IDTech Universal SDK Guide

for C / C++ / Java Developers

#80144504-001

Rev. A

Revision History

Revision	Description and Reason for Change	Date
A	Initial Release - Manual;User;C/C++/JAVA;SDK	02/14/2017

Contents

Chapter 1

ID TECH Universal SDK Reference Guide for Linux/Windows/Mac (C++/Java)

ID TECH provides this Universal SDK to drive multiple devices across multiple platforms.

The current version of the SDK supports the USB-HID interface of the listed ID TECH products. For devices that also have RS-232 interfaces, SDK communication support for COM on those devices is under development and will be released at a later date.

This SDK encompasses support for the following devices and platforms. Other ID TECH products are scheduled to be added in an upcoming release. Java supports all the devices that C/C++ supports.

C/C++

- **Platforms:** Macintosh, Windows, Linux Desktop (x86_64/amd64), Linux ARM (RaspberryPi)
- **Products:** UniPay 1.5, UniPay III, VP4880, MiniSmartII, BTPayMini, SpectrumPro, Kiosk III, Augusta

Java

- **Platforms:** Macintosh, Windows, Linux Desktop (x86_64/amd64), Linux ARM (RaspberryPi)
- **Products:** UniPay 1.5, UniPay III, VP4880, MiniSmartII, BTPayMini, SpectrumPro, Kiosk III, Augusta

Pre-requisites:

- **Macintosh:** None
- **Windows, Linux:** libusb-1.0

  Note:∗∗ We recommended installing libusb from the distributors' downloads. If that becomes a challenge, the SDK includes just the libusb library for Windows, Linux x86_65, and Linux ARM.

1.1 Demo Apps

Pre-Requisites C/C++:

- Eclipse for C/C++
- Windows: MinGW
- Linux ARM on x86_64 (cross compile): Poky

Pre-Requisites Java:

- Eclipse for Java Devlopers
- Java 1.8 or greater

1.2 Purpose

This document describes API requirements as well as the interface definitions and requirements for an integrator wishing to integrate it into a 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 Ma80149503sterCards 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 ID TECH 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, IEEE 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 SNMP 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

ID TECH 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

emv_startTransaction()

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

Authenticate EMV Transaction

emv_authenticateTransaction()

When the results to **emv_startTransaction()** come back as **EMV_RESULT_CODE.EMV_RESULT_CODE_AUTHENTICATE_TRANSACTION**, continuing the EMV transaction requires calling this method.

Complete Online EMV Transaction

emv_completeTransaction()

If start/authenticate transaction returns **EMV_RESULT_CODE.EMV_RESULT_CODE_GO_ONLINE**, finishing the transaction requires executing **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, finishing the EMV transaction still requires passing "TRUE" for **commError**.

Terminal Configuration

emv_retrieveTerminalData()
emv_removeTerminalData()
emv_setTerminalData()
Methods for terminal configuration. When setting terminal data, pass the tags in TLV format.

AID Management

emv_retrieveApplicationData()
emv_removeApplicationData()
emv_removeAllApplicationData()
emv_setApplicationData()

emv_retrieveAIDList()
Methods for AID management. When setting the AID, pass the tags in TLV format. When retrieving the AID, receive the results as tags in TLV format.

CAPK Management

emv_retrieveCAPK()
emv_removeCAPK()
emv_removeAllCAPK()
emv_setCAPK()
emv_retrieveCAPKList()

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

CRL Management

emv_removeCRL()
emv_removeAllCRL()
emv_retrieveCRL()
emv_setCRL()

Methods for Certificate Revocation List management.

Kernel Version

emv_getEMVKernelVersion()

Method to retrieve the kernel version.

Kernel Check Value

emv_getEMVKernelCheckValue()

Method to retrieve the kernel Check Value.

EMV Configuration Check Value

emv_getEMVConfigurationCheckValue()

Method to retrieve the EMV configuration check value.

3.2 MSR Methods

Start MSR Swipe

msr_startMSRSwipe()

Starts a swipe request. Returns card data in the callback method.

Cancel MSR Swipe

msr_cancelMSRSwipe()

Cancels a swipe request.

Chapter 4

Core Implementation: C/C++

IDTechSDK (libIDTechSDK.so-x_xx_xxx / libIDTechSDK-x_xx_xxx.dll / libIDTechSDK-x_xx_xxx.dylib) includes API methods to interface with ID TECH devices. This guide assumes a fair understanding of Eclipse, C, and general Linux, Windows, or Mac programming knowledge.

4.1 Integrating with IDTechSDK

- Import the Necessary Libraries
- Add Include Statements to Use Libraries
- Implement the Callback Function
- Initialize the Target Device

4.2 Import the Necessary Libraries

Header Files

Communicating with ID TECH devices requires that developers include the following header files in the project's source code folder:

- IDTDef.h
- The appropriate device header file for a single device OR **IDT_Device.h** to expose all methods for all devices. Usually, using the single device header file is appropriate as it filters out all unrelated methods.

Libraries

- libUSB installed on the system (not applicable to Mac).
- Place the IDTechSDK library file the system PATH of the target device.

IMPORTANT:

IDTechSDK libraries are distributed with versioning as part of the library name (libIDTechSDK.so-x_xx_xx x/ libIDTechSDK-x_xx_xxx.dl / libIDTechSDK-x_xx_xxx.dylib). The system must recognize them as a file WITHOUT the version info ((libIDTechSDK.so / libIDTechSDK.dll / libIDTechSDK.dylib). Accomplish this by either RENAMING the libraries by removing the version info OR create a Symbolic Link pointing to the original libraries and then remove the version info from the symbolic link:
(Symbolic Link) libIDTechSDK.so -> libIDTechSDK.so-x_xx_xxx (Compiled Library)
(Symbolic Link) libIDTechSDK.dll -> libIDTechSDK-x_xx_xxx.dll (Compiled Library)
(Symbolic Link) libIDTechSDK.dylib -> libIDTechSDK-x_xx_xxx.dylib (Compiled Library)
(Symbolic Link) /lib/libIDTechSDK.so -> /home/<USER>/proj/libIDTechSDK.so-x_xx_xxx (Compiled Library)

4.3 Add Include Statements to Use Libraries

Add a line of code to use the header files for IDTechSDK at the start of the file (a SpectrumPro header file is used here as an example):

```
#include <stdlib.h>
#include <stdio.h>

#include "IDTDef.h"
#include "libIDT_SpectrumPro.h"
```

4.4 Implement the Callback Function

There are two callbacks to implement, one for MSR and one for EMV:

```
void MSR_callBack(int type, IDTMSRData cardData){
    printf("\nMSR Callback\n");
    switch (type){
        case MSR_callBack_type_ERR:
            printf("Callback MSR cancelled\n");
        break;
        case MSR_callBack_type_RETURN_CODE:
            printf("Callback MSR data received\n");
        break;
        case MSR_callBack_type_TIMEOUT:
            printf("MSR Callback Timeout\n");
        break;
        default:
        break;
    }
}


void EMV_callBack(int device_type, int device_state, unsigned char * data, int dataLen,IDTTransactionData*
      cardData,EMV_Callback* emvCallback, int transactionResultCode){

    switch(device_state)
    {
    case EMVCallback:
        printf ("EMV Callback\n");
        break;
    case TransactionData:
        printf ("Transaction Data Callback\n");
        break;
    case TransactionFailed:
        printf("Transaction Failed Callback");
        break;

    }
}
```

4.5 Initialize SDK and Set the Target Device

Perform **device_init()** to initialize the SDK, establish the callbacks, and use the **device_setCurrentDevice()** function to specify the device to use.

```
int main(void) {
    int r = 0;
    printf("Initializing SDK...\n");
    r = device_init();
    if ( r != RETURN_CODE_DO_SUCCESS ) {
        printf("  Fail to init!\n");
            return 0;
    }
    emv_registerCallBk(EMV_callBack);
    msr_registerCallBk(MSR_callBack);
    device_setCurrentDevice(IDT_DEVICE_SPECTRUM_PRO)
    return 0;
}
```

Chapter 5

Core Implementation: Java

The **jnibridge.jar** Java library includes API methods to interface with ID TECH devices, accomplished by mapping Java methods onto native C methods used in IDTechSDK (IDTechSDK.so-x_xx_xxx / IDTechSDK-x_xx_xxx.dll / IDTechSDK-x_xx_xxx.dylib). This guide assumes a fair understanding of Eclipse, C, and general Linux, Windows, or Mac programming knowledge.

5.1 Integrating with IDTechSDK

- Add the Necessary Libraries
- Add Import Statements to Use Libraries
- Implement the Callback Function
- Initialize the SDK

5.2 Add the Necessary Libraries

Java Library

- import jnibridge.jar into your Java application.

Native Libraries

- libUSB must be installed on the system (not applicable to Mac).
- Place the IDTechSDK library file the target device's system PATH.

IMPORTANT:

IDTechSDK libraries are distributed with versioning as part of the library name (IDTechSDK.so-x_xx_xx x/ IDTechSDK-x_xx_xxx.dl / IDTechSDK-x_xx_xxx.dylib). The system must recognize them as a file WITHOUT the version info ((IDTechSDK.so / IDTechSDK.dll / IDTechSDK.dylib). Accomplish this by either RENAMING the libraries by removing the version info OR create a Symbolic Link pointing to the original libraries and then remove the version info from the symbolic link:
(Symbolic Link) IDTechSDK.so -> IDTechSDK.so-x_xx_xxx (Compiled Library)
(Symbolic Link) IDTechSDK.dll -> IDTechSDK-x_xx_xxx.dll (Compiled Library)
(Symbolic Link) IDTechSDK.dylib -> IDTechSDK-x_xx_xxx.dylib (Compiled Library)

5.3 Add Import Statements to Use Libraries

Add code to use the the Java bridge methods for IDTechSDK at the start of the file (using example of UniPayIII as the target device):

```
import IDTechSDK.ICCReaderStatusStruct;
import IDTechSDK.IDTEMVData;
import IDTechSDK.IDTMSRData;
import IDTechSDK.IDT_UniPayIII;
import IDTechSDK.IDTechSDKBridge;
import IDTechSDK.OnReceiverListener;
import IDTechSDK.ReaderInfo;
import IDTechSDK.ResDataStruct;
import IDTechSDK.StructConfigParameters;
```

5.4 Implement the Callback Function

Create a static class named **OnReceiverListenerImp** that implements **OnReceiverListener**. Put in all the required listeners.

NOTE:** although many of the items below are not used in this Native implementation, they still must be defined.

```
static class OnReceiverListenerImp implements OnReceiverListener {
    public void swipeMSRData(IDTMSRData card) {
    }

    public void lcdDisplay(int mode, String[] lines, int timeout) {
    }

    public void emvTransactionData(IDTEMVData emvData) {
    }

    public void deviceConnected() {
    }

    public void deviceDisconnected() {
    }

    public void timeout(int errorCode) {
    }

    public void autoConfigCompleted(StructConfigParameters profile) { } //NOT USED

    public void autoConfigProgress(int progressValue) { } //NOT USED

    public void msgRKICompleted(String MACResult) { } //NOT USED

    public void ICCNotifyInfo(byte[] dataNotify , String strMessage) { } //NOT USED

    public void msgBatteryLow() { } //NOT USED

    public void LoadXMLConfigFailureInfo(int index , String strMessage) { } //NOT USED

    public void msgToConnectDevice() { } //NOT USED

    public void msgAudioVolumeAjustFailed() { } //NOT USED

    public void dataInOutMonitor(byte[] data, boolean isIncoming) {
    }
}
```

5.5 Initialize SDK

Create an instance of **OnReceiverListenerImp**, then pass that to a new instance of the device class with which to communicate.

NOTE:** the SDK has a logging function that can be enabled by passing a parameter "2" as initial arguments.

```
public static void main(String[] args) {
    if (args.length > 0){
        System.out.println("\tLogging Level " + args[0]);
        IDTechSDK.IDTechSDKBridge.enableLogging(Integer.valueOf(args[0]));
    }
    OnReceiverListenerImp MessageCallBack = new OnReceiverListenerImp();
    device = new IDT_UniPayIII(MessageCallBack);

}
```

See JNI_TEST included with SDK for an example of communicating with a UniPay III.

Chapter 6

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 CORRECTO?	确定金
3	MSG_APPROVED	APPROVED	APPROUVE	APROVADO	通
4	MSG_CALL_YOUR_BANK	CALL YOUR BANK	APPE VOTRE BANQE	LLAME A SU BANCO	系您的行
5	MSG_CANCEL_OR_ENTER	CANCEL OR ENTER	ANNULE OU ENTRER	CANCEL O ENTRAR	取消或确定
6	MSG_CARD_ERROR	CARD ERROR	ERREUR CARTE	ERROR DE TARJETA	卡
7	MSG_DECLINED	DECLINED	REFUSE	DECLINADO	卡被拒
8	MSG_ENTER_AMOUNT	ENTER AMOUNT	ENTRER MONTANT	INGRESE MONTO	入金
9	MSG_ENTER_PIN	ENTER PIN:	ENTRER PIN:	ENTRAR NPI:	入密
10	MSG_INCORRECT_PIN	INCORRECT PIN	NIP INCORRECT	NPI INCORRECTO	密
11	MSG_ICC_MSR1	SWIPE OR INSERT	PASSER OU INSERT	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 TARJETA	插卡
14	MSG_USE_CHIP_READER	USE CHIP READER UTI	LECTEUR CHIP	USO CHIP LECTOR	使用芯片卡
15	MSG_NOT_ACCEPTED	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 ESPERE	等候中

18	MSG_PROCESSING_ERROR	PROCESSING ERROR	ERREUR DE TRAITE	ERROR PROCESANDO	理
19	MSG_USE_MAGSTRIPE	USE MAGSTRIPE	USAGE MAGSTRIPE	USO DE MAGSTRIPE	使用磁卡
20	MSG_TRY_AGAIN	TRY AGAIN	REESSAYER	VUELV INTENTARLO	重
21	MSG_ONLINE	GO ONLINE	GO LIGNE	GO LINEA	在
22	MSG_TRANSACTION_ERROR	TRANSACTION ERR	ERREUR DE TRANS	ERROR DE TRANSAC	交易
23	MSG_TERMINATE	TERMINATE	RESILIER	TERMINAR	止
24	MSG_ADVICE	ADVICE	CONSEILS	CONSEJOS	建
25	MSG_TIMEOUT	TIME OUT	TIMEOUT	TIEMPO DE ESPERA	超
26	MSG_PROCESSING	PROCESSING...	PROCESSUS...	PROCESANDO...	理中。。。
27	MSG_PIN_TRY_EX	PIN TRY LIMIT EX	PIN TRY DEPASSE	TRY PIN SUPERADA	密次多
28	MSG_ISSUER_AUTH_FAIL	ISSUER AUTH FAIL	EMETTEUR FAIL	EMISOR FALLA	与卡机构
29	MSG_CONTINUE_PROCESS	CONTINUE PROCESS	CONTINUER LA	CONTINUAR PROCES	理
30	MSG_GET_PIN_ERROR	GET PIN ERROR	GET PIN ERROR	OBTENER PIN ERR	密
31	MSG_GET_PIN_FAIL	GET PIN FAIL	GET PIN FAIL	OBTENER PIN FALL	取密
32	MSG_NOKEY_GET_PIN	NO KEY GET PIN	NO KEY GET PIN	NO CLAVE GET PIN	法入密
33	MSG_CANCELLED	CANCELLED	ANNULE	CANCELADO	取消
34	MSG_LAST_PIN_TRY	LAST PIN TRY	-	-	最后一次入密
35	MSG_WELCOME	WELCOME	BIENVENUE	BIENVENIDOS	迎使用
36	MSG_AMOUNT_OTHER	AMOUNT OTHER	MONTANT AUTRES	IMPORTE OTRAS	返
37	MSG_ENTER_AMOUNT_OTHER	ENTER AMOUNT OTHER	ENTRER MONTANT AUTRES	ENTRAR IMPORTE OTRAS	入返
38	MSG_CAPK_HASH_VALUE_FAIL	CAPK HASH VALUE FAIL	CAPK HASH VALEUR FAIL	CAPK HASH VALOR FAIL	公哈希值
64	MSG_TRY_MSR_AGAIN	TRY MSR AGAIN	-	-	再次使用磁卡
65	MSG_LAST_MSR_TRY	LAST MSR TRY	-	-	最后一次使用磁卡
66	MSG_ TRY_ICC_AGAIN	TRY ICC AGAIN	INSERER VOTRE CARTE	INTENTE ICC DE NUEVO	再次使用芯片卡

