of specified y, while x ignored • 4 = Display object at right of the screen of specified y, while x ignored
xCord	x-coordinate for Text, range 0-271
yCord	y-coordinate for Text, range 0-479
width	Width of text area
height	Height of text area
fontID	Font ID
color	Four bytes for color, example, Blue = 0xFF000000, Black = 0x00000000 • Byte 0 = B • Byte 1 = G • Byte 2 = R • Byte 3 = Reserved
label	Label to show on the text
returnItem	The item includes screen ID, object ID, top-left x-coordinate, top-left y-coordinate, bottom-right x-coordinate, bottom-left y-coordinate, which are all assigned to the created text area
ip	Optional: IP address to execute command on (for IP connected devices)

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

Font ID	Typography Name	Font	Size
0	RoundBold_12	RoundBold.ttf	12
1	RoundBold_18	RoundBold.ttf	18
2	RoundBold_24	RoundBold.ttf	24
3	RoundBold_36	RoundBold.ttf	36
4	RoundBold_48	RoundBold.ttf	48
5	RoundBold_60	RoundBold.ttf	60
6	RoundBold_72	RoundBold.ttf	72
7	RoundCondensedBold_12	RoundCondensedBold.ttf	12
8	RoundCondensedBold_18	RoundCondensedBold.ttf	18
9	RoundCondensedBold_24	RoundCondensedBold.ttf	24
10	RoundCondensedBold_36	RoundCondensedBold.ttf	36
11	RoundCondensedBold_48	RoundCondensedBold.ttf	48
12	RoundCondensedBold_60	RoundCondensedBold.ttf	60
13	RoundCondensedBold_72	RoundCondensedBold.ttf	72

Font ID	Typography Name	Font	Size
14	RoundCondensedMedium_12	RoundCondensedMedium_↩ 0.ttf	12
15	RoundCondensedMedium_18	RoundCondensedMedium_↩ 0.ttf	18
16	RoundCondensedMedium_24	RoundCondensedMedium_↩ 0.ttf	24
17	RoundCondensedMedium_36	RoundCondensedMedium_↩ 0.ttf	36
18	RoundCondensedMedium_48	RoundCondensedMedium_↩ 0.ttf	48
19	RoundCondensedMedium_60	RoundCondensedMedium_↩ 0.ttf	60
20	RoundCondensedMedium_72	RoundCondensedMedium_↩ 0.ttf	72
21	RoundCondensedSemibold_12	RoundCondensedSemibold.ttf	12
22	RoundCondensedSemibold_18	RoundCondensedSemibold.ttf	18
23	RoundCondensedSemibold_24	RoundCondensedSemibold.ttf	24
24	RoundCondensedSemibold_36	RoundCondensedSemibold.ttf	36
25	RoundCondensedSemibold_48	RoundCondensedSemibold.ttf	48
26	RoundCondensedSemibold_60	RoundCondensedSemibold.ttf	60
27	RoundCondensedSemibold_72	RoundCondensedSemibold.ttf	72
28	RoundMedium_12	RoundMedium.ttf	12
29	RoundMedium_18	RoundMedium.ttf	18
30	RoundMedium_24	RoundMedium.ttf	24
31	RoundMedium_36	RoundMedium.ttf	36
32	RoundMedium_48	RoundMedium.ttf	48
33	RoundMedium_60	RoundMedium.ttf	60
34	RoundMedium_72	RoundMedium.ttf	72
35	RoundSemibold_12	RoundSemibold.ttf	12
36	RoundSemibold_18	RoundSemibold.ttf	18
37	RoundSemibold_24	RoundSemibold.ttf	24
38	RoundSemibold_36	RoundSemibold.ttf	36
39	RoundSemibold_48	RoundSemibold.ttf	48
40	RoundSemibold_60	RoundSemibold.ttf	60
41	RoundSemibold_72	RoundSemibold.ttf	72

Item	Maximum can be created for each screen
Text Area	20
Large Button	8
Medium Button	16
Invisible Button	16
Numeric Entry	1
Ethernet Setting	1
Led widget	1
image	20

16.3.2.178 RETURN_CODE IDTechSDK.IDT_NEO2.lcd_clearAllLines ( )

Clear LCD Display

Clears all lines of the LCD Display.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

16.3.2.179 static RETURN_CODE IDTechSDK.IDT_NEO2.lcd_clearDisplay ( ) [static]

Clear Display

Command to clear the display screen on the reader.It returns the display to the currently defined background color and terminates all events

Returns

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

16.3.2.180 RETURN_CODE IDTechSDK.IDT_NEO2.lcd_clearScreenInfo (string *ip* = " ")

Clear Screen Info

Clear all current screen information in RAM and flash. And then show'power-on screen'

Parameters

ip	Optional: IP address to execute command on (for IP connected devices)

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

16.3.2.181 RETURN_CODE IDTechSDK.IDT_NEO2.lcd_cloneScreen (string *screenName,* string *cloneName,* ref UInt16 *cloneID,* string *ip* = " ")

Clone Screen

Clones an existing screen.

Parameters

screenID	Screen number to clone.
cloneID	Screen ID of the cloned screen
ip	Optional: IP address to execute command on (for IP connected devices)

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

16.3.2.182 RETURN_CODE IDTechSDK.IDT_NEO2.lcd_createScreen (string *screenName,* ref UInt16 *screenID,* string *ip* = " ")

Create Screen

Creates a new screen.

Parameters

screenName	Screen name to use.
screenID	Screen ID that was created.
ip	Optional: IP address to execute command on (for IP connected devices)

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

16.3.2.183 RETURN_CODE IDTechSDK.IDT_NEO2.lcd_destroyScreen (string *screenName,* string *ip =* " ")

Destroy Screen

Destroys a previously created inactive screen. The screen cannot be active

Parameters

screenName	Screen name to destroy. The screen number is assigned with lcd_createScreen.
ip	Optional: IP address to execute command on (for IP connected devices)

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

16.3.2.184 RETURN_CODE IDTechSDK.IDT_NEO2.lcd_displayMessage (int *lineNumber,* string *message*)

Display Message on Line

Displays a message on a display line. 16 characters per line. You can display up to 32 characters for line 1, and it will flow to line 2, or you can display up to 16 characters on just line 2

Parameters

lineNumber	Line number to display message on (1 or 2)
message	Message to display

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

16.3.2.185 RETURN_CODE IDTechSDK.IDT_NEO2.lcd_getActiveScreen (ref string *screenName,* string *ip =* " ")

Get Active Screen

Returns the active screen ID.