67	MSG_REMOV-E_CARD	REMOVE CARD	RETIRER VOTRE CARTE	QUITE TARJETA	移卡

Chapter 7

Error Code Reference

```
0: "no error, beginning task";
1: "no response from reader";
2: "invalid response data";
3: "time out for task or CMD";
4: "wrong parameter";
5: "SDK is doing MSR or ICC task";
6: "SDK is doing PINPad task";
7: "SDK is doing CTLS task";
8: "SDK is doing EMV task";
9: "SDK is doing Other task";
10: "err response or data";
11: "no reader attached";
12: "mono audio is enabled";
13: "did connection";
14: "audio volume is too low";
15: "task or CMD be canceled";
16: "UF wrong string format";
17: "UF file not found";
18: "UF wrong file format";
19: "Attempt to contact online host failed";
20: "Attempt to perform RKI failed";
22: "Buffer size is not enough";
0x300: "Key Type(TDES) of Session Key is not same as the related Master Key.";
0x400: "Related Key was not loaded.";
0x500: "Key Same.";
0x501: "Key is all zero";
0x502: "TR-31 format error";
0x702: "PAN is Error Key.";
0x705: "No Internal MSR PAN (or Internal MSR PAN is erased timeout)";
0X0C01: "Incorrect Frame Tag";
0X0C02: "Incorrect Frame Type";
0X0C03: "Unknown Frame Type";
0X0C04: "Unknown Command";
0X0C05: "Unknown Sub-Command";
0X0C06: "CRC Error";
0X0C07: "Failed";
0X0C08: "Timeout";
0X0C0A: "Incorrect Parameter";
0X0C0B: "Command Not Supported";
0X0C0C: "Sub-Command Not Supported";
0X0C0D: "Parameter Not Supported / Status Abort Command";
0X0C0F: "Sub-Command Not Allowed";
0X0D01: "Incorrect Header Tag";
0X0D02: "Unknown Command";
0X0D03: "Unknown Sub-Command";
0X0D04: "CRC Error in Frame";
0X0D05: "Incorrect Parameter";
0X0D06: "Parameter Not Supported";
0X0D07: "Mal-formatted Data";
0X0D08: "Timeout";
0X0D0A: "Failed / NACK";
0X0D0B: "Command not Allowed";
0X0D0C: "Sub-Command not Allowed";
0X0D0D: "Buffer Overflow (Data Length too large for reader buffer)";
0X0D0E: "User Interface Event";
0X0D11: "Communication type not supported, VT-1, burst, etc.";
0X0D12: "Secure interface is not functional or is in an intermediate state.";
0X0D13: "Data field is not mod 8";
0X0D14: "Pad 0x80 not found where expected";
0X0D15: "Specified key type is invalid";
0X0D1: "Could not retrieve key from the SAM(InitSecureComm)";
0X0D17: "Hash code problem";
0X0D18: "Could not store the key into the SAM(InstallKey)";
```

```
0X0D19: "Frame is too large";
0X0D1A: "Unit powered up in authentication state but POS must resend the InitSecureComm command";
0X0D1B: "The EEPROM may not be initialized because SecCommInterface does not make sense";
0X0D1C: "Problem encoding APDU";
0X0D20: "Unsupported Index(ILM) SAM Transceiver error - problem communicating with the SAM(Key Mgr)";
0X0D2: "Unexpected Sequence Counter in multiple frames for single bitmap(ILM) Length error in data returned
       from the SAM(Key Mgr)";
0X0D22: "Improper bit map(ILM)";
0X0D23: "Request Online Authorization";
0X0D24: "ViVOCard3 raw data read successful";
0X0D25: "Message index not available(ILM) ViVOcomm activate transaction card type(ViVOcomm)";
0X0D26: "Version Information Mismatch(ILM)";
0X0D27: "Not sending commands in correct index message index(ILM)";
0X0D28: "Time out or next expected message not received(ILM)";
0X0D29: "ILM languages not available for viewing(ILM)";
0X0D2A: "Other language not supported(ILM)";
0X0D41: "Unknown Error from SAM";
0X0D42: "Invalid data detected by SAM";
0X0D43: "Incomplete data detected by SAM";
0X0D44: "Reserved";
0X0D45: "Invalid key hash algorithm";
0X0D46: "Invalid key encryption algorithm";
0X0D47: "Invalid modulus length";
0X0D48: "Invalid exponent";
0X0D49: "Key already exists";
0X0D4A: "No space for new RID";
0X0D4B: "Key not found";
0X0D4C: "Crypto not responding";
0X0D4D: "Crypto communication error";
0X0D4E: "Module-specific error for Key Manager";
0X0D4F: "All key slots are full (maximum number of keys has been installed)";
0X0D50: "Auto-Switch OK";
0X0D51: "Auto-Switch failed";
0X0D90: "Account DUKPT Key not exist";
0X0D91: "Account DUKPT Key KSN exausted";
0x0D00: "This Key had been loaded.";
0x0E00: "Base Time was loaded.";
0x0F00: "Encryption Or Decryption Failed.";
0x1000: "Battery Low Warning (It is High Priority Response while Battery is Low.)";
0x1800: "Send 'Cancel Command' after send 'Get Encrypted PIN' & 'Get Numeric' & 'Get Amount';
0x1900: "Press 'Cancel' key after send 'Get Encrypted PIN' & 'Get Numeric' & 'Get Amount';
0x30FF: "Security Chip is not connect";
0x3000: "Security Chip is deactivation & Device is In Removal Legally State.";
0x3101: "Security Chip is activation &  Device is In Removal Legally State.";
0x5500: "No Admin DUKPT Key.";
0x5501: "Admin  DUKPT Key STOP.";
0x5502: "Admin DUKPT Key KSN is Error.";
0x5503: "Get Authentication Code1 Failed.";
0x5504: "Validate Authentication Code Error.";
0x5505: "Encrypt or Decrypt data failed.";
0x5506: "Not Support the New Key Type.";
0x5507: "New Key Index is Error.";
0x5508: "Step Error.";
0x5509: "KSN Error";
0x550A: "MAC Error.";
0x550B: "Key Usage Error.";
0x550C: "Mode Of Use Error.";
0x550F: "Other Error.";
0x6000: "Save or Config Failed / Or Read Config Error.";
0x6200: "No Serial Number.";
0x6900: "Invalid Command - Protocol is right, but task ID is invalid.";
0x6A01: "Unsupported Command - Protocol and task ID are right, but command is invalid - In this State";
0x6A00: "Unsupported Command - Protocol and task ID are right, but command is invalid.";
0x6B00: "Unknown parameter in command - Protocol task ID and command are right, but parameter is invalid.";
0x6C00: "Unknown parameter in command - Protocol task ID and command are right, but length is out of the
       requirement.";
0x7200: "Device is suspend (MKSK suspend or press password suspend).";
0x7300: "PIN DUKPT is STOP (21 bit 1).";
0x7400: "Device is Busy.";
0xE100: "Can not enter sleep mode";
0xE200: "File has existed";
0xE300: "File has not existed";
0xE313: "IO line low -- Card error after session start";
0xE400: "Open File Error";
0xE500: "SmartCard Error";
0xE600: "Get MSR Card data is error";
0xE700: "Command time out";
0xE800: "File read or write is error";
0xE900: "Active 1850 error!";
0xEA00: "Load bootloader error";
0xEF00: "Protocol Error- STX or ETX or check error.";
0xEB00: "Picture is not exist";
0x2C02: "No Microprocessor ICC seated";
0x2C06: "no card seated to request ATR";
0x2D01: "Card Not Supported,";
0x2D03: "Card Not Supported, wants CRC";
0x690D: "Command not supported on reader without ICC support";
```

```
0x8100: "ICC error time out on power-up";
0x8200: "invalid TS character received - Wrong operation step";
0x8300: "Decode MSR Error";
0x8400: "TriMagII no Response";
0x8500: "No Swipe MSR Card";
0x8510: "No Financial Card";
0x8600: "Unsupported F, D, or combination of F and D";
0x8700: "protocol not supported EMV TD1 out of range";
0x8800: "power not at proper level";
0x8900: "ATR length too long";
0x8B01: "EMV invalid TA1 byte value";
0x8B02: "EMV TB1 required";
0x8B03: "EMV Unsupported TB1 only 00 allowed";
0x8B04: "EMV Card Error, invalid BWI or CWI";
0x8B06: "EMV TB2 not allowed in ATR";
0x8B07: "EMV TC2 out of range";
0x8B08: "EMV TC2 out of range";
0x8B09: "per EMV96 TA3 must be > 0xF";
0x8B10: "ICC error on power-up";
0x8B11: "EMV T=1 then TB3 required";
0x8B12: "Card Error, invalid BWI or CWI";
0x8B13: "Card Error, invalid BWI or CWI";
0x8B17: "EMV TC1/TB3 conflict-";
0x8B20: "EMV TD2 out of range must be T=1";
0x8C00: "TCK error";
0xA304: "connector has no voltage setting";
0xA305: "ICC error on power-up invalid (SBLK(IFSD) exchange";
0xE301: "ICC error after session start";
0xFF00: "Request to go online";
0xFF01: "EMV: Accept the offline transaction";
0xFF02: "EMV: Decline the offline transaction";
0xFF03: "EMV: Accept the online transaction";
0xFF04: "EMV: Decline the online transaction";
0xFF05: "EMV: Application may fallback to magstripe technology";
0xFF06: "EMV: ICC detected tah the conditions of use are not satisfied";
0xFF07: "EMV: ICC didn't accept transaction";
0xFF08: "EMV: Transaction was cancelled";
0xFF09: "EMV: Application was not selected by kernel or ICC format error or ICC missing data error";
0xFF0A: "EMV: Transaction is terminated";
0xFF0B: "EMV: Other EMV Error";
0xFFFF: "NO RESPONSE";
0xF002: "ICC communication timeout";
0xF003: "ICC communication Error";
0xF00F: "ICC Card Seated and Highest Priority, disable MSR work request";
0xF200: "AID List / Application Data is not exist";
0xF201: "Terminal Data is not exist";
0xF202: "TLV format is error";
0xF203: "AID List is full";
0xF204: "Any CA Key is not exist";
0xF205: "CA Key RID is not exist";
0xF206: "CA Key Index it not exist";
0xF207: "CA Key is full";
0xF208: "CA Key Hash Value is Error";
0xF209: "Transaction  format error";
0xF20A: "The command will not be processing";
0xF20B: "CRL is not exist";
0xF20C: "CRL number  exceed max number";
0xF20D: "Amount,Other Amount,Trasaction Type  are  missing";
0xF20E: "The Identification of algorithm is mistake";
0xF20F: "No Financial Card";
0xF210: "In Encrypt Result state, TLV total Length is greater than Max Length";
0x1001: "INVALID ARG";
0x1002: "FILE_OPEN_FAILED";
0x1003: "FILE OPERATION_FAILED";
0x2001: "MEMORY_NOT_ENOUGH";
0x3002: "SMARTCARD_FAIL";
0x3003: "SMARTCARD_INIT_FAILED";
0x3004: "FALLBACK_SITUATION";
0x3005: "SMARTCARD_ABSENT";
0x3006: "SMARTCARD_TIMEOUT";
0x5001: "EMV_PARSING_TAGS_FAILED";
0x5002: "EMV_DUPLICATE_CARD_DATA_ELEMENT";
0x5003: "EMV_DATA_FORMAT_INCORRECT";
0x5004: "EMV_NO_TERM_APP";
0x5005: "EMV_NO_MATCHING_APP";
0x5006: "EMV_MISSING_MANDATORY_OBJECT";
0x5007: "EMV_APP_SELECTION_RETRY";
0x5008: "EMV_GET_AMOUNT_ERROR";
0x5009: "EMV_CARD_REJECTED";
0x5010: "EMV_AIP_NOT_RECEIVED";
0x5011: "EMV_AFL_NOT_RECEIVED";
0x5012: "EMV_AFL_LEN_OUT_OF_RANGE";
0x5013: "EMV_SFI_OUT_OF_RANGE";
0x5014: "EMV_AFL_INCORRECT";
0x5015: "EMV_EXP_DATE_INCORRECT";
0x5016: "EMV_EFF_DATE_INCORRECT";
0x5017: "EMV_ISS_COD_TBL_OUT_OF_RANGE";
```

```
0x5018: "EMV_CRYPTOGRAM_TYPE_INCORRECT";
0x5019: "EMV_PSE_NOT_SUPPORTED_BY_CARD";
0x5020: "EMV_USER_SELECTED_LANGUAGE";
0x5021: "EMV_SERVICE_NOT_ALLOWED";
0x5022: "EMV_NO_TAG_FOUND";
0x5023: "EMV_CARD_BLOCKED";
0x5024: "EMV_LEN_INCORRECT";
0x5025: "CARD_COM_ERROR";
0x5026: "EMV_TSC_NOT_INCREASED";
0x5027: "EMV_HASH_INCORRECT";
0x5028: "EMV_NO_ARC";
0x5029: "EMV_INVALID_ARC";
0x5030: "EMV_NO_ONLINE_COMM";
0x5031: "TRAN_TYPE_INCORRECT";
0x5032: "EMV_APP_NO_SUPPORT";
0x5033: "EMV_APP_NOT_SELECT";
0x5034: "EMV_LANG_NOT_SELECT";
0x5035: "EMV_NO_TERM_DATA";
0x6001: "CVM_TYPE_UNKNOWN";
0x6002: "CVM_AIP_NOT_SUPPORTED";
0x6003: "CVM_TAG_8E_MISSING";
0x6004: "CVM_TAG_8E_FORMAT_ERROR";
0x6005: "CVM_CODE_IS_NOT_SUPPORTED";
0x6006: "CVM_COND_CODE_IS_NOT_SUPPORTED";
0x6007: "NO_MORE_CVM";
0x6008: "PIN_BYPASSED_BEFORE";
0x7001: "PK_BUFFER_SIZE_TOO_BIG";
0x7002: "PK_FILE_WRITE_ERROR";
0x7003: "PK_HASH_ERROR";
0x8001: "NO_CARD_HOLDER_CONFIRMATION";
0x8002: "GET_ONLINE_PIN";
0xD000: "Data not exist";
0xD001: "Data access error";
0xD100: "RID not exist";
0xD101: "RID existed";
0xD102: "Index not exist";
0xD200: "Maximum exceeded";
0xD201: "Hash error";
0xD205: "System Busy";
0x0E01: "Unable to go online";
0x0E02: "Technical Issue";
0x0E03: "Declined";
0x0E04: "Issuer Referral transaction";
0x0F01: "Decline the online transaction";
0x0F02: "Request to go online";
0x0F03: "Transaction is terminated";
0x0F05: "Application was not selected by kernel or ICC format error or ICC missing data error";
0x0F07: "ICC didn't accept transaction";
0x0F0A: "Application may fallback to magstripe technology";
0x0F0C: "Transaction was cancelled";
0x0F0D: "Timeout";
0x0F0F: "Other EMV Error";
0x0F10: "Accept the offline transaction";
0x0F11: "Decline the offline transaction";
0x0F21: "ICC detected tah the conditions of use are not satisfied";
0x0F22: "No app were found on card matching terminal configuration";
0x0F23: "Terminal file does not exist";
0x0F24: "CAPK file does not exist";
0x0F25: "CRL Entry does not exist";
0x0FFE: "code when blocking is disabled";
0x0FFF: "code when command is not applicable on the selected device";
0xF005: "ICC Encrypted C-APDU Data Structure Length Error Or Format Error.";
0xBBE0: "CM100 Success";
0xBBE1: "CM100 Parameter Error";
0xBBE2: "CM100 Low Output Buffer";
0xBBE3: "CM100 Card Not Found";
0xBBE4: "CM100 Collision Card Exists";
0xBBE5: "CM100 Too Many Cards Exist";
0xBBE6: "CM100 Saved Data Does Not Exist";
0xBBE8: "CM100 No Data Available";
0xBBE9: "CM100 Invalid CID Returned";
0xBBEA: "CM100 Invalid Card Exists";
0xBBEC: "CM100 Command Unsupported";
0xBBED: "CM100 Error In Command Process";
0xBBEE: "CM100 Invalid Command";

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

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

0X9040: "Invalid Manufacturing system data";
0X9041: "Not authenticated";
0X9042: "Invalid Master DUKPT Key";
0X9043: "Invalid MAC Key";
```

```
0X9044: "Reserved for future use";
0X9045: "Reserved for future use";
0X9046: "Invalid DATA DUKPT Key";
0X9047: "Invalid PIN Pairing DUKPT Key";
0X9048: "Invalid DATA Pairing DUKPT Key";
0X9049: "No nonce generated";
0X9949: "No GUID available. Perform getVersion first.";
0X9950: "MAC Calculation unsuccessful. Check BDK value.";

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

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

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

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

Chapter 8

Enumeration Reference

Common

```
    enum DEVICE_TYPE{
        DEVICE_TYPE_UNKNOWN=0,
        IDT_DEVICE_AUGUSTA,
        AUGUSTA_KB,
        IDT_DEVICE_SPECTRUM_PRO,
        IDT_DEVICE_MINISMART_II,
        IDT_DEVICE_UNIPAY,
        IDT_DEVICE_UNIPAY_I_V,
        IDT_DEVICE_UNIPAY_III,
        IDT_DEVICE_L100,
    };


enum DEVICE_INTERFACE{
    DEVICE_INTERFACE_UNKNOWN=0,
    PROTOCOL_USBHID,
    PROTOCOL_USBKB,
    PROTOCOL_SEIRAL,
    PROTOCOL_BLUETOOTH,
};


enum DEVICE_PROTOCOL{
    DEVICE_PROTOCOL_UNKNOWN=0,
    PROTOCOL_NGA,
    PROTOCOL_IDG,
    PROTOCOL_ITP,
};


enum DEVICE_STATUS{
    DEVICE_DISCONNECT=0,
    DEVICE_CONNECTED,
};


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


enum MSR_callBack_type
{
    MSR_callBack_type_ERR=1,
    MSR_callBack_type_RETURN_CODE,
    MSR_callBack_type_TIMEOUT
};


enum PIN_callBack_type
{
    PIN_callBack_type_ERR=1,
    PIN_callBack_type_RETURN_CODE,
    PIN_callBack_type_TIMEOUT
};
```

```
enum CAPTURE_ENCODE_TYPE
{
    CAPTURE_ENCODE_TYPE_ISOABA,
    CAPTURE_ENCODE_TYPE_AAMVA,
    CAPTURE_ENCODE_TYPE_Other,
    CAPTURE_ENCODE_TYPE_Raw,
    CAPTURE_ENCODE_TYPE_Jis_II,
    CAPTURE_ENCODE_TYPE_Jis_I,
    CAPTURE_ENCODE_TYPE_Jis_II_Security,
    CAPTURE_ENCODE_TYPE_Contactless_Visa_Kernel1,
    CAPTURE_ENCODE_TYPE_Contactless_MasterCard,
    CAPTURE_ENCODE_TYPE_Contactless_Visa_Kernel3,
    CAPTURE_ENCODE_TYPE_Contactless_AmericanExpress,
    CAPTURE_ENCODE_TYPE_Contactless_JCB,
    CAPTURE_ENCODE_TYPE_Contactless_Discover,
    CAPTURE_ENCODE_TYPE_Contactless_UnionPay,
    CAPTURE_ENCODE_TYPE_Contactless_Others,
    CAPTURE_ENCODE_TYPE_Manual_Entry_Enhanced_Mode,
    CAPTURE_ENCODE_TYPE_JisI_II
};


enum CAPTURE_ENCRYPT_TYPE
{
    CAPTURE_ENCRYPT_TYPE_TDES, CAPTURE_ENCRYPT_TYPE_AES, CAPTURE_ENCRYPT_TYPE_NONE
};


enum EMV_PIN_MODE
{
    EMV_PIN_MODE_CANCEL=0, EMV_PIN_MODE_ONLINE_DUKPT=1, EMV_PIN_MODE_ONLINE_MKSK=2, EMV_PIN_MODE_OFFLINE=3,
};


enum EMV_CALLBACK_TYPE
{
    EMV_CALLBACK_TYPE_LCD=1, EMV_CALLBACK_TYPE_PINPAD=2, EMV_CALLBACK_MSR=3,
    EMV_callBack_type_ERR,
    EMV_callBack_type_RETURN_CODE,
};


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


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_TRANSACTION_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_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_MEMORY_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_FALLBACK_SITUATION = 0x3004,
        EMV_RESULT_CODE_SMARTCARD_ABSENT = 0x3005,
        EMV_RESULT_CODE_SMARTCARD_TIMEOUT = 0x3006,
        EMV_RESULT_CODE_MSR_CARD_ERROR = 0x3007,
        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_OBJECT_MISSING = 0x5006,
        EMV_RESULT_CODE_APP_SELECTION_RETRY = 0x5007,
        EMV_RESULT_CODE_AMOUNT_ERROR_GET = 0x5008,
        EMV_RESULT_CODE_CARD_REJECTED = 0x5009,
        EMV_RESULT_CODE_AIP_NOT_RECEIVED = 0x5010,
```

```
        EMV_RESULT_CODE_AFL_NOT_RECEIVED = 0x5011,
        EMV_RESULT_CODE_AFL_LEN_OUT_OF_RANGE = 0x5012,
        EMV_RESULT_CODE_SFI_OUT_OF_RANGE = 0x5013,
        EMV_RESULT_CODE_AFL_INCORRECT = 0x5014,
        EMV_RESULT_CODE_EXP_DATE_INCORRECT = 0x5015,
        EMV_RESULT_CODE_EFF_DATE_INCORRECT = 0x5016,
        EMV_RESULT_CODE_ISS_COD_TBL_OUT_OF_RANGE = 0x5017,
        EMV_RESULT_CODE_CRYPTOGRAM_TYPE_INCORRECT = 0x5018,
        EMV_RESULT_CODE_PSE_BY_CARD_NOT_SUPPORTED = 0x5019,
        EMV_RESULT_CODE_USER_LANGUAGE_SELECTED = 0x5020,
        EMV_RESULT_CODE_SERVICE_NOT_ALLOWED = 0x5021,
        EMV_RESULT_CODE_NO_TAG_FOUND = 0x5022,
        EMV_RESULT_CODE_CARD_BLOCKED = 0x5023,
        EMV_RESULT_CODE_LEN_INCORRECT = 0x5024,
        EMV_RESULT_CODE_CARD_COM_ERROR = 0x5025,
        EMV_RESULT_CODE_TSC_NOT_INCREASED = 0x5026,
        EMV_RESULT_CODE_HASH_INCORRECT = 0x5027,
        EMV_RESULT_CODE_ARC_NOT_PRESENCED = 0x5028,
        EMV_RESULT_CODE_ARC_INVALID = 0x5029,
        EMV_RESULT_CODE_COMM_NO_ONLINE = 0x5030,
        EMV_RESULT_CODE_TRAN_TYPE_INCORRECT = 0x5031,
        EMV_RESULT_CODE_APP_NO_SUPPORT = 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_CVM_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_RESULT_CODE_CVM_CODE_IS_NOT_SUPPORTED = 0x6005,
        EMV_RESULT_CODE_CVM_COND_CODE_IS_NOT_SUPPORTED = 0x6006,
        EMV_RESULT_CODE_CVM_NO_MORE = 0x6007,
        EMV_RESULT_CODE_PIN_BYPASSED_BEFORE = 0x6008,
        EMV_RESULT_CODE_UNKONWN = 0xffff,

    };


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
      ,
};


enum EVENT_NOTIFICATION_Types
{
    EVENT_NOTIFICATION_UNKNOWN,
    EVENT_NOTIFICATION_ICC_Card_Not_Seated,
    EVENT_NOTIFICATION_ICC_Card_Seated,
    EVENT_NOTIFICATION_MSR_Swipe_Card,
};


enum CTLS_APPLICATION
    {
        CTLS_APPLICATION_NONE = 0,
        CTLS_APPLICATION_MASTERCARD = 1,
        CTLS_APPLICATION_VISA = 2,
        CTLS_APPLICATION_AMEX = 3,
        CTLS_APPLICATION_DISCOVER = 4,
        CTLS_APPLICATION_SPEEDPASS = 5,
        CTLS_APPLICATION_GIFT_CARD = 6,
        CTLS_APPLICATION_DINERS_CLUB = 7,
        CTLS_APPLICATION_EN_ROUTE = 8,
        CTLS_APPLICATION_JCB = 9,
        CTLS_APPLICATION_VIVO_DIAGNOSTIC = 10,
        CTLS_APPLICATION_HID = 11,
        CTLS_APPLICATION_MSR_SWIPE = 12,
        CTLS_APPLICATION_RESERVED = 13,
        CTLS_APPLICATION_DES_FIRE_TRACK_DATA = 14,
        CTLS_APPLICATION_DES_FIRE_RAW_DATA = 15,
        CTLS_APPLICATION_RBS = 17,
        CTLS_APPLICATION_VIVO_COMM = 20,
    };


enum DeviceState
   {
       TransactionData,
       Connected,
       ConnectionFailed,
       DataReceived,
       DataSent,
       Disconnected,
       SwipeCanceled,
```

```
        ToConnect,
        ToSwipe,
        ToTap,
        CommandTimeout,
        SwipeTimeout,
        DeviceTimeout,
        EMVCallback,
        TransactionCancelled,
        TransactionFailed,
        MSRDecodeError,
        DefaultDeviceTypeChange,
        Notification,
        PINpadKeypress,
        PINCancelled,
        PINTimeout

    };

enum RETURN_CODE
 {
      RETURN_CODE_DO_SUCCESS = 0,
      RETURN_CODE_ERR_DISCONNECT,
      RETURN_CODE_ERR_CMD_RESPONSE,
      RETURN_CODE_ERR_TIMEDOUT,
      RETURN_CODE_ERR_INVALID_PARAMETER,
      RETURN_CODE_SDK_BUSY_MSR,
      RETURN_CODE_SDK_BUSY_PINPAD,
      RETURN_CODE_SDK_BUSY_CTLS,
      RETURN_CODE_SDK_BUSY_EMV,
      RETURN_CODE_ERR_OTHER,
      RETURN_CODE_FAILED,
      RETURN_CODE_NOT_ATTACHED,
      RETURN_CODE_MONO_AUDIO,
      RETURN_CODE_CONNECTED,
      RETURN_CODE_LOW_VOLUME,
      RETURN_CODE_CANCELED,
      RETURN_CODE_INVALID_STR,
      RETURN_CODE_NO_FILE,
      RETURN_CODE_INVALID_FILE,
      RETURN_CODE_HOST_UNREACHABLE,
      RETURN_CODE_RKI_FAILURE,
      RETURN_CODE_MISSING_DLL,
      RETURN_CODE_ERR_BUF_NOT_ENOUGH,
      RETURN_CODE_P1_INCORRECT_FRAME_TAG = 0X0C01,
      RETURN_CODE_P1_INCORRECT_FRAME_TYPE = 0X0C02,
      RETURN_CODE_P1_UNKNOWN_FRAME_TYPE = 0X0C03,
      RETURN_CODE_P1_UNKNOWN_COMMAND = 0X0C04,
      RETURN_CODE_P1_UNKNOWN_SUB_COMMAND = 0X0C05,
      RETURN_CODE_P1_CRC_ERROR = 0X0C06,
      RETURN_CODE_P1_FAILED = 0X0C07,
      RETURN_CODE_P1_TIMEOUT = 0X0C08,
      RETURN_CODE_P1_INCORRECT_PARAMETER = 0X0C0A,
      RETURN_CODE_P1_COMMAND_NOT_SUPPORTED = 0X0C0B,
      RETURN_CODE_P1_SUB_COMMAND_NOT_SUPPORTED = 0X0C0C,
      RETURN_CODE_P1_STATUS_ABORT_COMMAND = 0X0C0D,
      RETURN_CODE_P1_COMMAND_NOT_ALLOWED = 0X0C0F,
      RETURN_CODE_P2_ = 0X0D01,
      RETURN_CODE_P2_UNKNOWN_COMMAND = 0X0D02,
      RETURN_CODE_P2_UNKNOWN_SUB_COMMAND = 0X0D03,
      RETURN_CODE_P2_CRC_ERROR = 0X0D04,
      RETURN_CODE_P2_INCORRECT_PARAMETER = 0X0D05,
      RETURN_CODE_P2_PARAMETER_NOT_SUPPORTED = 0X0D06,
      RETURN_CODE_P2_MAL_FORMATTED_DATA = 0X0D07,
      RETURN_CODE_P2_TIMEOUT = 0X0D08,
      RETURN_CODE_P2_FAILED = 0X0D0A,
      RETURN_CODE_P2_COMMAND_NOT_ALLOWED = 0X0D0B,
      RETURN_CODE_P2_SUB_COMMAND_NOT_ALLOWED = 0X0D0C,
      RETURN_CODE_P2_BUFFER_OVERFLOW = 0X0D0D,
      RETURN_CODE_P2_USER_INTERFACE_EVENT = 0X0D0E,
      RETURN_CODE_P2_COMM_TYPE_NOT_SUPPORTED = 0X0D11,
      RETURN_CODE_P2_SECURE_INTERFACE_NOT_FUNCTIONAL = 0X0D12,
      RETURN_CODE_P2_DATA_FIELD_NOT_MOD_8 = 0X0D13,
      RETURN_CODE_P2_PADDING_UNEXPECTED = 0X0D14,
      RETURN_CODE_P2_KEY_TYPE_INVALID = 0X0D15,
      RETURN_CODE_P2_COULD_NOT_RETRIEVE_KEY = 0X0D16,
      RETURN_CODE_P2_HASH_CODE_ERROR = 0X0D17,
      RETURN_CODE_P2_COUND_NOT_STORE_KEY = 0X0D18,
      RETURN_CODE_P2_FRAME_TOO_LARGE = 0X0D19,
      RETURN_CODE_P2_RESEND_INITSECURECOMM_COMMAND = 0X0D1A,
      RETURN_CODE_P2_EEPROM_NOT_INITIALIZED = 0X0D1B,
      RETURN_CODE_P2_APDU_ENCODING_ERROR = 0X0D1C,
      RETURN_CODE_P2_SAM_COMM_ERROR = 0X0D20,
      RETURN_CODE_P2_SEQUENCE_COUNTER_ERROR = 0X0D21,
      RETURN_CODE_P2_IMPROPER_BITMAP = 0X0D22,
      RETURN_CODE_P2_REQUEST_ONLINE_AUTHORIZATION = 0X0D23,
      RETURN_CODE_P2_RAW_DATA_READ_SUCCESSFUL = 0X0D24,
      RETURN_CODE_P2_MESSAGE_INDEX_NOT_AVAILABLE = 0X0D25,
```

```
RETURN_CODE_P2_VERSION_INFORMATION_MISMATCH = 0X0D26,
RETURN_CODE_P2_INCORRECT_MESSAGE_INDEX = 0X0D27,
RETURN_CODE_P2_NEXT_MESSAGE_NOT_RECEIVED = 0X0D28,
RETURN_CODE_P2_ILM_LANGUAGE_NOT_AVAILABLE = 0X0D29,
RETURN_CODE_P2_OTHER_LANGUAGE_NOT_SUPPORTED = 0X0D2A,
RETURN_CODE_UNKNOWN_ERROR_FROM_SAM = 0X0D41,
RETURN_CODE_INVALID_DATA_DETECTED_BY_SAM = 0X0D42,
RETURN_CODE_INCOMPLETE_DATA_DETECTED_BY_SAM = 0X0D43,
RETURN_CODE_RESERVED = 0X0D44,
RETURN_CODE_INVALID_KEY_HASH_ALGORITHM = 0X0D45,
RETURN_CODE_INVALID_KEY_ENCRYPTION_ALRORITHM = 0X0D46,
RETURN_CODE_INVALID_MODULUS_LENGTH = 0X0D47,
RETURN_CODE_INVALID_EXPONENT = 0X0D48,
RETURN_CODE_KEY_ALREADY_EXISTS = 0X0D49,
RETURN_CODE_NO_SPACE_FOR_NEW_RID = 0X0D4A,
RETURN_CODE_KEY_NOT_FOUND = 0X0D4B,
RETURN_CODE_CRYPTO_NOT_RESPONDING = 0X0D4C,
RETURN_CODE_CRYPTO_COMMUNICATION_ERROR = 0X0D4D,
RETURN_CODE_P2_KEY_MANAGER_ERROR_4E = 0X0D4E,
RETURN_CODE_ALL_KEY_SLOTS_FULL = 0X0D4F,
RETURN_CODE_P2_AUTO_SWITCH_OK = 0X0D50,
RETURN_CODE_P2_AUTO_SWITCH_FAILED = 0X0D51,
RETURN_CODE_P2_DATA_DOES_NOT_EXIST = 0X0D60,
RETURN_CODE_P2_DATA_FULL = 0X0D61,
RETURN_CODE_P2_WRITE_FLASH_ERROR = 0X0D62,
RETURN_CODE_P2_OK_AND_HAVE_NEXT_COMMAND = 0X0D63,
RETURN_CODE_P2_ACCOUNT_DUKPT_KWY_NOT_EXIST = 0X0D90,
RETURN_CODE_P2_ACCOUNT_DUKPT_KEY_KSN_EXHAUSTED = 0X0D91,

RETURN_CODE_UNKNOWN_COMMAND = 0X9031, // Unknown command
RETURN_CODE_WRONG_PARAMETER = 0X9032, // Wrong parameter (such as the length of the command is
  incorrect)

RETURN_CODE_WAIT = 0X9038,
RETURN_CODE_BUSY = 0X9039,
RETURN_CODE_RETRIES_OVER_LIMIT = 0X903A,
RETURN_CODE_TIMEOUT = 0X8100,

RETURN_CODE_INVALID_MAN_SYSTEM_DATA = 0X9040,
RETURN_CODE_NOT_AUTHENTICATED = 0X9041,
RETURN_CODE_INVALID_MASTER_DUKPT_KEY = 0X9042,
RETURN_CODE_INVALID_MAC_KEY = 0X9043,
RETURN_CODE_RESERVED_FOR_FUTURE_USE = 0X9044,
RETURN_CODE_RESERVED_FOR_FUTURE_USE_ = 0X9045,
RETURN_CODE_INVALID_DATA_DUKPT_KEY = 0X9046,
RETURN_CODE_INVALID_PIN_PARING_DUKPT_KEY = 0X9047,
RETURN_CODE_INVALID_DATA_PAIRING_DUKPT_KEY = 0X9048,
RETURN_CODE_NO_NONCE_GENERATED = 0X9049,
RETURN_CODE_NO_GUID_AVAILABLE = 0X9949,
RETURN_CODE_NO_MAC_CALCULATION = 0X9950,

RETURN_CODE_NOT_READY = 0X904A,
RETURN_CODE_MSR_DATA_FAILED = 0X904B,

RETURN_CODE_INVALID_CERTIFICATE = 0X9050,
RETURN_CODE_DUPLICATE_KEY_DETECTED = 0X9051,
RETURN_CODE_AT_CHECKS_FAILED = 0X9052,
RETURN_CODE_TR34_CHECKS_FAILED = 0X9053,
RETURN_CODE_TR31_CHECKS_FAILED = 0X9054,
RETURN_CODE_AMC_CHECKS_FAILED = 0X9055,
RETURN_CODE_FIRMWARE_DOWNLOAD_FAILED = 0X9056,

RETURN_CODE_LOG_IS_FULL = 0X9060,
RETURN_CODE_REMOVAL_SENSOR_UNENGAGED = 0X9061,
RETURN_CODE_HARDWARE_PROBLEM = 0X9062,

RETURN_CODE_ICC_COMMUNICATION_TIMEOUT = 0X9070,
RETURN_CODE_IFF_DATA_ERROR = 0X9071,
RETURN_CODE_SMART_CARD_NOT_POWERED_UP = 0X9072,
RETURN_CODE_NO_AID = 0xF200,
RETURN_CODE_NO_TERMINAL_DATA = 0xF201,
RETURN_CODE_WRONG_TLV_FORMAT = 0xF202,
RETURN_CODE_AID_LIST_FULL = 0xF203,
RETURN_CODE_NO_CA_KEY = 0xF204,
RETURN_CODE_NO_CA_KEY_RID = 0xF205,
RETURN_CODE_NO_CA_KEY_INDEX = 0xF206,
RETURN_CODE_CA_KEY_LIST_FULL = 0xF207,
RETURN_CODE_CA_KEY_HASH_ERROR = 0xF208,
RETURN_CODE_COMMAND_FORMAT_ERROR = 0xF209,
RETURN_CODE_UNEXPECTED_COMMAND = 0xF20A,
RETURN_CODE_NO_CRL = 0xF20B,
RETURN_CODE_CRL_LIST_FULL = 0xF20C,
RETURN_CODE_MISSING_REQUIRED_PARAMETERS = 0xF20D,
RETURN_CODE_CA_INCORRECT_HASH_ALGORITHM = 0xF20E,


RETURN_CODE_EMV_AUTHORIZATION_ACCEPTED = 0x0E00,
```

```
    RETURN_CODE_EMV_AUTHORIZATION_UNABLE_TO_GO_ONLINE = 0x0E01,
    RETURN_CODE_EMV_AUTHORIZATION_TECHNICAL_ISSUE = 0x0E02,
    RETURN_CODE_EMV_AUTHORIZATION_DECLINED = 0x0E03,
    RETURN_CODE_EMV_AUTHORIZATION_ISSUER_REFERRAL = 0x0E04,
    RETURN_CODE_EMV_APPROVED = 0x0F00,
    RETURN_CODE_EMV_DECLINED = 0x0F01,
    RETURN_CODE_EMV_GO_ONLINE = 0x0F02,
    RETURN_CODE_EMV_FAILED = 0x0F03,
    RETURN_CODE_EMV_SYSTEM_ERROR = 0x0F05,
    RETURN_CODE_EMV_NOT_ACCEPTED = 0x0F07,
    RETURN_CODE_EMV_FALLBACK = 0x0F0A,
    RETURN_CODE_EMV_CANCEL = 0x0F0C,
    RETURN_CODE_EMV_TIMEOUT = 0x0F0D,
    RETURN_CODE_EMV_OTHER_ERROR = 0x0F0F,
    RETURN_CODE_EMV_OFFLINE_APPROVED = 0x0F10,
    RETURN_CODE_EMV_OFFLINE_DECLINED = 0x0F11,
    RETURN_CODE_EMV_NEW_SELECTION = 0x0F21,
    RETURN_CODE_EMV_NO_AVAILABLE_APPS = 0x0F22,
    RETURN_CODE_EMV_NO_TERMINAL_FILE = 0x0F23,
    RETURN_CODE_EMV_NO_CAPK_FILE = 0x0F24,
    RETURN_CODE_EMV_NO_CRL_ENTRY = 0x0F25,
    RETURN_CODE_BLOCKING_DISABLED = 0x0FFE,
    RETURN_CODE_CM100_WITHOUT_ERROR = 0xBBE0,
    RETURN_CODE_CM100_PARAMETER =  0xBBE1,
    RETURN_CODE_CM100_LOWOUTBUFFER =  0xBBE2,
    RETURN_CODE_CM100_CARD_NOT_FOUND =  0xBBE3,
    RETURN_CODE_CM100_COLLISION_CARD_EXIST = 0xBBE4,
    RETURN_CODE_CM100_TOO_MANY_CARDS_EXIST = 0xBBE5,
    RETURN_CODE_CM100_SAVED_DATA_NOT_EXIST =  0xBBE6,
    RETURN_CODE_CM100_NO_DATA_AVAILABLE =  0xBBE8,
    RETURN_CODE_CM100_INVALID_CID_RETURNED =  0xBBE9,
    RETURN_CODE_CM100_INVALID_CARD_EXIST = 0xBBEA,
    RETURN_CODE_CM100_COMMAND_UNSUPPORTED =  0xBBEC,
    RETURN_CODE_CM100_COMMAND_PROCESS = 0xBBED,
    RETURN_CODE_CM100_INVALID_COMMAND =  0xBBEE,
    RETURN_CODE_COMMAND_UNAVAILABLE = 0x0FFF
};
```

Chapter 9

EMV Callback

During an EMV transaction, ID TECH devices without a built-in LCD display return LCD Display messages as an EMV Callback.

```
void EMV_callBack(int device_type, int device_state, unsigned char * data, int dataLen,IDTTransactionData*
      cardData,EMV_Callback* emvCallback, int transactionResultCode)
```

There is an **DeviceState** enum in the SDK's EMV Callback that is returned for **device_state**. When this **DeviceState** is received as **EMVCallback**, it returns an **EMV_Callback()** class pointer. **EMV_Callback.callbackType** specifies the type of callback: **EMV_CALLBACK_TYPE_LCD**, **EMV_CALLBACK_TYPE_PINPAD**, or **EMV_CALLBACK_MSR**. The device only uses the **EMV_CALLBACK_TYPE_LCD**.

The callback type for LCD display messages is **EMV_CALLBACK_TYPE_LCD**. To determine the type of LCD message, 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

The **LCD_DISPLAY_MODE_MESSAGE** and **LCD_DISPLAY_MODE_CLEAR_SCREEN** 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 the EMV transaction pauses until a response is sent to **emv_callbackResponseLCD()**.

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

Chapter 10

EMV Tag Reference

Tag	Description
42	Issuer Identification Number (IIN)
4F	Application Identifier (ADF Name)
50	Application Label
52	Command to perform
56	Track 1 Data
57	Track 2 Equivalent Data
5A	Application Primary Account Number (PAN)
5D	Deleted (see 9D)
5F20	Cardholder Name
5F24	Application Expiration Date
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)
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
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
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
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)
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
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	SmartTap Delimiter
DFEF1E	Encrypted Sensitive Tags
DFEF1F	Auto Authenticate
DFEF20	MAC option in reponse data
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
DFEF4A	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
DFEF5B	Mask for Tag5A
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
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
FFEE1E	Group 0 Initialize Flag
FFEE1F	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 11

File Index

11.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 12

File Documentation

12.1 Source_C/libIDT_Augusta.h File Reference

Augusta API.