Parameters

screenName	Screen name this is active.
ip	Optional: IP address to execute command on (for IP connected devices)

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

16.3.2.186 RETURN_CODE IDTechSDK.IDT_NEO2.lcd_getAllObjects (string *screenName,* ref byte *objectNumbers,* ref Dictionary< UInt16, string > *returnObjects,* string *ip* = " ")

Get All Objects on Screen

Get all created objects' name on certain screen

Parameters

objectNumbers	Number of created objects
returnObjects	Dictionary all created objects -key ID of a created screen -value Name of a created screen
ip	Optional: IP address to execute command on (for IP connected devices)

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

16.3.2.187 RETURN_CODE IDTechSDK.IDT_NEO2.lcd_getAllScreens (ref byte *screenNumbers,* ref Dictionary< UInt16, string > *returnScreens,* string *ip* = " ")

Get All Screens

Get all created screens' name

Parameters

screenNumbers	Number of created screens
returnScreens	Dictionary all created screens -key ID of a created screen -value Name of a created screen
ip	Optional: IP address to execute command on (for IP connected devices)

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

16.3.2.188 RETURN_CODE IDTechSDK.IDT_NEO2.lcd_getButtonEvent (ref UInt16 *screenID,* ref UInt16 *objectID,* ref string *screenName,* ref string *objectName,* ref bool *isLongPress,* string *ip* = " ")

Get Button Event

Reports back the ID of the button that encountered a click event after the last Get Button Event.

Parameters

screenID	Screen ID of the last clicked button
objectID	Button ID of the last clicked button
screenName	Screen Name of the last clicked button
objectName	Button Name of the last clicked button
isLongPress	TRUE = Long Press, FALSE = Short Press
ip	Optional: IP address to execute command on (for IP connected devices)

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

16.3.2.189 RETURN_CODE IDTechSDK.IDT_NEO2.lcd_loadScreenInfo (string *ip* = " ")

Load Screen Info

Load all current screen information from RAM to flash

Parameters

ip	Optional: IP address to execute command on (for IP connected devices)

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

16.3.2.190 RETURN_CODE IDTechSDK.IDT_NEO2.lcd_playAudio (string *name,* int *type,* string *ip* = " ")

Play Audio

This command plays an audio file loaded from the inserted SD card. The VP6800 supports 16bit PCM format .WAV files.

Parameters

name	Name of file on SD Card
type	0 = Flash, 1 = SD Card
ip	Optional IP Address

Returns

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

16.3.2.191 RETURN_CODE IDTechSDK.IDT_NEO2.lcd_queryObjectbyID (UInt16 *objectID,* ref byte *objectNumbers,* ref List< string > *returnItems,* string *ip* = " ")

Queery Object by ID

Check if the given object exists or not. If exists, return all screen names which contains the object of the given object ID

Parameters

objectID	ID of the checked object
objectNumber	Number of the checked object
returnItems	screens containing the item
ip	Optional: IP address to execute command on (for IP connected devices)

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

16.3.2.192 RETURN_CODE IDTechSDK.IDT_NEO2.lcd_queryObjectbyName (string *objectName,* ref byte *objectNumbers,* ref List< string > *returnItems,* string *ip =* " ")

Queery Object by Name

Check if the given object exists or not. If exists, return all screen names which contains the object of the given object name

Parameters

objectName	Name of the checked object
objectNumber	Number of the checked object
returnItems	List of all the screens that contain the object
ip	Optional: IP address to execute command on (for IP connected devices)

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

16.3.2.193 RETURN_CODE IDTechSDK.IDT_NEO2.lcd_queryScreenbyID (UInt16 *screenID,* ref byte *result,* ref string *screenName,* string *ip =* " ")

Queery Screen by ID

Check if the given screen exists or not

Parameters

screenID	ID of the checked screen
result	the checking result
screenName	Name of the checked screen
ip	Optional: IP address to execute command on (for IP connected devices)

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

16.3.2.194 RETURN_CODE IDTechSDK.IDT_NEO2.lcd_queryScreenbyName (string *screenName,* ref byte *result,* string *ip =* " ")

Queery Screen by Name

Check if the given screen exists or not

Parameters

screenName	Name of the checked screen
result	the checking result
ip	Optional: IP address to execute command on (for IP connected devices)

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

16.3.2.195 RETURN_CODE IDTechSDK.IDT_NEO2.lcd_removeItem (string *screenName,* string *objectName,* string *ip =* " ")

Removed Item

Removes a component.

Parameters

screenName	Screen name where to remove the target from.
objectName	Identifier of the component
ip	Optional: IP address to execute command on (for IP connected devices)

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

16.3.2.196 static void IDTechSDK.IDT_NEO2.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

16.3.2.197 RETURN_CODE IDTechSDK.IDT_NEO2.lcd_setBacklight (byte *backlightVal,* string *ip =* " ")

Set Backlight

Set backlight percentage. If the percent $>$100, it will be rejected. If percent $<$ 10, backlight percent will be set to 10

Parameters

backlightVal	Backlight percent value to be sat
ip	Optional: IP address to execute command on (for IP connected devices)

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

16.3.2.198 void IDTechSDK.IDT_NEO2.lcd_setButtonCallback (string *screenName,* string *buttonName,* buttonCallback *callback,* string *ip*)

Set Button Callback

Sets the callback for a button.

Parameters

screenName	Screen name where the button is located
buttonName	Object name of the button
callback	buttonCallback to receive button presses. Passing NULL will return button presses to main messageCallback.
ip	Optional: IP address to execute command on (for IP connected devices)

NOTE ON BUTTON PRESS EVENTS: A button press consists of uInt16 specifying the screen, and uInt16 specifying the button ID If buttonnCallback is used, it will have the follow signature:
public delegate void buttonCallback(IDT_DEVICE_Types sender, UInt16 screenID, UInt16 itemID, string IP);
If buttonCallback is NULL, then the button presses will be passed to the messageCallback, with DeviceState.↩
ButtonEvent, and data = data[0], data[1] = uInt16 screenID, and data[2], data[3] = uint16 buttonID

16.3.2.199 void IDTechSDK.IDT_NEO2.lcd_setPinCancelPromptCallback (CancelPromptCallback *callback,* string *ip* = " ")

Set Pin Cancel Prompt Callback

Sets the callback for the PIN Cancel Prompt Callback.