```
#include "IDTDef.h"
```

Macros

- #define IN
- #define OUT
- #define IN_OUT

Typedefs

- typedef void(* pMessageHotplug)(int, int)
- typedef void(* pSendDataLog)(unsigned char *, int)
- typedef void(* pReadDataLog)(unsigned char *, int)
- typedef void(* pEMV_callBack)(int, int, unsigned char *, int, IDTTransactionData *, EMV_Callback *, int)
- typedef void(* pMSR_callBack)(int, IDTMSRData)
- typedef void(* pMSR_callBackp)(int, IDTMSRData *)
- typedef void(* pPIN_callBack)(int, IDTPINData *)
- typedef void(* pCMR_callBack)(int, IDTCMRData *)
- typedef void(* pCSFS_callBack)(BYTE status)
- typedef void(* pFW_callBack)(int, int, int, int, int)
- typedef void(* ftpComm_callBack)(int, int, int)
- typedef void(* httpComm_callBack)(BYTE *, int)
- typedef void(* v4Comm_callBack)(BYTE, BYTE, BYTE *, int)

Functions

- void registerHotplugCallBk (pMessageHotplug pMsgHotplug)
- void registerLogCallBk (pSendDataLog pFSend, pReadDataLog pFRead)
- void emv_registerCallBk (pEMV_callBack pEMVf)
- void msr_registerCallBk (pMSR_callBack pMSRf)
- void msr_registerCallBkp (pMSR_callBackp pMSRf)
- void pin_registerCallBk (pPIN_callBack pPINf)
- void device_registerCameraCallBk (pCMR_callBack pCMRf)

- void device_registerCardStatusFrontSwitchCallBk (pCSFS_callBack pCSFSf)
- void device_registerFWCallBk (pFW_callBack pFWf)
- char * SDK_Version ()
- int setAbsoluteLibraryPath (const char *absoluteLibraryPath)
- int device_init ()
- int device_setCurrentDevice (int deviceType)
- int device_close ()
- void device_getResponseCodeString (IN int returnCode, OUT char *despcrition)
- int device_isConnected ()
- int device_isAttached (int deviceType)
- int device_getFirmwareVersion (OUT char *firmwareVersion)
- int device_getFirmwareVersion_Len (OUT char *firmwareVersion, IN_OUT int *firmwareVersionLen)
- int device_getCurrentDeviceType ()
- int device_SendDataCommand (IN BYTE *cmd, IN int cmdLen, IN BYTE *data, IN int dataLen, OUT BYTE *response, IN_OUT int *respLen)
- int device_updateFirmware (IN BYTE *firmwareData, IN int firmwareDataLen, IN char *firmwareName, IN int encryptionType, IN BYTE *keyBlob, IN int keyBlobLen)
- int device_rebootDevice ()
- int device_controlLED (byte indexLED, byte control, int intervalOn, int intervalOff)
- int device_controlLED_ICC (int controlMode, int interval)
- int device_controlLED_MSR (byte control, int intervalOn, int intervalOff)
- int device_controlBeep (int index, int frequency, int duration)
- int device_getDRS (BYTE *codeDRS, int *codeDRSLen)
- int device_getKeyStatus (int *newFormat, BYTE *status, int *statusLen)
- int device_getSDKWaitTime ()
- void device_setSDKWaitTime (int waitTime)
- int device_getThreadStackSize ()
- void device_setThreadStackSize (int threadSize)
- int config_getModelNumber (OUT char *sNumber)
- int config_getModelNumber_Len (OUT char *sNumber, IN_OUT int *sNumberLen)
- int config_getSerialNumber (OUT char *sNumber)
- int config_getSerialNumber_Len (OUT char *sNumber, IN_OUT int *sNumberLen)
- int config_setLEDController (int firmwareControlMSRLED, int firmwareControlICCLED)
- int config_getLEDController (int *firmwareControlMSRLED, int *firmwareControlICCLED)
- int config_setBeeperController (int firmwareControlBeeper)
- int config_getBeeperController (int *firmwareControlBeeper)
- int config_setEncryptionControl (int msr, int icc)
- int config_getEncryptionControl (int *msr, int *icc)
- int icc_enable (IN int withNotification)
- int icc_disable ()
- int icc_powerOnICC (OUT BYTE *ATR, IN_OUT int *inLen)
- int icc_powerOffICC ()
- int icc_exchangeAPDU (IN BYTE *c_APDU, IN int cLen, OUT BYTE *reData, IN_OUT int *reLen)
- int icc_exchangeEncryptedAPDU (IN BYTE *c_APDU, IN int cLen, OUT BYTE *reData, IN_OUT int *reLen)
- int icc_getAPDU_KSN (OUT BYTE *KSN, IN_OUT int *inLen)
- int icc_getFunctionStatus (OUT int *enabled, OUT int *withNotification)
- int icc_getICCReaderStatus (OUT BYTE *status)
- int icc_getKeyFormatForICCDUKPT (OUT BYTE *format)
- int icc_getKeyTypeForICCDUKPT (OUT BYTE *type)
- int emv_getEMVKernelVersion (OUT char *version)
- int emv_getEMVKernelVersion_Len (OUT char *version, IN_OUT int *versionLen)
- int emv_getEMVKernelCheckValue (OUT BYTE *checkValue, IN_OUT int *checkValueLen)
- int emv_getEMVConfigurationCheckValue (OUT BYTE *checkValue, IN_OUT int *checkValueLen)
- void emv_allowFallback (IN int allow)
- void emv_setAutoAuthenticateTransaction (IN int authenticate)

- void emv_setAutoCompleteTransaction (IN int complete)
- int emv_getAutoAuthenticateTransaction ()
- int emv_getAutoCompleteTransaction ()
- int emv_startTransaction (IN double amount, IN double amtOther, IN int exponent, IN int type, IN int timeout, IN BYTE *tags, IN int tagsLen, IN int forceOnline)
- int emv_activateTransaction (IN int timeout, IN BYTE *tags, IN int tagsLen, IN int forceOnline)
- int emv_authenticateTransaction (IN BYTE *updatedTLV, IN int updatedTLVLen)
- int emv_authenticateTransactionWithTimeout (IN int timeout, IN BYTE *updatedTLV, IN int updatedTLVLen)
- int emv_completeTransaction (IN int commError, IN BYTE *authCode, IN int authCodeLen, IN BYTE *iad, IN int iadLen, IN BYTE *tlvScripts, IN int tlvScriptsLen, IN BYTE *tlv, IN int tlvLen)
- int emv_cancelTransaction ()
- int emv_retrieveTransactionResult (IN BYTE *tags, IN int tagsLen, IDTTransactionData *cardData)
- int emv_callbackResponseLCD (IN int type, byte selection)
- int emv_callbackResponseMSR (IN BYTE *MSR, IN_OUT int MSRLen)
- int emv_retrieveApplicationData (IN BYTE *AID, IN int AIDLen, OUT BYTE *tlv, IN_OUT int *tlvLen)
- int emv_setApplicationData (IN BYTE *name, IN int nameLen, IN BYTE *tlv, IN int tlvLen)
- int emv_removeApplicationData (IN BYTE *AID, IN int AIDLen)
- int emv_removeAllApplicationData ()
- int emv_retrieveAIDList (OUT BYTE *AIDList, IN_OUT int *AIDListLen)
- int emv_retrieveTerminalData (OUT BYTE *tlv, IN_OUT int *tlvLen)
- int emv_setTerminalData (IN BYTE *tlv, IN int tlvLen)
- int emv_removeTerminalData ()
- int emv_retrieveCAPK (IN BYTE *capk, IN int capkLen, OUT BYTE *key, IN_OUT int *keyLen)
- int emv_setCAPK (IN BYTE *capk, IN int capkLen)
- int emv_removeCAPK (IN BYTE *capk, IN int capkLen)
- int emv_removeAllCAPK ()
- int emv_retrieveCAPKList (OUT BYTE *keys, IN_OUT int *keysLen)
- int emv_retrieveTerminalID (OUT char *terminalID)
- int emv_retrieveTerminalID_Len (OUT char *terminalID, IN_OUT int *terminalIDLen)
- int emv_setTerminalID (IN char *terminalID)
- int emv_retrieveCRL (OUT BYTE *list, IN_OUT int *lssLen)
- int emv_setCRL (IN BYTE *list, IN int lsLen)
- int emv_removeCRL (IN BYTE *list, IN int lsLen)
- int emv_removeAllCRL ()
- int msr_getMSRData (OUT BYTE *reData, IN_OUT int *reLen)
- int msr_cancelMSRSwipe ()
- int msr_startMSRSwipe (IN int _timeout)
- void parseMSRData (IN BYTE *resData, IN int resLen, IN_OUT IDTMSRData *cardData)
- int msr_getKeyFormatForICCDUKPT (OUT BYTE *format)
- int msr_getKeyTypeForICCDUKPT (OUT BYTE *type)
- int msr_setKeyFormatForICCDUKPT (IN BYTE format)
- int msr_setKeyTypeForICCDUKPT (IN BYTE type)
- int msr_captureMode (int isBufferMode, int withNotification)
- int msr_setSetting (BYTE setting, BYTE *val, int valLen)
- int msr_getSetting (byte setting, BYTE *value, int *valueLen)
- int msr_setSwipeForcedEncryptionOption (int track1, int track2, int track3, int track3card0)
- int msr_getSwipeForcedEncryptionOption (BYTE *option)
- int msr_setSwipeMaskOption (int track1, int track2, int track3)
- int msr_getSwipeMaskOption (BYTE *option)
- int msr_setExpirationMask (int mask)
- int msr_getExpirationMask (BYTE *value)
- int msr_setClearPANID (BYTE val)
- int msr_getClearPANID (BYTE *value)
- int msr_disable ()
- int pin_cancelPINEntry ()

12.1.1 Detailed Description

Augusta API. Augusta Global API methods.

12.1.2 Macro Definition Documentation

12.1.2.1 #define IN

INPUT parameter.

12.1.2.2 #define IN_OUT

INPUT / OUTPUT PARAMETER.

12.1.2.3 #define OUT

OUTPUT parameter.

12.1.3 Typedef Documentation

12.1.3.1 typedef void(∗ ftpComm_callBack)(int, int, int)

Define the comm callback function to get FTP file transfer status

It should be passed as a parameter in a FTP request, Signature (int, int, int) = response code, current block, total blocks RESPONSE CODES: 100 = FILE DOWNLOAD STARTED 101 = FILE BLOCK XX OF XX RECEIVED 102 = FILE DOWNLOAD COMPLETED 103 = FILE DOWNLOAD TERMINATED PREMATURELY

12.1.3.2 typedef void(∗ httpComm_callBack)(BYTE ∗, int)

Define the comm callback function to get the async url data

It should be registered using the comm_registerHTTPCallback

12.1.3.3 typedef void(∗ pCMR_callBack)(int, IDTCMRData ∗)

Define the camera callback function to get the image data

It should be registered using the device_registerCameraCallBk,

12.1.3.4 typedef void(∗ pCSFS_callBack)(BYTE status)

Define the card status and front switch callback function to get card and front switch status

It should be registered using the device_registerCardStatusFrontSwitchCallBk,

12.1.3.5 typedef void(∗ pEMV_callBack)(int, int, unsigned char ∗, int, IDTTransactionData ∗, EMV_Callback ∗, int)

Define the EMV callback function to get the transaction message/data/result.

It should be registered using the emv_registerCallBk,

12.1.3.6 typedef void(* pFW_callBack)(int, int, int, int, int)

Define the firmware update callback function to get the status of firmware update

It should be registered using the device_registerFWCallBk,

12.1.3.7 typedef void(* pMessageHotplug)(int, int)

Define the USB hot-plug callback function to monitor the info when plug in/out the reader.

It should be registered using the registerHotplugCallBk, The first integer parameter is device type, and the second integer parameter is either 0: Device Plugged Out or 1: Device Plugged In

12.1.3.8 typedef void(* pMSR_callBack)(int, IDTMSRData)

Define the MSR callback function to get the MSR card data

It should be registered using the msr_registerCallBk, this callback function is for backward compatibility

12.1.3.9 typedef void(* pMSR_callBackp)(int, IDTMSRData *)

Define the MSR callback function to get pointer to the MSR card data

It should be registered using the msr_registerCallBk, this callback function is recommended instead of pMSR_call-Back

12.1.3.10 typedef void(* pPIN_callBack)(int, IDTPINData *)

Define the PINPad callback function to get the input PIN Pad data

It should be registered using the pin_registerCallBk,

12.1.3.11 typedef void(* pReadDataLog)(unsigned char *, int)

Define the read response callback function to monitor the reading response from the reader.

It should be registered using the registerLogCallBk,

12.1.3.12 typedef void(* pSendDataLog)(unsigned char *, int)

Define the send command callback function to monitor the sending command into the reader.

It should be registered using the registerLogCallBk,

12.1.3.13 typedef void(* v4Comm_callBack)(BYTE, BYTE, BYTE *, int)

Define the comm callback function to receive the V4 Protocol packets received by the device from an external source (IP/USB/RS-232) It should be registered using the comm_registerV4Callback, Data callback will contain command, sub-command, and data from V4 packet

12.1.4 Function Documentation

12.1.4.1 int config_getBeeperController (int * *firmwareControlBeeper*)

Get the Beeper Controller Status Set the Beeper controlled Status by software or firmware

Parameters

firmwareControl-Beeper	1 means firmware control the beeper, 0 means software control beeper.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.1.4.2 int config_getEncryptionControl (int * *msr,* int * *icc*)

Get Encryption Control

Get Encryption Control to switch status between MSR and ICC/EMV function. Following Encryption status supported:

- MSR ON, ICC/EMV ON,
- MSR ON, ICC/EMV OFF,
- MSR OFF, ICC/EMV OFF,

Parameters

msr	• 1: enabled MSR with Encryption, • 0: disabled MSR with Encryption,
icc	• 1: enabled ICC with Encryption, • 0: disabled ICC with Encryption,

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.1.4.3 int config_getLEDController (int * *firmwareControlMSRLED,* int * *firmwareControlICCLED*)

Get the LED Controller Status Get the MSR / ICC LED controlled status by software or firmware NOTE: The ICC LED always controlled by software.

Parameters

firmwareControl-MSRLED	• 1: firmware control the MSR LED • 0: software control the MSR LED

firmwareControl-ICCLED	• 1: firmware control the ICC LED • 0: software control the ICC LED

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.1.4.4 int config_getModelNumber (OUT char * *sNumber*)

Polls device for Model Number

Parameters

sNumber	Returns Model Number; needs to have at least 64 bytes of memory

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.1.4.5 int config_getModelNumber_Len (OUT char * *sNumber,* IN_OUT int * *sNumberLen*)

DEPRECATED : please use config_getModelNumber_Len(OUT char* sNumber, IN_OUT int *sNumberLen)

Polls device for Model Number

Parameters

sNumber	Returns Model Number
sNumber	length of Model Number

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.1.4.6 int config_getSerialNumber (OUT char * *sNumber*)

DEPRECATED : please use config_getSerialNumber_Len(OUT char* sNumber, IN_OUT int *sNumberLen)

Polls device for Serial Number

Parameters

sNumber	Returns Serial Number; needs to have at least 64 bytes of memory

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.1.4.7 int config_getSerialNumber_Len (OUT char * *sNumber,* IN_OUT int * *sNumberLen*)

Polls device for Serial Number

Parameters

sNumber	Returns Serial Number
sNumberLen	Length of Serial Number

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.1.4.8 int config_setBeeperController (int *firmwareControlBeeper*)

Set the Beeper Controller Set the Beeper controlled by software or firmware

Parameters

firmwareControl-Beeper	1 means firmware control the beeper, 0 means software control beeper.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.1.4.9 int config_setEncryptionControl (int *msr,* int *icc*)

Set Encryption Control

Set Encryption Control to switch status between MSR and ICC/EMV function. Following Encryption status supported:

- MSR ON, ICC/EMV ON,
- MSR ON, ICC/EMV OFF,
- MSR OFF, ICC/EMV OFF,

Parameters

msr	• 1: enable MSR with Encryption, • 0: disable MSR with Encryption,
icc	• 1: enable ICC with Encryption, • 0: disable ICC with Encryption,

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.1.4.10 int config_setLEDController (int *firmwareControlMSRLED,* int *firmwareControlICCLED*)

Set the LED Controller Set the MSR / ICC LED controlled by software or firmware NOTE: The ICC LED always controlled by software.

Parameters

firmwareControl-MSRLED	• 1: firmware control the MSR LED • 0: software control the MSR LED
firmwareControl-ICCLED	• 1: firmware control the ICC LED • 0: software control the ICC LED

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.1.4.11 int device_close ( )

Close the device

Returns

RETURN_CODE: 0: success, 0x0A: failed

12.1.4.12 int device_controlBeep (int *index,* int *frequency,* int *duration*)

Control Beep

Controls the Beeper

Parameters

index	For Augusta, must be set to 1 (only one beeper)
frequency	Frequency, range 1000-20000 (suggest minimum 3000)
duration	Duration, in milliseconds (range 1 - 65525)

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.1.4.13 int device_controlLED (byte *indexLED,* byte *control,* int *intervalOn,* int *intervalOff*)

Control MSR LED

Controls the LED for the MSR

Parameters

indexLED	For Augusta, must be set to 1 (MSR LED)
control	LED Status: • 00: OFF • 01: RED Solid • 02: RED Blink • 11: GREEN Solid • 12: GREEN Blink • 21: BLUE Solid • 22: BLUE Blink
intervalOn	Blink interval ON, in ms (Range 200 - 2000)
intervalOff	Blink interval OFF, in ms (Range 200 - 2000)

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.1.4.14 int device_controlLED_ICC (int *controlMode,* int *interval*)

Control ICC LED

Controls the LED for the ICC card slot

Parameters

controlMode	0 = off, 1 = solid, 2 = blink
interval	Blink interval, in ms (500 = 500 ms)

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.1.4.15 int device_controlLED_MSR (byte *control,* int *intervalOn,* int *intervalOff*)

Control the MSR LED

Controls the MSR / ICC LED This API not recommended to control ICC LED

Parameters

control	• 0x00 = off, • 0x01 = RED Solid, • 0x02 = RED Blink, • 0x11 = GREEN Solid, • 0x12 = GREEN Blink, • 0x21 = BLUE Solid, • 0x22 = BLUE Blink,
intervalOn	Blink interval on time last, in ms (500 = 500 ms, valid from 200 to 2000)
intervalOff	Blink interval off time last, in ms (500 = 500 ms, valid from 200 to 2000)

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.1.4.16 int device_getCurrentDeviceType ( )

Get current active device type

Returns

: return the device type defined as DEVICE_TYPE in the IDTDef.h

12.1.4.17 int device_getDRS (BYTE * *codeDRS,* int * *codeDRSLen*)

Get DRS Status

Gets the status of DRS(Destructive Reset).

Parameters

codeDRS	the data format is [DRS SourceBlk Number] [SourceBlk1] ... [SourceBlkN] [DRS SourceBlk Number] is 2 bytes, format is NumL NumH. It is Number of [SourceBlkX] [SourceBlkX] is n bytes, Format is [SourceID] [SourceLen] [SourceData] [SourceID] is 1 byte [SourceLen] is 1 byte, it is length of [SourceData]

[SourceID] [SourceLen] [SourceData] 00 1 01 - Application Error 01 1 01 - Application Error 02 1 0x01 - EMV L2 Configuration Check Value Error 0x02 - Future Key Check Value Error 10 1 01 - Battery Error 11 1 Bit 0 - Tamper Switch 1 (0-No, 1-Error) Bit 1 - Tamper Switch 2 (0-No, 1-Error) Bit 2 - Tamper Switch 3 (0-No, 1-Error) Bit 3 - Tamper Switch 4 (0-No, 1-Error) Bit 4 - Tamper Switch 5 (0-No, 1-Error) Bit 5 - Tamper Switch 6 (0-No, 1-Error)

12 1 01 - TemperatureHigh or Low 13 1 01 - Voltage High or Low 1F 4 Reg31∼24bits, Reg23∼16bits, Reg15∼8bits, Reg7∼0bits

Parameters

codeDRSLen	the length of codeDRS

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString() Note: Only support TTK devices

12.1.4.18 int device_getFirmwareVersion (OUT char * *firmwareVersion*)

Polls device for Firmware Version

Parameters

firmwareVersion	Response returned of Firmware Version; needs to have at least 128 bytes of memory

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.1.4.19 int device_getFirmwareVersion_Len (OUT char * *firmwareVersion,* IN_OUT int * *firmwareVersionLen*)

DEPRECATED : please use device_getFirmwareVersion_Len(OUT char* firmwareVersion, IN_OUT int *firmwareVersionLen)

Polls device for Firmware Version

Parameters

firmwareVersion	Response returned of Firmware Version
firmwareVersionLen	Length of Firmware Version

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.1.4.20 int device_getKeyStatus (int * *newFormat,* BYTE * *status,* int * *statusLen*)

Get Key Status

Gets the status of loaded keys

Parameters

status	newFormat for Augusta and miniSmartII only 1: new format of key status 0: reserved format for support previous device
status	For L100, Augusta and miniSmartII: When the newFormat is 0, data format as follows. For Augusta and miniSmartII: byte 0: PIN DUKPT Key, Does not support, always 0 byte 1: PIN Master Key, Does not support, always 0 byte 2: PIN Session Key, Does not support, always 0 byte 3: Account/MSR DUKPT Key, 1 Exists, 0 None, 0xFF STOP byte 4: Account/ICC DUKPT Key, 1 Exists, 0 None, 0xFF STOP byte 5: Admin DUKPT Key, 1 Exists, 0 None, 0xFF STOP For L100: byte 0: PIN DUKPT Key byte 1: PIN Master Key byte 2: Standard PIN Session Key byte 3: Desjardins PIN Session Key byte 4: Account/MSR DUKPT Key, 1 Exists, 0 None, 0xFF STOP, Does not support, always 0 byte 5: Account/ICC DUKPT Key, 1 Exists, 0 None, 0xFF STOP, Does not support, always 0 byte 6: Admin DUKPT Key, 1 Exists, 0 None, 0xFF STOP byte 7: Data DUKPT Key, 1 Exists, 0 None, 0xFF STOP byte 8: MAC DUKPT Key, 1 Exists, 0 None, 0xFF STOP

when the newFormat is 1, data format as follows. [Block Length] [KeyStatusBlock1] [KeyStatusBlock2]...[KeyStatusBlockN] Where: [Block Length] is 2 bytes, format is Len_L Len_H, is KeyStatusBlock Number [KeyStatusBlockX> is 4 bytes, format is [Key Index and Key Name] [key slot] [key status]: [Key Index and Key Name] is 1 byte. Please refer to following table 0x14 LCL-KEK to Encrypt Other Keys 0x02 Data encryption Key to Encrypt ICC/MSR 0x05 MAC DUKPT Key for Host-Device - MAC Verification 0x05 MTK DUKPT Key for TTK Self-Test 0x0C RKI-KEK for Remote Key Injection [key slot] is 2 bytes. Range is 0 - 9999 the MTK DUKPT Key slot is 16, the others are all 0 [key status] is 1 byte. 0 - Not Exist 1 - Exist 0xFF - (Stop. Only Valid for DUKPT Key) For NEO2 and SREDKey2: Each unit of three bytes represents one key's parameters (index and slot). Key Name Index (1 byte): 0x14 - LCL-KEK 0x01 - Pin encryption Key (NEO2 only) 0x02 - Data encryption Key 0x05 - MAC DUKPT Key 0x0A - PCI Pairing Key (NEO2 only) Key Slot (2 bytes): Indicate different slots of a certain Key Name Example: slot =5 (0x00 0x05), slot=300 (0x01 0x2C) For BTPay380, slot is always 0 For example, 0x14 0x00 0x00 0x02 0x00 0x00 0x0A 0x00 0x00 will represent [KeyNameIndex=0x14,KeySlot=0x0000], [KeyNameIndex=0x02,KeySlot=0x0000] and [KeyNameIndex=0x0A,KeySlot=0x0000]

Parameters

statusLen	the length of status

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.1.4.21 void device_getResponseCodeString (IN int *returnCode,* OUT char * *despcrition*)

Review the return code description.

Parameters

returnCode	the response result.
description	

Return values

the	string for description of response result

- 0: "no error, beginning task";
- 1: "no response from reader";
- 2: "invalid response data";
- 3: "time out for task or CMD";
- 4: "wrong parameter";
- 5: "SDK is doing MSR or ICC task";
- 6: "SDK is doing PINPad task";
- 7: "SDK is doing CTLS task";
- 8: "SDK is doing EMV task";
- 9: "SDK is doing Other task";
- 10: "err response or data";
- 11: "no reader attached";
- 12: "mono audio is enabled";
- 13: "did connection";
- 14: "audio volume is too low";
- 15: "task or CMD be canceled";
- 16: "UF wrong string format";
- 17: "UF file not found";
- 18: "UF wrong file format";
- 19: "Attempt to contact online host failed";
- 20: "Attempt to perform RKI failed";
- 22: "Buffer size is not enough";
- 0X300: "Key Type(TDES) of Session Key is not same as the related Master Key.";

- 0X400: "Related Key was not loaded.";
- 0X500: "Key Same.";
- 0X501: "Key is all zero";
- 0X502: "TR-31 format error";
- 0X702: "PAN is Error Key.";
- 0X705: "No Internal MSR PAN (or Internal MSR PAN is erased timeout)";
- 0X0C01: "Incorrect Frame Tag";
- 0X0C02: "Incorrect Frame Type";
- 0X0C03: "Unknown Frame Type";
- 0X0C04: "Unknown Command";
- 0X0C05: "Unknown Sub-Command";
- 0X0C06: "CRC Error";
- 0X0C07: "Failed";
- 0X0C08: "Timeout";
- 0X0C0A: "Incorrect Parameter";
- 0X0C0B: "Command Not Supported";
- 0X0C0C: "Sub-Command Not Supported";
- 0X0C0D: "Parameter Not Supported / Status Abort Command";
- 0X0C0F: "Sub-Command Not Allowed";
- 0X0D01: "Incorrect Header Tag";
- 0X0D02: "Unknown Command";
- 0X0D03: "Unknown Sub-Command";
- 0X0D04: "CRC Error in Frame";
- 0X0D05: "Incorrect Parameter";
- 0X0D06: "Parameter Not Supported";
- 0X0D07: "Mal-formatted Data";
- 0X0D08: "Timeout";
- 0X0D0A: "Failed / NACK";
- 0X0D0B: "Command not Allowed";
- 0X0D0C: "Sub-Command not Allowed";
- 0X0D0D: "Buffer Overflow (Data Length too large for reader buffer)";
- 0X0D0E: "User Interface Event";
- 0X0D11: "Communication type not supported, VT-1, burst, etc.";
- 0X0D12: "Secure interface is not functional or is in an intermediate state.";
- 0X0D13: "Data field is not mod 8";
- 0X0D14: "Pad - 0X80 not found where expected";

- 0X0D15: "Specified key type is invalid";
- 0X0D1: "Could not retrieve key from the SAM(InitSecureComm)";
- 0X0D17: "Hash code problem";
- 0X0D18: "Could not store the key into the SAM(InstallKey)";
- 0X0D19: "Frame is too large";
- 0X0D1A: "Unit powered up in authentication state but POS must resend the InitSecureComm command";
- 0X0D1B: "The EEPROM may not be initialized because SecCommInterface does not make sense";
- 0X0D1C: "Problem encoding APDU";
- 0X0D20: "Unsupported Index(ILM) SAM Transceiver error - problem communicating with the SAM(Key Mgr)";
- 0X0D2: "Unexpected Sequence Counter in multiple frames for single bitmap(ILM) Length error in data returned from the SAM(Key Mgr)";
- 0X0D22: "Improper bit map(ILM)";
- 0X0D23: "Request Online Authorization";
- 0X0D24: "ViVOCard3 raw data read successful";
- 0X0D25: "Message index not available(ILM) ViVOcomm activate transaction card type(ViVOcomm)";
- 0X0D26: "Version Information Mismatch(ILM)";
- 0X0D27: "Not sending commands in correct index message index(ILM)";
- 0X0D28: "Time out or next expected message not received(ILM)";
- 0X0D29: "ILM languages not available for viewing(ILM)";
- 0X0D2A: "Other language not supported(ILM)";
- 0X0D41: "Unknown Error from SAM";
- 0X0D42: "Invalid data detected by SAM";
- 0X0D43: "Incomplete data detected by SAM";
- 0X0D44: "Reserved";
- 0X0D45: "Invalid key hash algorithm";
- 0X0D46: "Invalid key encryption algorithm";
- 0X0D47: "Invalid modulus length";
- 0X0D48: "Invalid exponent";
- 0X0D49: "Key already exists";
- 0X0D4A: "No space for new RID";
- 0X0D4B: "Key not found";
- 0X0D4C: "Crypto not responding";
- 0X0D4D: "Crypto communication error";
- 0X0D4E: "Module-specific error for Key Manager";
- 0X0D4F: "All key slots are full (maximum number of keys has been installed)";
- 0X0D50: "Auto-Switch OK";

- 0X0D51: "Auto-Switch failed";
- 0X0D90: "Account DUKPT Key not exist";
- 0X0D91: "Account DUKPT Key KSN exausted";
- 0X0D00: "This Key had been loaded.";
- 0X0E00: "Base Time was loaded.";
- 0X0F00: "Encryption Or Decryption Failed.";
- 0X1000: "Battery Low Warning (It is High Priority Response while Battery is Low.)";
- 0X1800: "Send 'Cancel Command' after send 'Get Encrypted PIN' & 'Get Numeric' & 'Get Amount'; - 0X1900: "Press 'Cancel' key after send 'Get Encrypted PIN' & 'Get Numeric' & 'Get Amount';
- 0X30FF: "Security Chip is not connect";
- 0X3000: "Security Chip is deactivation & Device is In Removal Legally State.";
- 0X3101: "Security Chip is activation & Device is In Removal Legally State.";
- 0X5500: "No Admin DUKPT Key.";
- 0X5501: "Admin DUKPT Key STOP.";
- 0X5502: "Admin DUKPT Key KSN is Error.";
- 0X5503: "Get Authentication Code1 Failed.";
- 0X5504: "Validate Authentication Code Error.";
- 0X5505: "Encrypt or Decrypt data failed.";
- 0X5506: "Not Support the New Key Type.";
- 0X5507: "New Key Index is Error.";
- 0X5508: "Step Error.";
- 0X5509: "KSN Error";
- 0X550A: "MAC Error.";
- 0X550B: "Key Usage Error.";
- 0X550C: "Mode Of Use Error.";
- 0X550F: "Other Error.";
- 0X6000: "Save or Config Failed / Or Read Config Error.";
- 0X6200: "No Serial Number.";
- 0X6900: "Invalid Command - Protocol is right, but task ID is invalid.";
- 0X6A01: "Unsupported Command - Protocol and task ID are right, but command is invalid - In this State";
- 0X6A00: "Unsupported Command - Protocol and task ID are right, but command is invalid.";
- 0X6B00: "Unknown parameter in command - Protocol task ID and command are right, but parameter is invalid.";
- 0X6C00: "Unknown parameter in command - Protocol task ID and command are right, but length is out of the requirement.";
- 0X7200: "Device is suspend (MKSK suspend or press password suspend).";
- 0X7300: "PIN DUKPT is STOP (21 bit 1).";

- 0X7400: "Device is Busy.";
- 0XE100: "Can not enter sleep mode";
- 0XE200: "File has existed";
- 0XE300: "File has not existed";
- 0XE313: "IO line low -- Card error after session start";
- 0XE400: "Open File Error";
- 0XE500: "SmartCard Error";
- 0XE600: "Get MSR Card data is error";
- 0XE700: "Command time out";
- 0XE800: "File read or write is error";
- 0XE900: "Active 1850 error!";
- 0XEA00: "Load bootloader error";
- 0XEF00: "Protocol Error- STX or ETX or check error.";
- 0XEB00: "Picture is not exist";
- 0X2C02: "No Microprocessor ICC seated";
- 0X2C06: "no card seated to request ATR";
- 0X2D01: "Card Not Supported,";
- 0X2D03: "Card Not Supported, wants CRC";
- 0X690D: "Command not supported on reader without ICC support";
- 0X8100: "ICC error time out on power-up";
- 0X8200: "invalid TS character received - Wrong operation step";
- 0X8300: "Decode MSR Error";
- 0X8400: "TriMagII no Response";
- 0X8500: "No Swipe MSR Card";
- 0X8510: "No Financial Card";
- 0X8600: "Unsupported F, D, or combination of F and D";
- 0X8700: "protocol not supported EMV TD1 out of range";
- 0X8800: "power not at proper level";
- 0X8900: "ATR length too long";
- 0X8B01: "EMV invalid TA1 byte value";
- 0X8B02: "EMV TB1 required";
- 0X8B03: "EMV Unsupported TB1 only 00 allowed";
- 0X8B04: "EMV Card Error, invalid BWI or CWI";
- 0X8B06: "EMV TB2 not allowed in ATR";
- 0X8B07: "EMV TC2 out of range";
- 0X8B08: "EMV TC2 out of range";

- 0X8B09: "per EMV96 TA3 must be > - 0XF";
- 0X8B10: "ICC error on power-up";
- 0X8B11: "EMV T=1 then TB3 required";
- 0X8B12: "Card Error, invalid BWI or CWI";
- 0X8B13: "Card Error, invalid BWI or CWI";
- 0X8B17: "EMV TC1/TB3 conflict-";
- 0X8B20: "EMV TD2 out of range must be T=1";
- 0X8C00: "TCK error";
- 0XA304: "connector has no voltage setting";
- 0XA305: "ICC error on power-up invalid (SBLK(IFSD) exchange";
- 0XE301: "ICC error after session start";
- 0XFF00: "Request to go online";
- 0XFF01: "EMV: Accept the offline transaction";
- 0XFF02: "EMV: Decline the offline transaction";
- 0XFF03: "EMV: Accept the online transaction";
- 0XFF04: "EMV: Decline the online transaction";
- 0XFF05: "EMV: Application may fallback to magstripe technology";
- 0XFF06: "EMV: ICC detected tah the conditions of use are not satisfied";
- 0XFF07: "EMV: ICC didn't accept transaction";
- 0XFF08: "EMV: Transaction was cancelled";
- 0XFF09: "EMV: Application was not selected by kernel or ICC format error or ICC missing data error";
- 0XFF0A: "EMV: Transaction is terminated";
- 0XFF0B: "EMV: Other EMV Error";
- 0XFFFF: "NO RESPONSE";
- 0XF002: "ICC communication timeout";
- 0XF003: "ICC communication Error";
- 0XF00F: "ICC Card Seated and Highest Priority, disable MSR work request";
- 0XF200: "AID List / Application Data is not exist";
- 0XF201: "Terminal Data is not exist";
- 0XF202: "TLV format is error";
- 0XF203: "AID List is full";
- 0XF204: "Any CA Key is not exist";
- 0XF205: "CA Key RID is not exist";
- 0XF206: "CA Key Index it not exist";
- 0XF207: "CA Key is full";
- 0XF208: "CA Key Hash Value is Error";

- 0XF209: "Transaction format error";
- 0XF20A: "The command will not be processing";
- 0XF20B: "CRL is not exist";
- 0XF20C: "CRL number exceed max number";
- 0XF20D: "Amount,Other Amount,Trasaction Type are missing";
- 0XF20E: "The Identification of algorithm is mistake";
- 0XF20F: "No Financial Card";
- 0XF210: "In Encrypt Result state, TLV total Length is greater than Max Length";
- 0X1001: "INVALID ARG";
- 0X1002: "FILE_OPEN_FAILED";
- 0X1003: "FILE OPERATION_FAILED";
- 0X2001: "MEMORY_NOT_ENOUGH";
- 0X3002: "SMARTCARD_FAIL";
- 0X3003: "SMARTCARD_INIT_FAILED";
- 0X3004: "FALLBACK_SITUATION";
- 0X3005: "SMARTCARD_ABSENT";
- 0X3006: "SMARTCARD_TIMEOUT";
- 0X3012: "EMV_RESULT_CODE_MSR_CARD_ERROR_FALLBACK";
- 0X5001: "EMV_PARSING_TAGS_FAILED";
- 0X5002: "EMV_DUPLICATE_CARD_DATA_ELEMENT";
- 0X5003: "EMV_DATA_FORMAT_INCORRECT";
- 0X5004: "EMV_NO_TERM_APP";
- 0X5005: "EMV_NO_MATCHING_APP";
- 0X5006: "EMV_MISSING_MANDATORY_OBJECT";
- 0X5007: "EMV_APP_SELECTION_RETRY";
- 0X5008: "EMV_GET_AMOUNT_ERROR";
- 0X5009: "EMV_CARD_REJECTED";
- 0X5010: "EMV_AIP_NOT_RECEIVED";
- 0X5011: "EMV_AFL_NOT_RECEIVED";
- 0X5012: "EMV_AFL_LEN_OUT_OF_RANGE";
- 0X5013: "EMV_SFI_OUT_OF_RANGE";
- 0X5014: "EMV_AFL_INCORRECT";
- 0X5015: "EMV_EXP_DATE_INCORRECT";
- 0X5016: "EMV_EFF_DATE_INCORRECT";
- 0X5017: "EMV_ISS_COD_TBL_OUT_OF_RANGE";
- 0X5018: "EMV_CRYPTOGRAM_TYPE_INCORRECT";

- 0X5019: "EMV_PSE_NOT_SUPPORTED_BY_CARD";
- 0X5020: "EMV_USER_SELECTED_LANGUAGE";
- 0X5021: "EMV_SERVICE_NOT_ALLOWED";
- 0X5022: "EMV_NO_TAG_FOUND";
- 0X5023: "EMV_CARD_BLOCKED";
- 0X5024: "EMV_LEN_INCORRECT";
- 0X5025: "CARD_COM_ERROR";
- 0X5026: "EMV_TSC_NOT_INCREASED";
- 0X5027: "EMV_HASH_INCORRECT";
- 0X5028: "EMV_NO_ARC";
- 0X5029: "EMV_INVALID_ARC";
- 0X5030: "EMV_NO_ONLINE_COMM";
- 0X5031: "TRAN_TYPE_INCORRECT";
- 0X5032: "EMV_APP_NO_SUPPORT";
- 0X5033: "EMV_APP_NOT_SELECT";
- 0X5034: "EMV_LANG_NOT_SELECT";
- 0X5035: "EMV_NO_TERM_DATA";
- 0X5039: "EMV_PIN_ENTRY_TIMEOUT";
- 0X6001: "CVM_TYPE_UNKNOWN";
- 0X6002: "CVM_AIP_NOT_SUPPORTED";
- 0X6003: "CVM_TAG_8E_MISSING";
- 0X6004: "CVM_TAG_8E_FORMAT_ERROR";
- 0X6005: "CVM_CODE_IS_NOT_SUPPORTED";
- 0X6006: "CVM_COND_CODE_IS_NOT_SUPPORTED";
- 0X6007: "NO_MORE_CVM";
- 0X6008: "PIN_BYPASSED_BEFORE";
- 0X7001: "PK_BUFFER_SIZE_TOO_BIG";
- 0X7002: "PK_FILE_WRITE_ERROR";
- 0X7003: "PK_HASH_ERROR";
- 0X8001: "NO_CARD_HOLDER_CONFIRMATION";
- 0X8002: "GET_ONLINE_PIN";
- 0XD000: "Data not exist";
- 0XD001: "Data access error";
- 0XD100: "RID not exist";
- 0XD101: "RID existed";
- 0XD102: "Index not exist";

- 0XD200: "Maximum exceeded";
- 0XD201: "Hash error";
- 0XD205: "System Busy";
- 0X0E01: "Unable to go online";
- 0X0E02: "Technical Issue";
- 0X0E03: "Declined";
- 0X0E04: "Issuer Referral transaction";
- 0X0F01: "Decline the online transaction";
- 0X0F02: "Request to go online";
- 0X0F03: "Transaction is terminated";
- 0X0F05: "Application was not selected by kernel or ICC format error or ICC missing data error";
- 0X0F07: "ICC didn't accept transaction";
- 0X0F0A: "Application may fallback to magstripe technology";
- 0X0F0C: "Transaction was cancelled";
- 0X0F0D: "Timeout";
- 0X0F0F: "Other EMV Error";
- 0X0F10: "Accept the offline transaction";
- 0X0F11: "Decline the offline transaction";
- 0X0F21: "ICC detected tah the conditions of use are not satisfied";
- 0X0F22: "No app were found on card matching terminal configuration";
- 0X0F23: "Terminal file does not exist";
- 0X0F24: "CAPK file does not exist";
- 0X0F25: "CRL Entry does not exist";
- 0X0FFE: "code when blocking is disabled";
- 0X0FFF: "code when command is not applicable on the selected device";
- 0XF005: "ICC Encrypted C-APDU Data Structure Length Error Or Format Error.";
- 0XBBE0: "CM100 Success";
- 0XBBE1: "CM100 Parameter Error";
- 0XBBE2: "CM100 Low Output Buffer";
- 0XBBE3: "CM100 Card Not Found";
- 0XBBE4: "CM100 Collision Card Exists";
- 0XBBE5: "CM100 Too Many Cards Exist";
- 0XBBE6: "CM100 Saved Data Does Not Exist";
- 0XBBE8: "CM100 No Data Available";
- 0XBBE9: "CM100 Invalid CID Returned";
- 0XBBEA: "CM100 Invalid Card Exists";

- 0XBBEC: "CM100 Command Unsupported";
- 0XBBED: "CM100 Error In Command Process";
- 0XBBEE: "CM100 Invalid Command";
- 0X9031: "Unknown command";
- 0X9032: "Wrong parameter (such as the length of the command is incorrect)";
- 0X9038: "Wait (the command couldnt be finished in BWT)";
- 0X9039: "Busy (a previously command has not been finished)";
- 0X903A: "Number of retries over limit";
- 0X9040: "Invalid Manufacturing system data";
- 0X9041: "Not authenticated";
- 0X9042: "Invalid Master DUKPT Key";
- 0X9043: "Invalid MAC Key";
- 0X9044: "Reserved for future use";
- 0X9045: "Reserved for future use";
- 0X9046: "Invalid DATA DUKPT Key";
- 0X9047: "Invalid PIN Pairing DUKPT Key";
- 0X9048: "Invalid DATA Pairing DUKPT Key";
- 0X9049: "No nonce generated";
- 0X9949: "No GUID available. Perform getVersion first.";
- 0X9950: "MAC Calculation unsuccessful. Check BDK value.";
- 0X904A: "Not ready";
- 0X904B: "Not MAC data";
- 0X9050: "Invalid Certificate";
- 0X9051: "Duplicate key detected";
- 0X9052: "AT checks failed";
- 0X9053: "TR34 checks failed";
- 0X9054: "TR31 checks failed";
- 0X9055: "MAC checks failed";
- 0X9056: "Firmware download failed";
- 0X9060: "Log is full";
- 0X9061: "Removal sensor unengaged";
- 0X9062: "Any hardware problems";
- 0X9070: "ICC communication timeout";
- 0X9071: "ICC data error (such check sum error)";
- 0X9072: "Smart Card not powered up";

12.1.4.22 int device_getSDKWaitTime ( )

Get SDK Wait Time

Get the SDK wait time for transactions

Returns

SDK wait time in seconds

12.1.4.23 int device_getThreadStackSize ( )

Get Thread Stack Size

Get the stack size setting for newly created threads

Returns

Thread Stack Size

12.1.4.24 int device_init ( )

Initial the device by USB

It will detect the device and trying connect.

The connect status can be checked by device_isConnected().

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.1.4.25 int device_isAttached (int *deviceType*)

Check if the device is attached to the USB port The function device_init() must be called before this function.

Parameters

deviceType,the	device type of the USB device

Returns

1 if the device is attached, or 0 if the device is not attached

12.1.4.26 int device_isConnected ( )

Check the device conntected status

Returns

DEVICE_DISCONNECT=0, or DEVICE_CONNECTED = 1

12.1.4.27 int 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()

12.1.4.28 void device_registerCameraCallBk (pCMR_callBack *pCMRf*)

To register the camera callback function to get the image data. (Pass NULL to disable the callback.)

12.1.4.29 void device_registerCardStatusFrontSwitchCallBk (pCSFS_callBack *pCSFSf*)

To register the card status and front switch callback function to get status. (Pass NULL to disable the callback.)

12.1.4.30 void device_registerFWCallBk (pFW_callBack *pFWf*)

To register the firmware update callback function to get the status of firmware update. (Pass NULL to disable the callback.)

12.1.4.31 int device_SendDataCommand (IN BYTE * *cmd*, IN int *cmdLen*, IN BYTE * *data*, IN int *dataLen*, OUT BYTE * *response*, IN_OUT int * *respLen*)

Send a Command to device

Sends a command to the device .

Parameters

cmd	buffer of command to execute.
cmdLen,the	length of the buffer cmd.
data	buffer of IDG command data.
dataLen,the	length of the buffer data.
response	Response data
respLen,the	length of Response data

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.1.4.32 int device_setCurrentDevice (int *deviceType*)

Sets the current device to talk to

The connect status can be checked by device_isConnected().

Parameters

deviceType	Device to connect to
	
	`enum DEVICE_TYPE`
	`{`
	`IDT_DEVICE_UNKNOWN=0,`
	`IDT_DEVICE_AUGUSTA_HID,`
	`IDT_DEVICE_AUGUSTA_KB,`
	`IDT_DEVICE_AUGUSTA_S_HID,`
	`IDT_DEVICE_AUGUSTA_S_KB,`
	`IDT_DEVICE_AUGUSTA_S_TTK_HID,`
	`IDT_DEVICE_SPECTRUM_PRO,`
	`IDT_DEVICE_MINISMART_II,`
	`IDT_DEVICE_L100,`
	`IDT_DEVICE_UNIPAY,`
	`IDT_DEVICE_UNIPAY_I_V,`
	`IDT_DEVICE_VP3300_AJ,`
	`IDT_DEVICE_KIOSK_III,`
	`IDT_DEVICE_KIOSK_III_S,`
	`IDT_DEVICE_PIP_READER,`
	`IDT_DEVICE_VENDI,`
	`IDT_DEVICE_VP3300_USB,`
	`IDT_DEVICE_UNIPAY_I_V_TTK,`
	`IDT_DEVICE_VP3300_BT,`
	`IDT_DEVICE_VP8800,`
	`IDT_DEVICE_SREDKEY2_HID,`
	`IDT_DEVICE_SREDKEY2_KB,`
	`IDT_DEVICE_NEO2,`
	`IDT_DEVICE_MINISMART_II_COM = IDT_DEVICE_NEO2+5,`
	`IDT_DEVICE_SPECTRUM_PRO_COM,`
	`IDT_DEVICE_KIOSK_III_COM,`
	`IDT_DEVICE_KIOSK_III_S_COM,`
	`IDT_DEVICE_VP3300_COM,`
	`IDT_DEVICE_NEO2_COM,`
	`IDT_DEVICE_MAX_DEVICES = IDT_DEVICE_NEO2_COM+5`
	`};`

Returns

RETURN_CODE: 1: success, 0: failed

12.1.4.33 void device_setSDKWaitTime (int *waitTime*)

Set SDK Wait Time

Set the SDK wait time for transactions

Parameters

waitTime	The SDK wait time for transaction in seconds

12.1.4.34 void device_setThreadStackSize (int *threadSize*)

Set Thread Stack Size

Set the stack size setting for newly created threads

12.1.4.35 int device_updateFirmware (IN BYTE * *firmwareData,* IN int *firmwareDataLen,* IN char * *firmwareName,* IN int *encryptionType,* IN BYTE * *keyBlob,* IN int *keyBlobLen*)

Update Firmware Updates the firmware of Augusta.

Parameters

firmwareData	Signed binary data of a firmware file provided by IDTech
firmwareDataLen	Length of firmwareData
firmwareName	Firmware name. • For example "Augusta_S_TTK_V1.00.002.fm"
encryptionType	Encryption type • 0 : Plaintext • 1 : TDES ECB, PKCS#5 padding • 2 : TDES CBC, PKCS#5, IV is all 0
keyBlob	Encrypted firmware session key blob, TR-31 Rev B, wrapped by FW Key (Optional, none if firmware is plaintext)
keyBlobLen	Length of keyBlob

Returns

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

Firmware update status is returned in the callback with the following values: sender = AUGUSTA state = DeviceState.FirmwareUpdate data = File Progress. Two bytes, with byte[0] = current block, and byte[1] = total blocks. 0x0310 = block 3 of 16 transactionResultCode:

- RETURN_CODE_DO_SUCCESS = Firmware Update Completed Successfully
- RETURN_CODE_BLOCK_TRANSFER_SUCCESS = Current block transferred successfully
- Any other return code represents an error condition

12.1.4.36 int emv_activateTransaction (IN int *timeout,* IN BYTE * *tags,* IN int *tagsLen,* IN int *forceOnline*)

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	Tags to be included in the request. Passed as a TLV stream. Example, tag 9F0C with amount 0x000000000100 would be 0x9F0C06000000000100
tagsLen	Length of tags
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

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString >>>>>IMPORTANT: parameters for amount and amtOther MUST BE PASSED AS A DOUBLE VALUE WITH DECIMAL POINT. Example, do not pass 1, but instead pass 1.0 or 1.00. Otherwise, results will be unpredictable

12.1.4.37 void emv_allowFallback (IN int *allow*)

Allow fallback for EMV transactions. Default is TRUE

Parameters

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

12.1.4.38 int emv_authenticateTransaction (IN BYTE * *updatedTLV,* IN int *updatedTLVLen*)

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.

Parameters

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, 9F37 to be included in response = DFEE1-A079F029F36959F37
updatedTLVLen	

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.1.4.39 int emv_authenticateTransactionWithTimeout (IN int *timeout,* IN BYTE * *updatedTLV,* IN int *updatedTLVLen*)

Authenticate EMV Transaction Request with Timeout

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.

Parameters

timeout	Timeout value in seconds.
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, 9F37 to be included in response = DFEE1-A079F029F36959F37

updatedTLVLen	

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.1.4.40 int emv_callbackResponseLCD (IN int *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_DISPLAY_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 device_getResponseCodeString()

12.1.4.41 int emv_callbackResponseMSR (IN BYTE * *MSR*, IN_OUT int *MSRLen*)

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
MSRLen	the length of Swiped track data

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.1.4.42 int emv_cancelTransaction ( )

Cancel EMV Transaction

Cancels the currently executing EMV transaction.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.1.4.43 int emv_completeTransaction (IN int *commError,* IN BYTE * *authCode,* IN int *authCodeLen,* IN BYTE * *iad,* IN int *iadLen,* IN BYTE * *tlvScripts,* IN int *tlvScriptsLen,* IN BYTE * *tlv,* IN int *tlvLen*)

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(1) if host was unreachable, or FALSE(0) 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
authCodeLen	the length of authCode
iad	Issuer Authentication Data, if any. Example 0x11223344556677883030 (tag value 91).
iadLen	the length of iadLen
tlvScripts	71/72 scripts, if any
tlvScriptsLen	the length of tlvScriptsLen
tlv	Additional TLV data to return with transaction results (if any)
tlvLen	the length of tlv

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.1.4.44 int emv_getAutoAuthenticateTransaction ( )

Gets auto authenticate value for EMV transactions.

Returns

RETURN_CODE: TRUE = auto authenticate, FALSE = manually authenticate

12.1.4.45 int emv_getAutoCompleteTransaction ( )

Gets auto complete value for EMV transactions.

Returns