Parameters

callback	CancelPromptCallback
ip	Optional: IP address to execute command on (for IP connected devices)

Callback will have the follow signature:
public delegate void CancelPromptCallback(string ipAddress);

16.3.2.200 void IDTechSDK.IDT_NEO2.lcd_setPinFailureCallback (FailureCallback *callback,* string *ip* = " ")

Set Pin Failure Callback

Sets the callback for the PIN Failure Callback.

Parameters

callback	FailureCallback
ip	Optional: IP address to execute command on (for IP connected devices)

Callback will have the follow signature:
public delegate void FailureCallback(string ipAddress, RETURN_CODE errorCode);

16.3.2.201 void IDTechSDK.IDT_NEO2.lcd_setPinInputCallback (SwipeCallback *callback,* string *ip* = " ")

Set Pin Input Callback

Sets the callback for the PIN Input Callback.

Parameters

callback	SwipeCallback
ip	Optional: IP address to execute command on (for IP connected devices)

Callback will have the follow signature:
public delegate void SwipeCallback(string ipAddress, string input);

16.3.2.202 void IDTechSDK.IDT_NEO2.lcd_setPinSwipeCallback (SwipeCallback *callback,* string *ip* = " ")

Set Pin Swipe Callback

Sets the callback for the PIN Swipe Callback.

Parameters

callback	SwipeCallback
ip	Optional: IP address to execute command on (for IP connected devices)

Callback will have the follow signature:
public delegate void InputCallback(string ipAddress, IDTTransactionData input);

16.3.2.203 void IDTechSDK.IDT_NEO2.lcd_setPinTimeoutCallback (TimeoutCallback *callback,* string *ip* = " ")

Set Pin Timeout Callback

Sets the callback for the PIN Timeout Callback.

Parameters

callback	TimeoutCallback
ip	Optional: IP address to execute command on (for IP connected devices)

Callback will have the follow signature:
public delegate void TimeoutCallback(string ipAddress);

16.3.2.204 RETURN_CODE IDTechSDK.IDT_NEO2.lcd_showScreen (string *screenName,* string *ip* = " ")

Show Screen

Displays and makes active a previously created screen.

Parameters

screenName	Screen to display. The screen name is defined with lcd_createScreen.
ip	Optional: IP address to execute command on (for IP connected devices)

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

16.3.2.205 RETURN_CODE IDTechSDK.IDT_NEO2.lcd_startCameraCapture (ushort *timeout,* string *ip* = " ")

Start Camera Capture

When the camera is turned on, a “take a picture” button appears on the display. If the button is pressed, the device takes a picture, which is stored in SD card and displayed on-screen for five seconds. The device stores a maximum of 20 pictures. Upon reaching capacity, the 21st picture replaces the 1st.

Parameters

ip	timeout Timeout value. Minimum is 30 seconds (values under 30 seconds will default to 30 seconds)
ip	Optional IP Address

Returns

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

16.3.2.206 RETURN_CODE IDTechSDK.IDT_NEO2.lcd_startScanQR (ushort *timeout,* string *ip* = " ")

Start QR Scanning

This command will enable the camera and attempt to scan a QR code. The data will be returned to the Message↩ Callback, with DeviceState.CameraEventData

Parameters

ip	timeout Timeout value. Minimum is 30 seconds (values under 30 seconds will default to 30 seconds)
ip	Optional IP Address

Returns

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

16.3.2.207 RETURN_CODE IDTechSDK.IDT_NEO2.lcd_startScreenSaver (string *name,* string *ip* = " ")

Start Screen Saver

The command starts the screen saver, which is disabled when the touchpad is touched, the customize screen is shown, or a transaction is started. The device reads the video from the inserted SD Card; the video format must be MJPEG with a maximum frame width of 480px and maximum frame height of 272px. Note that the video displays rotated +90 degrees.

Parameters

name	Name of file on SD Card
ip	Optional IP Address

Returns

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

16.3.2.208 RETURN_CODE IDTechSDK.IDT_NEO2.lcd_stopAudio (string *ip* = " ")

Stop Audio

This command stop playing audio started with lcd_playAudio.

Parameters

ip	Optional IP Address

Returns

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

16.3.2.209 RETURN_CODE IDTechSDK.IDT_NEO2.lcd_stopCameraCapture (string *ip* = " ")

Stop Camera Capture

This command will stop the camera that started with lcd_startCameraCapture

Parameters

ip	Optional IP Address

Returns

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

16.3.2.210 RETURN_CODE IDTechSDK.IDT_NEO2.lcd_stopScanQR (string *ip* = " ")

Stop QR Scanning

This command will stop QR scanning that started with lcd_startScanQR

Parameters

ip	Optional IP Address

Returns

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

16.3.2.211 RETURN_CODE IDTechSDK.IDT_NEO2.lcd_storeScreenInfo (string *ip* = " ")

Store Screen Info

Store all current screen information from RAM to flash

Parameters

ip	Optional: IP address to execute command on (for IP connected devices)

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

16.3.2.212 RETURN_CODE IDTechSDK.IDT_NEO2.lcd_updateColor (string *screenName,* string *objectName,* byte[ ] *color,* string *ip =* " ")

Update Color

Updates the component color, or updates the LED colors if specifying LCD component

Parameters

screenName	Screen name that will be the target of update color
objecName	Identifier of the component
color	Non LCD Widget: Four bytes for color, example, Blue = 0xFF000000, Black = 0x00000000 • Byte 0 = B • Byte 1 = G • Byte 2 = R • Byte 3 = Reserved LCD Widget: Four bytes for LED color, byte 0 = LED 0, byte 1 = LED 1, byte 2 = LED2, byte 3 = LED 3 • Value 0 = LED OFF • Value 1 = LED Green • Value 2 = LED Yellow • Value 3 = LED Red
ip	Optional: IP address to execute command on (for IP connected devices)

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

16.3.2.213 RETURN_CODE IDTechSDK.IDT_NEO2.lcd_updateLabel (string *screenName,* string *objectName,* string *label,* string *ip =* " ")

Update Label

Updates the component label.

Parameters

screenName	Screen name that will be the target of update label
objectName	Identifier of the component
label	Label to show on the component
ip	Optional: IP address to execute command on (for IP connected devices)

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

16.3.2.214 RETURN_CODE IDTechSDK.IDT_NEO2.lcd_updatePosition (string *screenName,* string *objectName,* byte *alignment,* UInt16 *new_xCord,* UInt16 *new_yCord,* string *ip =* " ")

Update Position

Updates the component position.

Parameters

screenName	Screen Name that will be the target of update position
objectName	Identifier of the component
alignment	Alignment for the target • 0 = Display object at the horizon center of specified y, while x ignored • 1 = Display object at specified x andy • 2 = Display object at center of screen, x, y are both ignored • 3 = Display object at left of the screen of specified y, while x ignored • 4 = Display object at right of the screen of specified y, while x ignored
new_xCord	x-coordinate for Text, range 0-271
new_yCord	y-coordinate for Text, range 0-479
ip	Optional: IP address to execute command on (for IP connected devices)

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

16.3.2.215 RETURN_CODE IDTechSDK.IDT_NEO2.msr_cancelMSRSwipe (string *ip =* " ")

Disable MSR Swipe Cancels MSR swipe request.

Parameters

ip	Optional: IP address to execute command on (for IP connected devices)

Returns

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

16.3.2.216 RETURN_CODE IDTechSDK.IDT_NEO2.msr_getMSRTrack (ref int *val,* string *ip*)

Gets the MSR tracks setting.

Parameters

val	Track Value: • 0 = Any Track • 1 = Track 1 • 2 = Track 2 • 3 = Track 1, Track 2 • 4 = Track 3 • 5 = Track 1, Track 3 • 6 = Track 2, Track 3 • 7 = Track 1, Track 2, Track 3
ip	Optional IP parameter

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

16.3.2.217 RETURN_CODE IDTechSDK.IDT_NEO2.msr_setMSRTrack (int *val,* string *ip*)

Sets the MSR tracks to recognize. When "Any Track" is selected (0 - default value), an error condition will be returned if no tracks are read When specific tracks are selected (1 - 7), if any of the requested tracks are missing, it will return error.

Parameters

val	Track Value: • 0 = Any Track • 1 = Track 1 • 2 = Track 2 • 3 = Track 1, Track 2 • 4 = Track 3 • 5 = Track 1, Track 3 • 6 = Track 2, Track 3 • 7 = Track 1, Track 2, Track 3
ip	Optional IP parameter

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

16.3.2.218 RETURN_CODE IDTechSDK.IDT_NEO2.msr_startMSRSwipe (int *timeout,* string *ip* = " ")

Enable MSR Swipe

Enables MSR, waiting for swipe to occur.

Parameters

timeout	Swipe Timeout Value
ip	Optional: IP address to execute command on (for IP connected devices)

Returns

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

16.3.2.219 RETURN_CODE IDTechSDK.IDT_NEO2.msr_startMSRSwipe (int *timeout,* string *ip,* SwipeCallback *swipeCallback,* TimeoutCallback *timeoutCallback,* FailureCallback *failureCallback*)

Enable MSR Swipe

Enables MSR, waiting for swipe to occur.

Parameters

timeout	Swipe Timeout Value
ip	IP address to execute command on (for IP connected devices)
swipeCallback	Redirects the input from main callback to SwipeCallback(string ipAddress, IDTTransactionData transactionData)
failureCallback	Redirects the input from main callback to FailureCallback(string ipAddress, RETURN_CODE errorCode)
timeoutCallback	Redirects the input from main callback to TimeoutCallback(string ipAddress)

Returns

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

16.3.2.220 RETURN_CODE IDTechSDK.IDT_NEO2.pin_cancelPINEntry (string *ip* = " ")

Cancel PIN Entry

Cancel “Get Function Key” & “Get Encrypted PIN” & “Get Numeric” & “Get Amount”

Returns

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

16.3.2.221 RETURN_CODE IDTechSDK.IDT_NEO2.pin_capturePin (int *timeout,* int *type,* string *PAN,* int *minPIN,* int *maxPIN,* string *message,* string *ip* = " ")

Capture PIN

Parameters

timeout	Timeout, in seconds. Value of 0 will use system timeout, if any
type	PAN and Key Type • 00h = MKSK to encrypt PIN, Internal PAN (from MSR) • 01h = DUKPT to encrypt PIN, Internal PAN (from MSR) • 10h = MKSK to encrypt PIN, External Plaintext PAN • 11h = DUKPT to encrypt PIN, External Plaintext PAN • 20h = MKSK to encrypt PIN, External Ciphertext PAN • 21h = DUKPT to encrypt PIN, External Ciphertext PAN
PAN	Personal Account Number (if internal, value is 0)
minPIN	Minimum PIN Length
maxPIN	Maximum PIN Length
message	LCD Message
ip	Optional: IP address to execute command on (for IP connected devices) Results returned to MessageCallback. If successful PIN capture, data is returned as DeviceState.TransactionData, with IDTTransactionData cData.pin_pinblock and cData.pin_KSN. If timeout, returns DeviceState.PINTimeout If error, returns DeviceState.PINFail with one of the following values in data[0]: • 01h – Fail, Key Pad Cancel • 02h – Fail, External Command Cancel • 03h – Fail, Invalid input parameters • 04h – Fail, PAN error • 05h – Fail, PIN DUKPT Key is absent • 06h – Fail, PIN DUKPT Key is exhausted • 07h – Fail, Display message error • 0Ch – Fail, Operation Failed

Returns

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

16.3.2.222 RETURN_CODE IDTechSDK.IDT_NEO2.pin_capturePin (int *timeout,* int *type,* string *PAN,* int *minPIN,* int *maxPIN,* string *message,* string *ip,* SwipeCallback *inputCallback,* FailureCallback *failureCallback,* TimeoutCallback *timeoutCallback,* CancelPromptCallback *cancelPromptCallback*)