RETURN_CODE: TRUE = auto complete, FALSE = manually complete

12.1.4.46 int emv_getEMVConfigurationCheckValue (OUT BYTE * *checkValue,* IN_OUT int * *checkValueLen*)

Get EMV Kernel configuration check value info

Parameters

checkValue	Response returned of Kernel configuration check value info
checkValueLen	the length of checkValue

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.1.4.47 int emv_getEMVKernelCheckValue (OUT BYTE * *checkValue,* IN_OUT int * *checkValueLen*)

Get EMV Kernel check value info

Parameters

checkValue	Response returned of Kernel check value info
checkValueLen	the length of checkValue

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.1.4.48 int emv_getEMVKernelVersion (OUT char * *version*)

DEPRECATED : please use emv_getEMVKernelVersion_Len(OUT char* version, IN_OUT int *versionLen)

Polls device for EMV Kernel Version

Parameters

version	Response returned of Kernel Version; needs to have at least 128 bytes of memory.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.1.4.49 int emv_getEMVKernelVersion_Len (OUT char * *version,* IN_OUT int * *versionLen*)

Polls device for EMV Kernel Version

Parameters

version	Response returned of Kernel Version
versionLen	Length of version

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.1.4.50 void emv_registerCallBk (pEMV_callBack *pEMVf*)

To register the emv callback function to get the EMV processing response. (Pass NULL to disable the callback.)

12.1.4.51 int emv_removeAllApplicationData ( )

Remove All Application Data

Removes all the Application Data

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.1.4.52 int emv_removeAllCAPK ( )

Remove All Certificate Authority Public Key

Removes all the CAPK

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.1.4.53 int emv_removeAllCRL ( )

Remove All Certificate Revocation List Entries

Removes all CRLEntry entries

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.1.4.54 int emv_removeApplicationData (IN BYTE * *AID*, IN int *AIDLen*)

Remove Application Data by AID

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

Parameters

AID	Name of ApplicationID Must be between 5 and 16 bytes
AIDLen	the length of AID data buffer

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.1.4.55 int emv_removeCAPK (IN BYTE * *capk*, IN int *capkLen*)

Remove Certificate Authority Public Key

Removes the CAPK as specified by the RID/Index

Parameters

capk	6 byte CAPK = 5 bytes RID + 1 byte INDEX
capkLen	the length of capk data buffer

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.1.4.56 int emv_removeCRL (IN BYTE * *list*, IN int *lsLen*)

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
lssLen	the length of list data buffer

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.1.4.57 int emv_removeTerminalData ( )

Remove Terminal Data

Removes the Terminal Data

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.1.4.58 int emv_retrieveAIDList (OUT BYTE * *AIDList*, IN_OUT int * *AIDListLen*)

Retrieve AID list

Returns all the AID names installed on the terminal.

Parameters

AIDList	array of AID name byte arrays
AIDListLen	the length of AIDList array buffer

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.1.4.59 int emv_retrieveApplicationData (IN BYTE * *AID*, IN int *AIDLen*, OUT BYTE * *tlv*, IN_OUT int * *tlvLen*)

Retrieve Application Data by AID

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

Parameters

AID	Name of ApplicationID. Must be between 5 and 16 bytes
AIDLen	the length of AID data buffer.
tlv	The TLV elements of the requested AID
tlvLen	the length of tlv data buffer.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.1.4.60 int emv_retrieveCAPK (IN BYTE * *capk*, IN int *capkLen*, OUT BYTE * *key*, IN_OUT int * *keyLen*)

Retrieve Certificate Authority Public Key

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

Parameters

capk	6 bytes CAPK = 5 bytes RID + 1 byte Index
capkLen	the length of capk data buffer
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.
keyLen	the length of key data buffer •

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.1.4.61 int emv_retrieveCAPKList (OUT BYTE * *keys*, IN_OUT int * *keysLen*)

Retrieve the Certificate Authority Public Key list

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

Parameters

keys	[key1][key2]...[keyn], each key 6 bytes where key = 5 bytes RID + 1 byte index
keysLen	the length of keys data buffer

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.1.4.62 int emv_retrieveCRL (OUT BYTE * *list*, IN_OUT int * *lssLen*)

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
lssLen	the length of list data buffer

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.1.4.63 int emv_retrieveTerminalData (OUT BYTE * *tlv*, IN_OUT int * *tlvLen*)

Retrieve Terminal Data

Retrieves the Terminal Data.

Parameters

tlv	Response returned as a TLV
tlvLen	the length of tlv data buffer

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.1.4.64 int emv_retrieveTerminalID (OUT char * *terminalID*)

DEPRECATED : please use emv_retrieveTerminalID_Len(OUT char* terminalID, IN_OUT int *terminalIDLen)

Gets the terminal ID as printable characters .

Parameters

terminalID	Terminal ID string; needs to have at least 30 bytes of memory

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.1.4.65 int emv_retrieveTerminalID_Len (OUT char * *terminalID*, IN_OUT int * *terminalIDLen*)

Gets the terminal ID as printable characters .

Parameters

terminalID	Terminal ID string
terminalIDLen	Length of terminalID

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.1.4.66 int emv_retrieveTransactionResult (IN BYTE * *tags*, IN int *tagsLen*, IDTTransactionData * *cardData*)

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
tagsLen	Length of tag list
cardData	All requested tags returned as unencrypted, encrypted and masked TLV data in IDTTransactionData object

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.1.4.67 int emv_setApplicationData (IN BYTE * *name*, IN int *nameLen*, IN BYTE * *tlv*, IN int *tlvLen*)

Set Application Data by AID

Sets the Application Data as specified by the application name and TLV data

Parameters

name	Application name, 10-32 ASCII hex characters representing 5-16 bytes Example "a0000000031010"
nameLen	the length of name data buffer of Application name,
tlv	Application data in TLV format
tlvLen	the length of tlv data buffer

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.1.4.68 void emv_setAutoAuthenticateTransaction (IN int *authenticate*)

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

Parameters

authenticate	TRUE = auto authenticate, FALSE = manually authenticate

12.1.4.69 void emv_setAutoCompleteTransaction (IN int *complete*)

Enables complete for EMV transactions. If a emv_authenticateTranaction results in code 0x0004 (go online), then emv_completeTransaction can automatically execute if parameter is set to TRUE

Parameters

complete	TRUE = auto complete, FALSE = manually complete

12.1.4.70 int emv_setCAPK (IN BYTE * *capk*, IN int *capkLen*)

Set Certificate Authority Public Key

Sets the CAPK as specified by the CAKey structure

Parameters

capk	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.
capkLen	the length of capk data buffer

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.1.4.71 int emv_setCRL (IN BYTE * *list*, IN int *lsLen*)

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]
lssLen	the length of list data buffer

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.1.4.72 int emv_setTerminalData (IN BYTE * *tlv*, IN int *tlvLen*)

Set Terminal Data

Sets the Terminal Data as specified by the TerminalData structure passed as a parameter

Parameters

tlv	TerminalData configuration file
tlvLen	the length of tlv data buffer

Return values

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

12.1.4.73 int emv_setTerminalID (IN char * *terminalID*)

Sets the terminal ID as printable characters .

Parameters

terminalID	Terminal ID to set

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.1.4.74 int emv_startTransaction (IN double *amount,* IN double *amtOther,* IN int *exponent,* IN int *type,* IN int *timeout,* IN BYTE * *tags,* IN int *tagsLen,* IN int *forceOnline*)

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) - SEE IMPORTANT NOTE BELOW
amtOther	Other amount value, if any (tag value 9F03) - SEE IMPORTANT NOTE BELOW
exponent	Number of characters after decimal point
type	Transaction type (tag value 9C).
timeout	Timeout value in seconds.
tags	Any other tags to be included in the request. Passed as a TLV stream. Example, tag 9F0C with amount 0x000000000100 would be 0x9F0C06000000000100 If tags 9F02 (amount),9-F03 (other amount), or 9C (transaction type) are included, they will take priority over these values supplied as individual parameters to this method.
tagsLen	Length of tags
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 = DFEE1A079F029-F36959F37

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString >>>>>IMPORTANT: parameters for amount and amtOther MUST BE PASSED AS A DOUBLE VALUE WITH DECIMAL POINT. Example, do not pass 1, but instead pass 1.0 or 1.00. Otherwise, results will be unpredictable

12.1.4.75 int icc_disable ( )

ICC Function enable/disable Disable ICC function

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.1.4.76 int icc_enable (IN int *withNotification*)

ICC Function enable/disable Enable ICC function with or without seated notification

Parameters

withNotification	• 1: with notification when ICC seated status changed, • 0: without notification.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.1.4.77 int icc_exchangeAPDU (IN BYTE * *c_APDU,* IN int *cLen,* OUT BYTE * *reData,* IN_OUT int * *reLen*)

Exchange APDU with plain text For Non-SRED Augusta Only

Sends an APDU packet to the ICC. If successful, response is the APDU data in response parameter.

Parameters

c_APDU	APDU data packet
cLen	APDU data packet length
reData	Unencrypted APDU response
reLen	Unencrypted APDU response data length

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.1.4.78 int icc_exchangeEncryptedAPDU (IN BYTE * *c_APDU,* IN int *cLen,* OUT BYTE * *reData,* IN_OUT int * *reLen*)

Exchange APDU with encrypted data For SRED Augusta Only

Sends an APDU packet to the ICC. If successful, response is the APDU data in response parameter.

Parameters

c_APDU	KSN + encytpted APDU data packet, or no KSN (use last known KSN) + encrypted APDU data packet With KSN: [0A][KSN][Encrypted C-APDU] Without KSN: [00][Encrypted C-APDU]

The format of Raw C-APDU Data Structure of [m-bytes Encrypted C-APDU] is below:

- m = 2 bytes Valid C-APDU Length + x bytes Valid C-APDU + y bytes Padding (0x00) Note: For TDES mode: 2+x should be multiple of 8. If it was not multiple of 8, unit should padded y bytes 0x00 automatically (2+x+y should be multiple of 8). For AES mode: 2+x should be multiple of 16. If it was not multiple of 16, unit should padded y bytes 0x00 automatically (2+x+y should be multiple of 16).

  Parameters

cLen	data packet length
reData	response encrypted APDU response. Can be three options:

[00] + [Plaintext R-APDU]

- [01] + [0A] + [KSN] + [n bytes Encrypted R-APDU without Status Bytes] + [2 bytes Status Bytes]
- [01] + [00] + [n bytes Encrypted R-APDU without Status Bytes] + [2 bytes Status Bytes]

  The KSN, when provided, will be 10 bytes. The KSN will only be provided when it has changed since the last provided KSN. Each card Power-On generates a new KSN. During a sequence of commands where the KSN

is identical, the first response will have a KSN length set to [0x0A] followed by the KSN, while subsequent commands with the same KSN value will have a KSN length of [0x00] followed by the Encrypted R-APDU without Status Bytes.

Parameters

reLen	encrypted APDU response data length

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.1.4.79 int icc_getAPDU_KSN (OUT BYTE * *KSN,* IN_OUT int * *inLen*)

Get APDU KSN

Retrieves the KSN used in ICC Encypted APDU usage

Parameters

KSN	Returns the encrypted APDU packet KSN
inLen	KSN data length

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.1.4.80 int icc_getFunctionStatus (OUT int * *enabled,* OUT int * *withNotification*)

Get ICC Function status Get ICC Function status about enable/disable and with or without seated notification

Parameters

enabled	• 1: ICC Function enabled, • 0: means disabled.
withNotification	1 means with notification when ICC seated status changed. 0 means without notification.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.1.4.81 int icc_getICCReaderStatus (OUT 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

12.1.4.82 int icc_getKeyFormatForICCDUKPT (OUT BYTE * *format*)

Get Key Format For DUKPT

Specifies how data will be encrypted with Data Key or PIN key (if DUKPT key loaded). This applies to both MSR and ICC

Parameters

format	Response returned from method: • 'TDES': Encrypted card data with TDES if DUKPT Key had been loaded.(default) • 'AES': Encrypted card data with AES if DUKPT Key had been loaded. • 'NONE': No Encryption.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.1.4.83 int icc_getKeyTypeForICCDUKPT (OUT BYTE * *type*)

Get Key Type for DUKPT

Specifies the key type used for ICC DUKPT encryption. This applies to both MSR and ICC.

Parameters

type	Response returned from method: • 'DATA': Encrypted card data with Data Key DUKPT Key had been loaded.(default) • 'PIN': Encrypted card data with PIN Key if DUKPT Key had been loaded.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.1.4.84 int icc_powerOffICC ( )

Power Off ICC

Powers down the ICC

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

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

12.1.4.85 int icc_powerOnICC (OUT BYTE * *ATR,* IN_OUT int * *inLen*)

Power On ICC

Power up the currently selected microprocessor card in the ICC reader

Parameters

ATR,the	ATR data response when succeeded power on ICC,
inLen,the	length of ATR data

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.1.4.86 int msr_cancelMSRSwipe ( )

Disable MSR Swipe Cancels MSR swipe request.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.1.4.87 int msr_captureMode (int *isBufferMode,* int *withNotification*)

Set MSR Capture Mode.

For Non-SRED Augusta Only

Switch between Auto mode and Buffer mode. Auto mode only available on KB interface

Parameters

isBufferMode	• 1: Enter into Buffer mode. • 0: Enter into Auto mode. KB mode only. Swipes automatically captured and sent to keyboard buffer
withNotification	• 1: with notification when swiped MSR data. • 0: without notification when swiped MSR data.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.1.4.88 int msr_disable ( )

Disable MSR Disable MSR functions.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.1.4.89 int msr_getClearPANID (BYTE * *value*)

Get Clear PAN ID.

Returns the number of digits that begin the PAN that will be in the clear

Parameters

value	4901 <Setting value>="">. setting Value: Number of digits in clear. Values are char '0' - '6'

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.1.4.90 int msr_getExpirationMask (BYTE * *value*)

Get MSR expiration date mask.

Parameters

value	5001 <Setting value>="">. setting Value: '0' = masked, '1' = not-masked

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.1.4.91 int msr_getKeyFormatForICCDUKPT (OUT BYTE * *format*)

Get Key Format For DUKPT

Specifies how data will be encrypted with Data Key or PIN key (if DUKPT key loaded). This applies to both MSR and ICC

Parameters

format	Response returned from method: • 'TDES': Encrypted card data with TDES if DUKPT Key had been loaded.(default) • 'AES': Encrypted card data with AES if DUKPT Key had been loaded. • 'NONE': No Encryption.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.1.4.92 int msr_getKeyTypeForICCDUKPT (OUT BYTE * *type*)

Get Key Type for DUKPT

Specifies the key type used for ICC DUKPT encryption. This applies to both MSR and ICC.

Parameters

type	Response returned from method: • 'DATA': Encrypted card data with Data Key DUKPT Key had been loaded.(default) • 'PIN': Encrypted card data with PIN Key if DUKPT Key had been loaded.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.1.4.93 int msr_getMSRData (OUT BYTE * *reData,* IN_OUT int * *reLen*)

Get MSR Data Reads the MSR Data buffer

Parameters

reData	Card data
reLen	Card data length

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.1.4.94 int msr_getSetting (byte *setting*, BYTE * *value*, int * *valueLen*)

Get Single MSR Setting value

Returns the encryption used for swipe data

Parameters

setting	the msr setting to retrieve.
value	MSR Setting value.
valueLen	the length of value.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.1.4.95 int msr_getSwipeForcedEncryptionOption (BYTE * *option*)

Get MSR Swipe Forced Encryption Option.

Parameters

option	8401 <Setting value>="">. Setting Value Byte using lower four bits as flags. 0 = Force Encryption Off, 1 = Force Encryption On bit0 = Track 1 bit1 = Track 2 bit2 = Track 3 bit4 = Track 3 Card Option 0

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.1.4.96 int msr_getSwipeMaskOption (BYTE * *option*)

Get MSR Swipe Mask Option.

Gets the swipe mask/clear data sending option

Parameters

option	8601 <Setting value>="">. Setting Value Byte using lower three bits as flags. 0 = Mask Option Off, 1 = Mask Option On bit0 = Track 1 bit1 = Track 2 bit2 = Track 3 Example: Response 0x03 = Track1/Track2 Masked Option ON, Track3 Masked Option Off

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.1.4.97 void msr_registerCallBk (pMSR_callBack *pMSRf*)

To register the msr callback function to get the MSR card data. (Pass NULL to disable the callback.)

12.1.4.98 void msr_registerCallBkp (pMSR_callBackp *pMSRf*)

To register the msr callback function to get the MSR card data pointer. (Pass NULL to disable the callback.)

12.1.4.99 int msr_setClearPANID (BYTE *val*)

Set Clear PAN ID.

Parameters

val	Set Clear PAN ID to value: Number of digits to show in clear. Range 0-6.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.1.4.100 int msr_setExpirationMask (int *mask*)

Set Expiration Masking

Sets the flag to mask the expiration date

Parameters

mask	TRUE = mask expiration

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.1.4.101 int msr_setKeyFormatForICCDUKPT (IN BYTE *format*)

Set Key Format for DUKPT

Sets how data will be encrypted, with either TDES or AES (if DUKPT key loaded) This applies to both MSR and ICC

Parameters

format	encryption Encryption Type • 00: Encrypt with TDES • 01: Encrypt with AES

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.1.4.102 int msr_setKeyTypeForICCDUKPT (IN BYTE *type*)

Set Key Type for DUKPT Key

Sets which key the data will be encrypted with, with either Data Key or PIN key (if DUKPT key loaded) This applies to both MSR and ICC

Parameters

type	Encryption Type • 00: Encrypt with Data Key • 01: Encrypt with PIN Key

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.1.4.103 int msr_setSetting (BYTE *setting*, BYTE * *val*, int *valLen*)

Set MSR settings.

Parameters

setting	the msr setting to set.
val	the value to set to the msr setting.
valLen	the length of val.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.1.4.104 int msr_setSwipeForcedEncryptionOption (int *track1*, int *track2*, int *track3*, int *track3card0*)

Set MSR Swipe Forced Encryption Option.

Parameters

tarck1	Set track1 encryption to true or false.
tarck2	Set track2 encryption to true or false.
tarck3	Set track3 encryption to true or false.
tarck3card0	Set track3 card0 encryption to true or false.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.1.4.105 int msr_setSwipeMaskOption (int *track1*, int *track2*, int *track3*)

Set MSR Swipe Mask Option.

Sets the swipe mask/clear data sending option

Parameters

tarck1	Set track1 mask to true or false.
tarck2	Set track2 mask to true or false.
tarck3	Set track3 mask to true or false.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.1.4.106 int msr_startMSRSwipe (IN int *_timeout*)

Start MSR Swipe Enables MSR, waiting for swipe to occur. Allows track selection. Returns IDTMSRData instance to deviceDelegate::swipeMSRData:()

Parameters

timeout	Swipe Timeout Value

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.1.4.107 void parseMSRData (IN BYTE * *resData*, IN int *resLen*, IN_OUT IDTMSRData * *cardData*)

Parser the MSR data from the buffer into IDTMSTData structure

Parameters

resData	MSR card data buffer
resLen	the length of resData
cardData	the parser result with IDTMSTData structure

12.1.4.108 int pin_cancelPINEntry ( )

Cancel PIN Entry

Cancels PIN entry request

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.1.4.109 void pin_registerCallBk (pPIN_callBack *pPINf*)

To register the pin callback function to get the PINPad data. (Pass NULL to disable the callback.)

12.1.4.110 void registerHotplugCallBk (pMessageHotplug *pMsgHotplug*)

To register the USB HID hot-plug callback function which implemented in the application to monitor the hotplug message from the SDK.

12.1.4.111 void registerLogCallBk (pSendDataLog *pFSend*, pReadDataLog *pFRead*)

To register the log callback function which implemented in the application to monitor sending/reading data between application and reader.

12.1.4.112 char* SDK_Version ( )

To Get SDK version

Returns

return the SDK version string

12.1.4.113 int setAbsoluteLibraryPath (const char * *absoluteLibraryPath*)

Set the path to use when searching for ID TECH's libraries. If this is not set, the libraries will be searched for with the system's default procedures.

Parameters

absoluteLibrary-Path	The absolute path to ID TECH's libraries.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.2 Source_C/libIDT_Device.h File Reference

Windows C++ API.