Capture PIN

Parameters

timeout	Timeout, in seconds. Value of 0 will use system timeout, if any

Parameters

type	PAN and Key Type • 00h = MKSK to encrypt PIN, Internal PAN (from MSR) • 01h = DUKPT to encrypt PIN, Internal PAN (from MSR) • 10h = MKSK to encrypt PIN, External Plaintext PAN • 11h = DUKPT to encrypt PIN, External Plaintext PAN • 20h = MKSK to encrypt PIN, External Ciphertext PAN • 21h = DUKPT to encrypt PIN, External Ciphertext PAN
PAN	Personal Account Number (if internal, value is 0)
minPIN	Minimum PIN Length
maxPIN	Maximum PIN Length
message	LCD Message
ip	IP address to execute command on (for IP connected devices) Results returned to MessageCallback. If successful PIN capture, data is returned as DeviceState.TransactionData, with IDTTransactionData cData.pin_pinblock and cData.pin_KSN. If timeout, returns DeviceState.PINTimeout If error, returns DeviceState.PINFail with one of the following values in data[0]: • 01h – Fail, Key Pad Cancel • 02h – Fail, External Command Cancel • 03h – Fail, Invalid input parameters • 04h – Fail, PAN error • 05h – Fail, PIN DUKPT Key is absent • 06h – Fail, PIN DUKPT Key is exhausted • 07h – Fail, Display message error • 0Ch – Fail, Operation Failed
inputCallback	Redirects the input from main callback to SwipeCallback(string ipAddress, IDTTransactionData input);
failureCallback	Redirects the input from main callback to FailureCallback(string ipAddress, RETURN_CODE errorCode)
timeoutCallback	Redirects the input from main callback to TimeoutCallback(string ipAddress)
cancelPromptCallback	Redirects the input from main callback to CancelPromptCallback(string ipAddress)

Returns

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

16.3.2.223 RETURN_CODE IDTechSDK.IDT_NEO2.pin_getFunctionKey (int *timeout,* string *ip* = " ")

Get Function Key

Results returned to MessageCallback. If successful function key capture, data is returned as DeviceState.Function↩ Key, with data being the ASCII value of the key that was pressed.

Parameters

timeout	Timeout, in seconds. Value of 0 will use system timeout, if any
ip	Optional: IP address to execute command on (for IP connected devices)

Returns

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

16.3.2.224 RETURN_CODE IDTechSDK.IDT_NEO2.pin_getFunctionKey (int *timeout,* string *ip,* SwipeCallback *inputCallback,* FailureCallback *failureCallback,* TimeoutCallback *timeoutCallback,* CancelPromptCallback *cancelPromptCallback*)

Get Function Key

Results returned to MessageCallback. If successful function key capture, data is returned as DeviceState.Function↩ Key, with data being the ASCII value of the key that was pressed.

Parameters

timeout	Timeout, in seconds. Value of 0 will use system timeout, if any
ip	IP address to execute command on (for IP connected devices)
inputCallback	Redirects the input from main callback to InputCallback(string ipAddress, IDTTranasctionData input);
failureCallback	Redirects the input from main callback to FailureCallback(string ipAddress, RETURN_CODE errorCode)
timeoutCallback	Redirects the input from main callback to TimeoutCallback(string ipAddress)
cancelPromptCallback	Redirects the input from main callback to CancelPromptCallback(string ipAddress)

Returns

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

16.3.2.225 RETURN_CODE IDTechSDK.IDT_NEO2.pin_getPanEntry (bool *csc,* bool *expDate,* bool *ADR,* bool *ZIP,* bool *mod10,* UInt16 *timeout,* bool *encPANOnly* = `false`, string *ip* = " ")

Get Pan

prompt the user to manually enter a card PAN and Expiry Date (and optionally CSC) from the keypad and return it to the POS.

Parameters

timeout	Number of seconds that the reader waits for the data entry session to complete, stored as a big-endian number. 0 = no timeout
csc	Request CSS
expDate	Request Expiration Date
ADR	Request Address
ZIP	Request Zip
mod10	Validate entered PAN passes MOD-10 checking before accepting
encPANOnly	Output only encrypted PAN
ip	Optional: IP address to execute command on (for IP connected devices)
swipeCallback	Optional: Redirects the input from main callback to SwipeCallback(string ipAddress, IDTTransactionData transactionData)

Parameters

timeoutCallback	Optional: Redirects the input from main callback to TimeoutCallback(string ipAddress)
cancelPromptCallback	Optional: Redirects the input from main callback to CancelPromptCallback(string ipAddress)

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

16.3.2.226 RETURN_CODE IDTechSDK.IDT_NEO2.pin_getPanEntry (bool *csc,* bool *expDate,* bool *ADR,* bool *ZIP,* bool *mod10,* UInt16 *timeout,* bool *encPANOnly,* string *ip,* SwipeCallback *swipeCallback,* TimeoutCallback *timeoutCallback,* CancelPromptCallback *cancelPromptCallback*)

Get Pan

prompt the user to manually enter a card PAN and Expiry Date (and optionally CSC) from the keypad and return it to the POS.

Parameters

timeout	Number of seconds that the reader waits for the data entry session to complete, stored as a big-endian number. 0 = no timeout
csc	Request CSS
expDate	Request Expiration Date
ADR	Request Address
ZIP	Request Zip
mod10	Validate entered PAN passes MOD-10 checking before accepting
encPANOnly	Output only encrypted PAN
ip	IP address to execute command on (for IP connected devices)
swipeCallback	Redirects the input from main callback to SwipeCallback(string ipAddress, IDTTransactionData transactionData)
timeoutCallback	Redirects the input from main callback to TimeoutCallback(string ipAddress)
cancelPromptCallback	Redirects the input from main callback to CancelPromptCallback(string ipAddress)

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

16.3.2.227 RETURN_CODE IDTechSDK.IDT_NEO2.pin_promptForAmount (int *timeout,* int *minLen,* int *maxLen,* string *message,* byte[ ] *signature,* string *ip* = " ")

Capture Amount

Parameters

timeout	Timeout, in seconds. Value of 0 will use system timeout, if any
minLen	Minimum input Length
maxLen	Maximum input Length
message	LCD Message

Parameters

signature	Display message signed by Numeric Private Key using RSAPSS algorithm: 1. Calculate 32 bytes Hash for "<Display Flag><Key max="" length>=""><Key Min Length><Plaintext display="" message>=""> 2. Using RSAPSS algorithm calculate the Hash to be 256 bytes Raw Data 3. Using Numeric Private Key to sign the Raw Data to be 256 bytes signature
ip	Optional: IP address to execute command on (for IP connected devices)

Returns

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

16.3.2.228 RETURN_CODE IDTechSDK.IDT_NEO2.pin_promptForAmount (int *timeout,* int *minLen,* int *maxLen,* string *message,* byte[] *signature,* string *ip,* SwipeCallback *inputCallback,* FailureCallback *failureCallback,* TimeoutCallback *timeoutCallback,* CancelPromptCallback *cancelPromptCallback*)

Capture Amount

Parameters

timeout	Timeout, in seconds. Value of 0 will use system timeout, if any
minLen	Minimum input Length
maxLen	Maximum input Length
message	LCD Message
signature	Display message signed by Numeric Private Key using RSAPSS algorithm: 1. Calculate 32 bytes Hash for "<Display Flag><Key max="" length>=""><Key Min Length><Plaintext display="" message>=""> 2. Using RSAPSS algorithm calculate the Hash to be 256 bytes Raw Data 3. Using Numeric Private Key to sign the Raw Data to be 256 bytes signature
ip	IP address to execute command on (for IP connected devices)
inputCallback	Redirects the input from main callback to SwipeCallback(string ipAddress, IDTTransactionData input);
failureCallback	Redirects the input from main callback to FailureCallback(string ipAddress, RETURN_CODE errorCode)
timeoutCallback	Redirects the input from main callback to TimeoutCallback(string ipAddress)
cancelPromptCallback	Redirects the input from main callback to CancelPromptCallback(string ipAddress)

Returns

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

16.3.2.229 RETURN_CODE IDTechSDK.IDT_NEO2.pin_promptForInput (int *messageID,* short *timeout,* string *ip* = `null`)

Capture Numeric Input From Message ID

Parameters

messageID	Message ID • 01 = "Please Swipe Or Key Employee ID", delayed masking, enable MSR, min len = 1, max len = 16
timeout	Timeout, in seconds. Value of 0 will use system timeout, if any
ip	Optional: IP address to execute command on (for IP connected devices)

Returns

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

16.3.2.230 RETURN_CODE IDTechSDK.IDT_NEO2.pin_promptForInput (int *messageID,* short *timeout,* string *ip,* SwipeCallback *inputCallback,* SwipeCallback *swipeCallback,* FailureCallback *failureCallback,* TimeoutCallback *timeoutCallback,* CancelPromptCallback *cancelPromptCallback*)

Capture Numeric Input From Message ID

Parameters

messageID	Message ID • 01 = "Please Swipe Or Key Employee ID", delayed masking, enable MSR, min len = 1, max len = 16
timeout	Timeout, in seconds. Value of 0 will use system timeout, if any
ip	IP address to execute command on (for IP connected devices)
inputCallback	Redirects the input from main callback to SwipeCallback(string ipAddress, IDTTransactionData input);
swipeCallback	Redirects the input from main callback to SwipeCallback(string ipAddress, IDTTransactionData transactionData)
failureCallback	Redirects the input from main callback to FailureCallback(string ipAddress, RETURN_CODE errorCode)
timeoutCallback	Redirects the input from main callback to TimeoutCallback(string ipAddress)
cancelPromptCallback	Redirects the input from main callback to CancelPromptCallback(string ipAddress)

Returns

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

16.3.2.231 RETURN_CODE IDTechSDK.IDT_NEO2.pin_promptForNumericKeyWithSwipe (short *timeout,* byte *function,* int *minLen,* int *maxLen,* string *line1,* string *line2,* byte[ ] *signature,* string *ip* = " ")

Capture Numeric Input

Parameters

timeout	Timeout, in seconds. Value of 0 will use system timeout, if any

Parameters

function	• 0x00 = Plaintext Input • 0x01 = Masked Input • 0x02 = Delayed Masking Input • 0x10 = Plaintext Input + MSR Active • 0x11 = Masked Input + MSR Active • 0x12 = Delayed Masking Input + MSR Active
minLen	Minimum input Length
maxLen	Maximum input Length
line1	Line 1 of LCD Message - 16 chars max
line2	Line 2 of LCD Message - 16 chars max
signature	Display message signed by Numeric Private Key using RSAPSS algorithm: 1. Calculate 32 bytes Hash for "<Display Flag><Key max="" length>=""><< Key Min Length><Plaintext display="" message>=""> 2. Using RSAPSS algorithm calculate the Hash to be 256 bytes Raw Data 3. Using Numeric Private Key to sign the Raw Data to be 256 bytes signature
ip	Optional: IP address to execute command on (for IP connected devices)

Returns

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

16.3.2.232 RETURN_CODE IDTechSDK.IDT_NEO2.pin_promptForNumericKeyWithSwipe (short *timeout,* byte *function,* int *minLen,* int *maxLen,* string *line1,* string *line2,* byte[ ] *signature,* string *ip,* SwipeCallback *inputCallback,* SwipeCallback *swipeCallback,* FailureCallback *failureCallback,* TimeoutCallback *timeoutCallback,* CancelPromptCallback *cancelPromptCallback*)

Capture Numeric Input

Parameters

timeout	Timeout, in seconds. Value of 0 will use system timeout, if any
function	• 0x00 = Plaintext Input • 0x01 = Masked Input • 0x02 = Delayed Masking Input • 0x10 = Plaintext Input + MSR Active • 0x11 = Masked Input + MSR Active • 0x12 = Delayed Masking Input + MSR Active
minLen	Minimum input Length
maxLen	Maximum input Length

Parameters

line1	Line 1 of LCD Message - 16 chars max
line2	Line 2 of LCD Message - 16 chars max
signature	Display message signed by Numeric Private Key using RSAPSS algorithm: 1. Calculate 32 bytes Hash for "<Display Flag><Key max="" length>=""><Key Min Length><Plaintext display="" message>=""> 2. Using RSAPSS algorithm calculate the Hash to be 256 bytes Raw Data 3. Using Numeric Private Key to sign the Raw Data to be 256 bytes signature
ip	: IP address to execute command on (for IP connected devices)
inputCallback	Redirects the input from main callback to SwipeCallback(string ipAddress, IDTTransactionData input);
swipeCallback	Redirects the input from main callback to SwipeCallback(string ipAddress, IDTTransactionData transactionData)
failureCallback	Redirects the input from main callback to FailureCallback(string ipAddress, RETURN_CODE errorCode)
timeoutCallback	Redirects the input from main callback to TimeoutCallback(string ipAddress)p
cancelPromptCallback	Redirects the input from main callback to CancelPromptCallback(string ipAddress)

Returns

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

16.3.2.233 RETURN_CODE IDTechSDK.IDT_NEO2.pin_sendBeep (string *ip* = " ")

Send Beep

Executes a beep request.