```
#include "IDTDef.h"
```

Macros

- #define IN
- #define OUT
- #define IN_OUT

Typedefs

- typedef void(∗ pMessageHotplug)(int, int)
- typedef void(∗ pSendDataLog)(unsigned char ∗, int)
- typedef void(∗ pReadDataLog)(unsigned char ∗, int)
- typedef void(∗ pEMV_callBack)(int, int, unsigned char ∗, int, IDTTransactionData ∗, EMV_Callback ∗, int)
- typedef void(∗ pMSR_callBack)(int, IDTMSRData)
- typedef void(∗ pMSR_callBackp)(int, IDTMSRData ∗)
- typedef void(∗ pPIN_callBack)(int, IDTPINData ∗)
- typedef void(∗ pLCD_callBack)(int, IDTLCDItem ∗)
- typedef void(∗ pCMR_callBack)(int, IDTCMRData ∗)
- typedef void(∗ pCSFS_callBack)(BYTE status)
- typedef void(∗ pFW_callBack)(int, int, int, int, int)
- typedef void(∗ pRKI_callBack)(int, char ∗)
- typedef void(∗ httpComm_callBack)(BYTE ∗, int)
- typedef void(∗ v4Comm_callBack)(BYTE, BYTE, BYTE ∗, int)
- typedef void(∗ ftpComm_callBack)(int, int, int)
- typedef void(∗ pLog_callback)(BYTE, char ∗)

Functions

- void registerHotplugCallBk (pMessageHotplug pMsgHotplug)
- void registerLogCallBk (pSendDataLog pFSend, pReadDataLog pFRead)
- void emv_registerCallBk (pEMV_callBack pEMVf)
- void loyalty_registerCallBk (pEMV_callBack pEMVf)
- void msr_registerCallBk (pMSR_callBack pMSRf)
- void msr_registerCallBkp (pMSR_callBackp pMSRf)
- void pin_registerCallBk (pPIN_callBack pPINf)
- void lcd_registerCallBk (pLCD_callBack pLCDf)
- void device_registerCameraCallBk (pCMR_callBack pCMRf)
- void device_registerCardStatusFrontSwitchCallBk (pCSFS_callBack pCSFSf)

- void device_registerFWCallBk (pFW_callBack pFWf)
- void device_registerRKICallBk (pRKI_callBack pRKIf)
- char * SDK_Version ()
- int setAbsoluteLibraryPath (const char *absoluteLibraryPath)
- int device_setConfigPath (const char *path)
- int device_setNEO2DevicesConfigs (IN const char *configs, IN int len)
- int device_init ()
- int rs232_device_init (int deviceType, int port_number, int brate)
- int device_setCurrentDevice (int deviceType)
- int device_close ()
- void device_getResponseCodeString (IN int returnCode, OUT char *despcrition)
- void device_getIDGStatusCodeString (IN int returnCode, OUT char *despcrition)
- int device_isConnected ()
- int device_isAttached (int deviceType)
- int device_startTransaction (IN double amount, IN double amtOther, IN int type, IN const int _timeout, IN BYTE *tags, IN int tagsLen)
- int loyalty_startTransaction (IN double amount, IN double amtOther, IN int type, IN const int _timeout, IN BYTE *tags, IN int tagsLen, IN const int cardType, IN const int iccReadType)
- void device_setTransactionExponent (int exponent)
- int device_activateTransaction (IN const int _timeout, IN BYTE *tags, IN int tagsLen)
- int device_cancelTransaction ()
- int loyalty_cancelTransaction ()
- int device_setCancelTransactionMode (int mode)
- int device_cancelTransactionSilent (int enable)
- int loyalty_cancelTransactionSilent (int enable)
- int device_getDriveFreeSpace (OUT int *free, OUT int *used)
- int device_listDirectory (IN char *directoryName, IN int directoryNameLen, IN int recursive, IN int onSD, OUT char *directory, IN_OUT int *directoryLen)
- int device_createDirectory (IN char *directoryName, IN int directoryNameLen)
- int device_deleteDirectory (IN char *dirName, IN int dirNameLen)
- int device_transferFile (IN char *fileName, IN int fileNameLen, IN BYTE *file, IN int fileLen)
- int device_deleteFile (IN char *fileName, IN int fileNameLen)
- int device_queryFile (IN char *directoryName, IN int directoryNameLen, IN char *fileName, IN int fileNameLen, OUT int *isExist, OUT BYTE *timeStamp, IN_OUT int *timeStampLen, OUT char *fileSize, IN_OUT int *fileSizeLen)
- int device_startListenNotifications ()
- int device_stopListenNotifications ()
- int device_getFirmwareVersion (OUT char *firmwareVersion)
- int device_getFirmwareVersion_Len (OUT char *firmwareVersion, IN_OUT int *firmwareVersionLen)
- int device_getDateTime (OUT BYTE *dateTime)
- int device_getDateTime_Len (OUT BYTE *dateTime, IN_OUT int *dateTimeLen)
- int device_controlLED (byte indexLED, byte control, int intervalOn, int intervalOff)
- int device_controlLED_ICC (int controlMode, int interval)
- int device_controlLED_MSR (byte control, int intervalOn, int intervalOff)
- int device_controlBeep (int index, int frequency, int duration)
- int device_getDRS (BYTE *codeDRS, int *codeDRSLen)
- int device_getKeyStatus (int *newFormat, BYTE *status, int *statusLen)
- int device_enterStopMode ()
- int device_setSleepModeTime (int time)
- int device_verifyBackdoorKey ()
- int device_selfCheck ()
- int device_pingDevice ()
- int device_controlUserInterface (IN BYTE *values)
- int device_controlIndicator (IN int indicator, IN int enable)
- int device_getCurrentDeviceType ()

- int device_SendDataCommandNEO (IN int cmd, IN int subCmd, IN BYTE *data, IN int dataLen, OUT BYTE *response, IN_OUT int *respLen)
- int device_SendDataCommand (IN BYTE *cmd, IN int cmdLen, IN BYTE *data, IN int dataLen, OUT BYTE *response, IN_OUT int *respLen)
- int device_SendDataCommandITP (IN BYTE *cmd, IN int cmdLen, IN BYTE *data, IN int dataLen, OUT BYTE *response, IN_OUT int *respLen)
- int device_rebootDevice ()
- int device_updateFirmware (IN BYTE *firmwareData, IN int firmwareDataLen, IN char *firmwareName, IN int encryptionType, IN BYTE *keyBlob, IN int keyBlobLen)
- int device_getDeviceMemoryUsageInfo (OUT int *freeHeapSize, OUT int *notFreedBlockCnt, OUT int *minEverFreeHeapSize)
- int felica_authentication (IN BYTE *key, IN int keyLen)
- int felica_readWithMac (IN int blockCnt, IN BYTE *blockList, IN int blockListLen, OUT BYTE *blockData, OUT int *blockDataLen)
- int felica_writeWithMac (IN BYTE blockNum, IN BYTE *blockData, IN int blockDataLen)
- int felica_read (IN BYTE *serviceCodeList, IN int serviceCodeListLen, IN int blockCnt, IN BYTE *blockList, IN int blockListLen, OUT BYTE *blockData, OUT int *blockDataLen)
- int felica_write (IN BYTE *serviceCodeList, IN int serviceCodeListLen, IN int blockCnt, IN BYTE *blockList, IN int blockListLen, IN BYTE *blockData, IN int blockDataLen, OUT BYTE *statusFlag, OUT int *statusFlagLen)
- int felica_poll (IN BYTE *systemCode, IN int systemCodeLen, OUT BYTE *respData, OUT int *respDataLen)
- int felica_SendCommand (IN BYTE *command, IN int commandLen, OUT BYTE *respData, OUT int *respDataLen)
- int felica_requestService (IN BYTE *nodeCode, IN int nodeCodeLen, OUT BYTE *respData, OUT int *respDataLen)
- int config_getModelNumber (OUT char *sNumber)
- int config_getModelNumber_Len (OUT char *sNumber, IN_OUT int *sNumberLen)
- int device_setSystemLanguage (char *language)
- int config_getSerialNumber (OUT char *sNumber)
- int config_getSerialNumber_Len (OUT char *sNumber, IN_OUT int *sNumberLen)
- int config_setCmdTimeOutDuration (IN int millisecond)
- int config_setLEDController (int firmwareControlMSRLED, int firmwareControlICCLED)
- int config_getLEDController (int *firmwareControlMSRLED, int *firmwareControlICCLED)
- int config_setBeeperController (int firmwareControlBeeper)
- int config_getBeeperController (int *firmwareControlBeeper)
- int config_setEncryptionControl (int msr, int icc)
- int config_getEncryptionControl (int *msr, int *icc)
- int device_enablePassThrough (int enablePassThrough)
- int device_enableL100PassThrough (int enableL100PassThrough)
- int device_getL100PassThroughMode ()
- int device_enhancedPassthrough (IN BYTE *data, IN int dataLen)
- int device_setBurstMode (IN BYTE mode)
- int device_setPollMode (IN BYTE mode)
- int device_pollForToken (IN int timeout, OUT BYTE *respData, IN_OUT int *respDataLen)
- int device_setMerchantRecord (int index, int enabled, char *merchantID, char *merchantURL)
- int device_getMerchantRecord (IN int index, OUT BYTE *record)
- int device_getMerchantRecord_Len (IN int index, OUT BYTE *record, IN_OUT int *recordLen)
- int device_pollCardReader (OUT BYTE *status)
- int device_pollCardReader_Len (OUT BYTE *status, IN_OUT int *statusLen)
- int device_getSpectrumProKSN (IN int type, OUT BYTE *KSN)
- int device_getSpectrumProKSN_Len (IN int type, OUT BYTE *KSN, IN_OUT int *KSNLen)
- int device_calibrateParameters (BYTE delta)
- int device_getRTCDateTime (IN BYTE *dateTime, IN_OUT int *dateTimeLen)
- int device_setRTCDateTime (IN BYTE *dateTime, IN int dateTimeLen)
- int device_configureButtons (IN BYTE done, IN BYTE swipe, IN BYTE delay)
- int device_getButtonConfiguration (OUT BYTE *done, OUT BYTE *swipe, OUT BYTE *delay)

- int device_disableBlueLED ()
- int device_enableBlueLED (IN BYTE *data, IN int dataLen)
- int device_lcdDisplayClear ()
- int device_enableExternalLCDMessages (IN int enableExtLCDMsg)
- int device_enableRFAntenna (IN int enableAntenna)
- int device_turnOffYellowLED ()
- int device_turnOnYellowLED ()
- int device_buzzerOnOff ()
- int device_lcdDisplayLine1Message (IN BYTE *message, IN int messageLen)
- int device_lcdDisplayLine2Message (IN BYTE *message, IN int messageLen)
- int device_startRKI (const char *caPath)
- int device_startQRCodeScan (IN int _timeout)
- int device_stopQRCodeScan ()
- int device_startTakingPhoto (IN int _timeout)
- int device_stopTakingPhoto ()
- int device_getSDKWaitTime ()
- void device_setSDKWaitTime (int waitTime)
- int device_getThreadStackSize ()
- void device_setThreadStackSize (int threadSize)
- void device_toSDCard (int forSDCard)
- int icc_enable (IN int withNotification)
- int icc_disable ()
- int icc_powerOnICC (OUT BYTE *ATR, IN_OUT int *inLen)
- int icc_powerOffICC ()
- int icc_exchangeAPDU (IN BYTE *c_APDU, IN int cLen, OUT BYTE *reData, IN_OUT int *reLen)
- int icc_exchangeEncryptedAPDU (IN BYTE *c_APDU, IN int cLen, OUT BYTE *reData, IN_OUT int *reLen)
- int icc_getAPDU_KSN (OUT BYTE *KSN, IN_OUT int *inLen)
- int icc_getFunctionStatus (OUT int *enabled, OUT int *withNotification)
- int icc_getICCReaderStatus (OUT BYTE *status)
- int icc_getKeyFormatForICCDUKPT (OUT BYTE *format)
- int icc_getKeyTypeForICCDUKPT (OUT BYTE *type)
- int icc_setKeyFormatForICCDUKPT (IN BYTE format)
- int icc_setKeyTypeForICCDUKPT (IN BYTE type)
- int iso8583_get1987Handler (OUT DL_ISO8583_HANDLER *ISOHandler)
- int iso8583_get1993Handler (OUT DL_ISO8583_HANDLER *ISOHandler)
- int iso8583_get2003Handler (OUT DL_ISO8583_HANDLER *ISOHandler)
- int iso8583_getField (IN DL_UINT16 dataField, IN DL_ISO8583_HANDLER *ISOHandler, OUT DL_ISO8583_FIELD_DEF *field)
- int iso8583_initializeMessage (OUT DL_ISO8583_MSG *ISOMessage)
- int iso8583_getMessageField (IN DL_UINT16 dataField, IN DL_ISO8583_MSG *ISOMessage, OUT DL_ISO8583_MSG_FIELD *messageField)
- int iso8583_setMessageField (IN DL_UINT16 dataField, IN const DL_UINT8 *data, OUT DL_ISO8583_MSG *ISOMessage)
- int iso8583_removeMessageField (IN DL_UINT16 dataField, OUT DL_ISO8583_MSG *ISOMessage)
- int iso8583_packMessage (IN const DL_ISO8583_HANDLER *ISOHandler, IN const DL_ISO8583_MSG *ISOMessage, OUT DL_UINT8 *packedData, OUT DL_UINT16 *packedDataLength)
- int iso8583_unpackMessage (IN const DL_ISO8583_HANDLER *ISOHandler, IN const DL_UINT8 *packedData, IN DL_UINT16 packedDataLength, OUT DL_ISO8583_MSG *ISOMessage)
- int iso8583_freeMessage (IN DL_ISO8583_MSG *ISOMessage)
- int iso8583_serializeToXML (IN DL_ISO8583_HANDLER *ISOHandler, IN DL_ISO8583_MSG *ISOMessage, OUT BYTE *serializedMessage, OUT int *serializedMessageLength)
- int iso8583_deserializeFromXML (IN BYTE *serializedMessage, IN int serializedMessageLength, OUT DL_ISO8583_HANDLER *ISOHandler, OUT DL_ISO8583_MSG *ISOMessage)
- int iso8583_displayMessage (IN DL_ISO8583_HANDLER *ISOHandler, IN DL_ISO8583_MSG *ISOMessage)

- int lcd_resetInitialState ()
- int lcd_customDisplayMode (IN int enable)
- int lcd_setForeBackColor (IN BYTE *foreRGB, IN int foreRGBLen, IN BYTE *backRGB, IN int backRGBLen)
- int lcd_clearDisplay (IN BYTE control)
- int lcd_captureSignature (IN int timeout)
- int lcd_startSlideShow (IN char *files, IN int filesLen, IN int posX, IN int posY, IN int posMode, IN int touchEnable, IN int recursion, IN int touchTerminate, IN int delay, IN int loops, IN int clearScreen)
- int lcd_cancelSlideShow (OUT BYTE *statusCode, IN_OUT int *statusCodeLen)
- int lcd_setDisplayImage (IN char *file, IN int fileLen, IN int posX, IN int posY, IN int posMode, IN int touchEnable, IN int clearScreen)
- int lcd_setBackgroundImage (IN char *file, IN int fileLen, IN int enable)
- int lcd_displayText (IN int posX, IN int posY, IN int displayWidth, IN int displayHeight, IN int fontDesignation, IN int fontID, IN int screenPosition, IN char *displayText, OUT BYTE *graphicsID)
- int lcd_displayText_Len (IN int posX, IN int posY, IN int displayWidth, IN int displayHeight, IN int fontDesignation, IN int fontID, IN int screenPosition, IN char *displayText, OUT BYTE *graphicsID, IN_OUT int *graphicsIDLen)
- int lcd_displayParagraph (IN int posX, IN int posY, IN int displayWidth, IN int displayHeight, IN int fontDesignation, IN int fontID, IN int displayProperties, IN char *displayText)
- int lcd_displayButton (IN int posX, IN int posY, IN int buttonWidth, IN int buttonHeight, IN int fontDesignation, IN int fontID, IN int displayPosition, IN char *buttonLabel, IN int buttonTextColorR, IN int buttonTextColorG, IN int buttonTextColorB, IN int buttonBackgroundColorR, IN int buttonBackgroundColorG, IN int buttonBackgroundColorB, OUT BYTE *graphicsID)
- int lcd_displayButton_Len (IN int posX, IN int posY, IN int buttonWidth, IN int buttonHeight, IN int fontDesignation, IN int fontID, IN int displayPosition, IN char *buttonLabel, IN int buttonTextColorR, IN int buttonTextColorG, IN int buttonTextColorB, IN int buttonBackgroundColorR, IN int buttonBackgroundColorG, IN int buttonBackgroundColorB, OUT BYTE *graphicsID, IN_OUT int *graphicsIDLen)
- int lcd_createList (IN int posX, IN int posY, IN int numOfColumns, IN int numOfRows, IN int fontDesignation, IN int fontID, IN int verticalScrollArrowsVisible, IN int borderedListItems, IN int borderdScrollArrows, IN int touchSensitive, IN int automaticScrolling, OUT BYTE *graphicsID)
- int lcd_createList_Len (IN int posX, IN int posY, IN int numOfColumns, IN int numOfRows, IN int fontDesignation, IN int fontID, IN int verticalScrollArrowsVisible, IN int borderedListItems, IN int borderdScrollArrows, IN int touchSensitive, IN int automaticScrolling, OUT BYTE *graphicsID, IN_OUT int *graphicsIDLen)
- int lcd_addItemToList (IN BYTE *listGraphicsID, IN char *itemName, IN char *itemID, IN int selected)
- int lcd_getSelectedListItem (IN BYTE *listGraphicsID, OUT char *itemID)
- int lcd_getSelectedListItem_Len (IN BYTE *listGraphicsID, OUT char *itemID, IN_OUT int *itemIDLen)
- int lcd_clearEventQueue ()
- int lcd_getInputEvent (IN int timeout, OUT int *dataReceived, OUT BYTE *eventType, OUT BYTE *graphicsID, OUT BYTE *eventData)