Parameters

ip	Optional: IP address to execute command on (for IP connected devices)

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

16.3.2.234 static String IDTechSDK.IDT_NEO2.SDK_Version ( ) [static]

SDK Version

- All Devices

Returns the current version of SDK

Returns

Framework version

16.3.2.235 static void IDTechSDK.IDT_NEO2.setCallback (CallBack *my_Callback*) [static]

Set Callback

Sets the class callback

16.3.2.236 static void IDTechSDK.IDT_NEO2.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

16.3.2.237 static void IDTechSDK.IDT_NEO2.setCallbackIP (CallBackIP *my_Callback,* string *ip* = " ") [static]

Set Callback with IP

Sets the class callback that also reports back IP device information (for network connected devices)

Parameters

my_Callback	The callback function to receive the response message from device. defined as follows. public delegate void CallBackIP(IDT_DEVICE_Types sender, DeviceState state, byte[] data, IDTTransactionData card, EMV_Callback emvCallback, RETURN_CODE transactionResultCode, string IP);
ip	Optional: The callback function will only be applicable to the provided IP.

16.3.2.238 static void IDTechSDK.IDT_NEO2.setCommandTimeout (int *milliseconds*) [static]

Set Command Timeout

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

Parameters

milliseconds	Time

16.3.2.239 static void IDTechSDK.IDT_NEO2.setLongPressCallback (longPressCallback *callback,* string *ip* = " ") [static]

Set Longpress Callback

Sets the class callback that also reports back IP device information (for network connected devices)

Parameters

callback	The callback function to receive the response message from device. defined as follows. Location 0 = top left, Location 1 = top right. public delegate void longPressCallback(IDT_DEVICE_Types sender, UInt16 screenID, byte location, string IP);
ip	Optional: The callback function will only be applicable to the provided IP.

16.3.2.240 static bool IDTechSDK.IDT_NEO2.useSerialPort (int *port*) [static]

Use Serial Port Interface

Instructs SDK to attempt to use the Serial Port for communication with IDT_NEO2 using default baud rate

Parameters

port	Serial Port to use. Example COM1 = 1.

Returns

bool TRUE=successful, FALSE=failure

16.3.2.241 static bool IDTechSDK.IDT_NEO2.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 IDT_NEO2

Parameters

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

Returns

bool TRUE=successful, FALSE=failure

16.3.2.242 static bool IDTechSDK.IDT_NEO2.useUSB ( ) [static]

Use USB Interface

Instructs SDK to attempt to use USB for communication with IDT_NEO2

Return values

success	Returns TRUE if connection successful, otherwise returens false

16.3.3 Property Documentation

16.3.3.1 IDT_NEO2 IDTechSDK.IDT_NEO2.SharedController [static],[get]

Singleton Instance

Establishes an singleton instance of IDT_NEO2 class.

Returns

Instance of IDT_NEO2

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

- Source_Windows/IDT_NEO2.cs

16.4 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
- byte[] finalPAN

16.4.1 Detailed Description

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

16.4.2 Member Data Documentation

16.4.2.1 byte [] IDTechSDK.IDTCryptoData.BDK

Base Derivation Key.

16.4.2.2 byte [] IDTechSDK.IDTCryptoData.clearPinBlock

Decrypted Pin Block

16.4.2.3 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)

16.4.2.4 byte [ ] IDTechSDK.IDTCryptoData.dataToProcess

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

16.4.2.5 byte [ ] IDTechSDK.IDTCryptoData.DataVariant

Data Encryption Key (variant of DEK).

16.4.2.6 byte [ ] IDTechSDK.IDTCryptoData.DEK

Derived Encryption Key.

16.4.2.7 string IDTechSDK.IDTCryptoData.errorString

Encryption/Decryption Error.

16.4.2.8 byte [ ] IDTechSDK.IDTCryptoData.finalPAN

Final PAN

16.4.2.9 byte [ ] IDTechSDK.IDTCryptoData.IPEK

Initial Public Encryption Key.

16.4.2.10 bool IDTechSDK.IDTCryptoData.isDecryption

TRUE = Decrypt Data. FALSE = Encrypt Data

16.4.2.11 bool IDTechSDK.IDTCryptoData.isTDES

TRUE = Use TDES. FALSE = Use AES

16.4.2.12 int IDTechSDK.IDTCryptoData.keyVariant

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

16.4.2.13 byte [ ] IDTechSDK.IDTCryptoData.KSN

Key Serial Number.

16.4.2.14 bool IDTechSDK.IDTCryptoData.MAC_Command

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

16.4.2.15 byte [ ] IDTechSDK.IDTCryptoData.MACVariant

Message Authentication Challenge Key (variant of DEK).

16.4.2.16 string IDTechSDK.IDTCryptoData.PAN

Primary Account Number used with clearPinBlock to derive/encode PIN

16.4.2.17 string IDTechSDK.IDTCryptoData.PIN

PIN derived from clearPinBlock, or used to create clearPinBlock

16.4.2.18 byte [ ] IDTechSDK.IDTCryptoData.pinBlock

Encrypted Pin Block

16.4.2.19 int IDTechSDK.IDTCryptoData.PINBlockType

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

16.4.2.20 byte [ ] IDTechSDK.IDTCryptoData.PINVariant

PIN Encryption Key (variant of DEK).

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

- Source_Windows/IDT_Transactions.cs

16.5 IDTechSDK.IDTTransactionData Class Reference

Public Member Functions

- **IDTTransactionData** (byte[ ] rawData=null)

Static Public Member Functions

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

Public Attributes

- string Base64
- 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
- string fastEMV
- string message
- string msr_KBOutput

16.5.1 Detailed Description

Class for swipe data

16.5.2 Member Data Documentation

16.5.2.1 string IDTechSDK.IDTTransactionData.Base64

Raw Transaction Data converted to Base64.

16.5.2.2 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;

16.5.2.3 byte [ ] IDTechSDK.IDTTransactionData.captured_CSC

Captured Customer Service Code

16.5.2.4 byte [ ] IDTechSDK.IDTTransactionData.captured_Expiry

Captured Expiry Date

16.5.2.5 string IDTechSDK.IDTTransactionData.captured_firstPANDigits

First plaintext PAN Digits

16.5.2.6 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).

16.5.2.7 byte [ ] IDTechSDK.IDTTransactionData.captured_KSN

KSN used to encrypt manually captured PAN from keyed input

16.5.2.8 string IDTechSDK.IDTTransactionData.captured_lastPANDigits

Last plaintext PAN digits

16.5.2.9 byte [ ] IDTechSDK.IDTTransactionData.captured_MACKSN

KSN for MAC DUKPT key.

16.5.2.10 byte [ ] IDTechSDK.IDTTransactionData.captured_MACValue

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

16.5.2.11 byte [ ] IDTechSDK.IDTTransactionData.captured_PAN

Manually captured PAN from keyed input

16.5.2.12 bool IDTechSDK.IDTTransactionData.captured_SHA256

TRUE = SHA-256, FALSE = SHA-1

16.5.2.13 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

16.5.2.14 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

16.5.2.15 CTLS_APPLICATION IDTechSDK.IDTTransactionData.ctlsApplication

CTLS Application

16.5.2.16 String IDTechSDK.IDTTransactionData.device_RSN

Get the Reader Serial Number.

16.5.2.17 CEMV_APP_ERROR_FN IDTechSDK.IDTTransactionData.emv_appErrorFn

EMV App Error Function (select AR products)

16.5.2.18 CEMV_APP_ERROR_STATE IDTechSDK.IDTTransactionData.emv_appErrorState

EMV App Error State (select AR products)

16.5.2.19 byte [ ] IDTechSDK.IDTTransactionData.emv_clearingRecord

clearing record TLV

16.5.2.20 byte [ ] IDTechSDK.IDTTransactionData.emv_encipheredOnlinePIN

enciphered Online Pin

16.5.2.21 byte [ ] IDTechSDK.IDTTransactionData.emv_encryptedTags

Encrypted Tags TLV

16.5.2.22 EXTENDED_STATUS_CODES IDTechSDK.IDTTransactionData.emv_ESC

Extended Status Code (select AR products)

16.5.2.23 bool IDTechSDK.IDTTransactionData.emv_hasAdvise

Advise

16.5.2.24 bool IDTechSDK.IDTTransactionData.emv_hasReversal

Reversal

16.5.2.25 byte [ ] IDTechSDK.IDTTransactionData.emv_maskedTags

Masked Tags TLV

16.5.2.26 EMV_RESULT_CODE IDTechSDK.IDTTransactionData.emv_resultCode

EMV Result Code

16.5.2.27 RF_STATE IDTechSDK.IDTTransactionData.emv_RF_State

RF_State (select AR products)

16.5.2.28 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.

16.5.2.29 TRANS_ERROR_CODE IDTechSDK.IDTTransactionData.emv_transaction_Error_Code

Transaction Error Code (select AR products)

16.5.2.30 byte [ ] IDTechSDK.IDTTransactionData.emv_unencryptedTags

Unencrypted Tags TLV

16.5.2.31 EVENT_TRANSACTION_DATA_Types IDTechSDK.IDTTransactionData.Event

Transaction Data type,please see EVENT_TRANSACTION_DATA_Types for more information.

16.5.2.32 string IDTechSDK.IDTTransactionData.fastEMV

if a Fast EMV transaction was specified, this field will contain the ASCII version of the data similiar to IDTech FastEMV KB output format.

16.5.2.33 bool IDTechSDK.IDTTransactionData.hasMACVerificationData

Existance of MAC Verification Data for Encrypted Data

16.5.2.34 int IDTechSDK.IDTTransactionData.iccPresent

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

16.5.2.35 int IDTechSDK.IDTTransactionData.isCTLS

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

16.5.2.36 byte [ ] IDTechSDK.IDTTransactionData.mac

Message Authentication Code

16.5.2.37 byte [ ] IDTechSDK.IDTTransactionData.macKSN

Message Authentication Code Key Serial Number

16.5.2.38 string IDTechSDK.IDTTransactionData.message

if a Notification == EVENT_NOTIFICATION_Types.EVENT_NOTIFICATION_Message, this will contain the string message.

16.5.2.39 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;

16.5.2.40 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

16.5.2.41 byte [ ] IDTechSDK.IDTTransactionData.msr_encTrack1

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

16.5.2.42 byte [ ] IDTechSDK.IDTTransactionData.msr_encTrack2

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

16.5.2.43 byte [ ] IDTechSDK.IDTTransactionData.msr_encTrack3

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

16.5.2.44 int IDTechSDK.IDTTransactionData.msr_errorCode

Contains error code when data is not returned

16.5.2.45 byte [ ] IDTechSDK.IDTTransactionData.msr_extendedField

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

16.5.2.46 byte [ ] IDTechSDK.IDTTransactionData.msr_hashTrack1

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

16.5.2.47 byte [ ] IDTechSDK.IDTTransactionData.msr_hashTrack2

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

16.5.2.48 byte [ ] IDTechSDK.IDTTransactionData.msr_hashTrack3

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

16.5.2.49 string IDTechSDK.IDTTransactionData.msr_KBOutput

String data of IDTech Enhanced MSR Format that would output if MSR data was captured by Keyboard

16.5.2.50 KEY_VARIANT_TYPE IDTechSDK.IDTTransactionData.msr_keyVariantType

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

16.5.2.51 byte [ ] IDTechSDK.IDTTransactionData.msr_KSN

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

16.5.2.52 byte [ ] IDTechSDK.IDTTransactionData.msr_rawData

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

16.5.2.53 byte [ ] IDTechSDK.IDTTransactionData.msr_sessionID

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

16.5.2.54 String IDTechSDK.IDTTransactionData.msr_track1

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

16.5.2.55 int IDTechSDK.IDTTransactionData.msr_track1Length

Get the swiped card length of Track1 data.

16.5.2.56 String IDTechSDK.IDTTransactionData.msr_track2

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

16.5.2.57 int IDTechSDK.IDTTransactionData.msr_track2Length

Get the swiped card length of Track2 data.

16.5.2.58 String IDTechSDK.IDTTransactionData.msr_track3

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

16.5.2.59 int IDTechSDK.IDTTransactionData.msr_track3Length

Get the swiped card length of Track3 data.

16.5.2.60 EVENT_NOTIFICATION_Types IDTechSDK.IDTTransactionData.Notification

Event Notification type,please see EVENT_NOTIFICATION_Types for more information.

16.5.2.61 string IDTechSDK.IDTTransactionData.pin_KeyEntry

KSN for Pinblock

16.5.2.62 string IDTechSDK.IDTTransactionData.pin_KSN

KSN for Pinblock

16.5.2.63 string IDTechSDK.IDTTransactionData.pin_pinblock

PIN block from PINPAD

16.5.2.64 byte IDTechSDK.IDTTransactionData.SW1

SW1

16.5.2.65 byte IDTechSDK.IDTTransactionData.SW2

SW2

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

- Source_Windows/IDT_Transactions.cs

Index