- int lcd_getInputEvent_Len (IN int timeout, OUT int *dataReceived, OUT BYTE *eventType, IN_OUT int *eventTypeLen, OUT BYTE *graphicsID, IN_OUT int *graphicsIDLen, OUT BYTE *eventData, IN_OUT int *eventDataLen)
- int lcd_createInputField (IN BYTE *specs, IN int specsLen, OUT BYTE *graphicId)
- int lcd_createInputField_Len (IN BYTE *specs, IN int specsLen, OUT BYTE *graphicId, IN_OUT int *graphicIdLen)
- int lcd_getInputFieldValue (IN BYTE *graphicId, OUT BYTE *retData, IN_OUT int *retDataLen)
- int lcd_createScreen (IN char *screenName, IN int screenNameLen, OUT int *ScreenID)
- int lcd_destroyScreen (IN char *screenName, IN int screenNameLen)
- int lcd_getActiveScreen (OUT char *screenName, IN_OUT int *screenNameLen)
- int lcd_showScreen (IN char *screenName, IN int screenNameLen)
- int lcd_getButtonEvent (OUT int *screenID, OUT int *objectID, OUT char *screenName, IN_OUT int *screenNameLen, OUT char *objectName, IN_OUT int *objectNameLen, OUT int *isLongPress)
- int lcd_addButton (IN char *screenName, IN int screenNameLen, IN char *buttonName, IN int buttonNameLen, IN BYTE type, IN BYTE alignment, IN int xCord, IN int yCord, IN char *label, IN int labelLen, OUT IDTLCDItem *returnItem)
- int lcd_addEthernet (IN char *screenName, IN int screenNameLen, IN char *objectName, IN int objectNameLen, IN BYTE alignment, IN int xCord, IN int yCord, OUT IDTLCDItem *returnItem)

- int lcd_addLED (IN char *screenName, IN int screenNameLen, IN char *objectName, IN int objectNameLen, IN BYTE alignment, IN int xCord, IN int yCord, OUT IDTLCDItem *returnItem, IN BYTE *LED, IN int LEDLen)
- int lcd_addText (IN char *screenName, IN int screenNameLen, IN char *objectName, IN int objectNameLen, IN BYTE alignment, IN int xCord, IN int yCord, IN int width, IN int height, IN BYTE fontID, IN BYTE *color, IN int colorLen, IN char *label, IN int labelLen, OUT IDTLCDItem *returnItem)
- int lcd_addImage (IN char *screenName, IN int screenNameLen, IN char *objectName, IN int objectNameLen, IN BYTE alignment, IN int xCord, IN int yCord, IN char *filename, IN int filenameLen, OUT IDTLCDItem *returnItem)
- int lcd_addVideo (IN char *screenName, IN int screenNameLen, IN char *objectName, IN int objectNameLen, IN BYTE alignment, IN int xCord, IN int yCord, IN char *filename, IN int filenameLen, OUT IDTLCDItem *returnItem)
- int lcd_cloneScreen (IN char *screenName, IN int screenNameLen, IN char *cloneName, IN int cloneNameLen, OUT int *cloneID)
- int lcd_updateLabel (IN char *screenName, IN int screenNameLen, IN char *objectName, IN int objectNameLen, IN char *label, IN int labelLen)
- int lcd_updateColor (IN char *screenName, IN int screenNameLen, IN char *objectName, IN int objectNameLen, IN BYTE *color, IN int colorLen)
- int lcd_updatePosition (IN char *screenName, IN int screenNameLen, IN char *objectName, IN int objectNameLen, IN BYTE alignment, IN int new_xCord, IN int new_yCord)
- int lcd_removeItem (IN char *screenName, IN int screenNameLen, IN char *objectName, IN int objectNameLen)
- int lcd_storeScreenInfo ()
- int lcd_loadScreenInfo ()
- int lcd_clearScreenInfo ()
- int lcd_getAllScreens (IN_OUT int *screenNumbers, OUT IDTScreenInfo *screenInfo)
- int lcd_getAllObjects (IN char *screenName, IN int screenNameLen, IN_OUT int *objectNumbers, OUT IDTObjectInfo *objectInfo)
- int lcd_queryScreenbyName (IN char *screenName, IN int screenNameLen, OUT int *result)
- int lcd_queryObjectbyName (IN char *objectName, IN int objectNameLen, IN_OUT int *objectNumbers, OUT IDTScreenInfo *screenInfo)
- int lcd_queryScreenbyID (IN int screenID, OUT int *result, OUT int *screenName, IN_OUT int *screenNameLen)
- int lcd_queryObjectbyID (IN int objectID, OUT int *objectNumbers, OUT IDTScreenInfo *screenInfo)
- int lcd_setBacklight (IN BYTE backlightVal)
- int emv_getEMVKernelVersion (OUT char *version)
- int emv_getEMVKernelVersion_Len (OUT char *version, IN_OUT int *versionLen)
- int emv_getEMVKernelCheckValue (OUT BYTE *checkValue, IN_OUT int *checkValueLen)
- int emv_getEMVConfigurationCheckValue (OUT BYTE *checkValue, IN_OUT int *checkValueLen)
- void emv_setAutoAuthenticateTransaction (IN int authenticate)
- void emv_setAutoCompleteTransaction (IN int complete)
- int emv_getAutoAuthenticateTransaction ()
- int emv_getAutoCompleteTransaction ()
- void emv_allowFallback (IN int allow)
- void emv_setTransactionParameters (IN double amount, IN double amtOther, IN int type, IN int timeout, IN BYTE *tags, IN int tagsLen)
- int emv_startTransaction (IN double amount, IN double amtOther, IN int exponent, IN int type, IN int timeout, IN BYTE *tags, IN int tagsLen, IN int forceOnline)
- int emv_activateTransaction (IN int timeout, IN BYTE *tags, IN int tagsLen, IN int forceOnline)
- int emv_authenticateTransaction (IN BYTE *updatedTLV, IN int updatedTLVLen)
- int emv_authenticateTransactionWithTimeout (IN int timeout, IN BYTE *updatedTLV, IN int updatedTLVLen)
- int emv_completeTransaction (IN int commError, IN BYTE *authCode, IN int authCodeLen, IN BYTE *iad, IN int iadLen, IN BYTE *tlvScripts, IN int tlvScriptsLen, IN BYTE *tlv, IN int tlvLen)
- int emv_cancelTransaction ()
- int emv_retrieveTransactionResult (IN BYTE *tags, IN int tagsLen, IDTTransactionData *cardData)
- int emv_callbackResponseLCD (IN int type, byte selection)
- int emv_callbackResponseMSR (IN BYTE *MSR, IN_OUT int MSRLen)

- int emv_retrieveApplicationData (IN BYTE *AID, IN int AIDLen, OUT BYTE *tlv, IN_OUT int *tlvLen)
- int emv_setApplicationData (IN BYTE *name, IN int nameLen, IN BYTE *tlv, IN int tlvLen)
- int emv_setApplicationDataTLV (IN BYTE *tlv, IN int tlvLen)
- int emv_removeApplicationData (IN BYTE *AID, IN int AIDLen)
- int emv_removeAllApplicationData ()
- int emv_retrieveAIDList (OUT BYTE *AIDList, IN_OUT int *AIDListLen)
- int emv_retrieveTerminalData (OUT BYTE *tlv, IN_OUT int *tlvLen)
- int emv_setTerminalData (IN BYTE *tlv, IN int tlvLen)
- int emv_setTerminalMajorConfiguration (IN int configuration)
- int emv_removeTerminalData ()
- int emv_retrieveCAPK (IN BYTE *capk, IN int capkLen, OUT BYTE *key, IN_OUT int *keyLen)
- int emv_setCAPK (IN BYTE *capk, IN int capkLen)
- int emv_removeCAPK (IN BYTE *capk, IN int capkLen)
- int emv_removeAllCAPK ()
- int emv_retrieveCAPKList (OUT BYTE *keys, IN_OUT int *keysLen)
- int emv_retrieveTerminalID (OUT char *terminalID)
- int emv_retrieveTerminalID_Len (OUT char *terminalID, IN_OUT int *terminalIDLen)
- int emv_setTerminalID (IN char *terminalID)
- int emv_retrieveCRL (OUT BYTE *list, IN_OUT int *lssLen)
- int emv_setCRL (IN BYTE *list, IN int lsLen)
- int emv_removeCRL (IN BYTE *list, IN int lsLen)
- int emv_removeAllCRL ()
- int msr_clearMSRData ()
- int msr_getMSRData (OUT BYTE *reData, IN_OUT int *reLen)
- int msr_cancelMSRSwipe ()
- int msr_startMSRSwipe (IN int _timeout)
- void parseMSRData (IN BYTE *resData, IN int resLen, IN_OUT IDTMSRData *cardData)
- int msr_getKeyFormatForICCDUKPT (OUT BYTE *format)
- int msr_getKeyTypeForICCDUKPT (OUT BYTE *type)
- int msr_setKeyFormatForICCDUKPT (IN BYTE format)
- int msr_setKeyTypeForICCDUKPT (IN BYTE type)
- int msr_captureMode (int isBufferMode, int withNotification)
- int msr_flushTrackData ()
- int msr_setExpirationMask (int mask)
- int msr_getExpirationMask (BYTE *value)
- int msr_setClearPANID (BYTE val)
- int msr_getClearPANID (BYTE *value)
- int msr_setSwipeForcedEncryptionOption (int track1, int track2, int track3, int track3card0)
- int msr_getSwipeForcedEncryptionOption (BYTE *option)
- int msr_setSwipeMaskOption (int track1, int track2, int track3)
- int msr_getSwipeMaskOption (BYTE *option)
- int msr_disable ()
- int msr_getFunctionStatus (int *enable, int *isBufferMode, int *withNotification)
- int pin_getPIN (IN int mode, IN int PANSource, IN char *iccPAN, IN int iccPANLen, IN int startTimeout, IN int entryTimeout, IN char *language, IN int languageLen)
- int pin_cancelPINEntry ()
- int pin_setKeyValues (int mode)
- int pin_getEncryptedOnlinePIN (IN int keyType, IN int timeout)
- int pin_getPAN (IN int getCSC, IN int timeout)
- int pin_promptCreditDebit (IN char *currencySymbol, IN int currencySymbolLen, IN char *displayAmount, IN int displayAmountLen, IN int timeout, OUT BYTE *retData, IN_OUT int *retDataLen)
- int pin_getEncryptedPIN (int keyType, char *PAN, int PANLen, char *message, int messageLen, int timeout)
- int pin_promptForKeyInput (int messageID, int languageID, int maskInput, int minLen, int maxLen, int timeout)
- int pin_promptForAmountInput (int messageID, int languageID, int minLen, int maxLen, int timeout)
- int pin_getFunctionKey (int timeout)

- int pin_sendBeep (int frequency, int duration)
- int pin_capturePin (IN int timeout, IN int type, IN char *PAN, IN int PANLen, IN int minPIN, IN int maxPIN, IN char *message, IN int messageLen)
- int pin_capturePinExt (IN int type, IN char *PAN, IN int PANLen, IN int minPIN, IN int maxPIN, IN char *message, IN int messageLen, IN char *verify, IN int verifyLen)
- int pin_promptForNumericKeyWithSwipe (IN int timeout, IN BYTE function, IN int minLen, IN int maxLen, IN char *line1, IN int line1Len, IN char *line2, IN int line2Len, BYTE *signature, IN int signatureLen)
- int pin_promptForNumericKey (IN int timeout, IN int maskInput, IN int minLen, IN int maxLen, IN char *message, IN int messageLen, BYTE *signature, IN int signatureLen)
- int pin_inputFromPrompt (BYTE mask, BYTE preClearText, BYTE postClearText, int minLen, int maxLen, char *lang, BYTE promptID, char *defaultResponse, int defaultResponseLen, int timeout)
- int pin_promptForAmount (IN int timeout, IN int minLen, IN int maxLen, IN char *message, IN int messageLen, BYTE *signature, IN int signatureLen)
- int pin_getPanEntry (IN int csc, IN int expDate, IN int ADR, IN int ZIP, IN int mod10CK, IN int timeout, IN int encPANOnly)
- int lcd_savePrompt (int promptNumber, char *prompt, int promptLen)
- int lcd_displayPrompt (int promptNumber, int lineNumber)
- int lcd_displayMessage (int lineNumber, char *message, int messageLen)
- int lcd_enableBacklight (int enable)
- int lcd_getBacklightStatus (int *enabled)
- int ws_requestCSR (OUT RequestCSR *csr)
- int ws_loadSSLCert (IN char *name, IN int nameLen, IN char *dataDER, IN int dataDERLen)
- int ws_revokeSSLCert (IN char *name, IN int nameLen)
- int ws_deleteSSLCert (IN char *name, IN int nameLen)
- int ws_getCertChainType (OUT int *type)
- int ws_updateRootCertificate (IN char *name, IN int nameLen, IN char *dataDER, IN int dataDERLen, IN char *signature, IN int signatureLen)
- int ctls_startTransaction (IN double amount, IN double amtOther, IN int type, IN const int _timeout, IN BYTE *tags, IN int tagsLen)
- int ctls_activateTransaction (IN const int _timeout, IN BYTE *tags, IN int tagsLen)
- int ctls_cancelTransaction ()
- int ctls_retrieveApplicationData (IN BYTE *AID, IN int AIDLen, OUT BYTE *tlv, IN_OUT int *tlvLen)
- int ctls_setApplicationData (IN BYTE *tlv, IN int tlvLen)
- int ctls_removeApplicationData (IN BYTE *AID, IN int AIDLen)
- int ctls_removeAllApplicationData ()
- int ctls_retrieveAIDList (OUT BYTE *AIDList, IN_OUT int *AIDListLen)
- int ctls_retrieveTerminalData (OUT BYTE *tlv, IN_OUT int *tlvLen)
- int ctls_setTerminalData (IN BYTE *tlv, IN int tlvLen)
- int ctls_retrieveCAPK (IN BYTE *capk, IN int capkLen, OUT BYTE *key, IN_OUT int *keyLen)
- int ctls_setCAPK (IN BYTE *capk, IN int capkLen)
- int ctls_removeCAPK (IN BYTE *capk, IN int capkLen)
- int ctls_removeAllCAPK ()
- int ctls_retrieveCAPKList (OUT BYTE *keys, IN_OUT int *keysLen)
- int ctls_setConfigurationGroup (IN BYTE *tlv, IN int tlvLen)
- int ctls_getConfigurationGroup (IN int group, OUT BYTE *tlv, OUT int *tlvLen)
- int ctls_getAllConfigurationGroups (OUT BYTE *tlv, IN_OUT int *tlvLen)
- int ctls_removeConfigurationGroup (int group)
- int ctls_displayOnlineAuthResult (IN int statusCode, IN BYTE *TLV, IN int TLVLen)
- void parsePINBlockData (IN BYTE *resData, IN int resLen, IN_OUT IDTPINData *cardData)
- void parsePINData (IN BYTE *resData, IN int resLen, IN_OUT IDTPINData *cardData)

12.2.1 Detailed Description

Windows C++ API. Windows C++ Global API methods.

12.2.2 Macro Definition Documentation

12.2.2.1 #define IN

INPUT parameter.

12.2.2.2 #define IN_OUT

INPUT / OUTPUT PARAMETER.

12.2.2.3 #define OUT

OUTPUT parameter.

12.2.3 Typedef Documentation

12.2.3.1 typedef void(∗ ftpComm_callBack)(int, int, int)

Define the comm callback function to get FTP file transfer status

It should be passed as a parameter in a FTP request, Signature (int, int, int) = response code, current block, total blocks RESPONSE CODES: 100 = FILE DOWNLOAD STARTED 101 = FILE BLOCK XX OF XX RECEIVED 102 = FILE DOWNLOAD COMPLETED 103 = FILE DOWNLOAD TERMINATED PREMATURELY

12.2.3.2 typedef void(∗ httpComm_callBack)(BYTE ∗, int)

Define the comm callback function to get the async url data

It should be registered using the comm_registerHTTPCallback,

12.2.3.3 typedef void(∗ pCMR_callBack)(int, IDTCMRData ∗)

Define the camera callback function to get the image data

It should be registered using the device_registerCameraCallBk,

12.2.3.4 typedef void(∗ pCSFS_callBack)(BYTE status)

Define the card status and front switch callback function to get card and front switch status

It should be registered using the device_registerCardStatusFrontSwitchCallBk,

12.2.3.5 typedef void(∗ pEMV_callBack)(int, int, unsigned char ∗, int, IDTTransactionData ∗, EMV_Callback ∗, int)

Define the EMV callback function to get the transaction message/data/result.

It should be registered using the emv_registerCallBk,

12.2.3.6 typedef void(∗ pFW_callBack)(int, int, int, int, int)

Define the FW callback function to get the status of the firmware update

It should be registered using the device_registerFWCallBk,

12.2.3.7 typedef void(* pLCD_callBack)(int, IDTLCDItem *)

Define the LCD callback function to get the input LCDItem

It should be registered using the lcd_registerCallBk,

12.2.3.8 typedef void(* pLog_callback)(BYTE, char *)

Define the log callback function to receive log messages.

12.2.3.9 typedef void(* pMessageHotplug)(int, int)

Define the USB hot-plug callback function to monitor the info when plug in/out the reader.

It should be registered using the registerHotplugCallBk, The first integer parameter is device type, and the second integer parameter is either 0: Device Plugged Out or 1: Device Plugged In

12.2.3.10 typedef void(* pMSR_callBack)(int, IDTMSRData)

Define the MSR callback function to get the MSR card data

It should be registered using the msr_registerCallBk, this callback function is for backward compatibility

12.2.3.11 typedef void(* pMSR_callBackp)(int, IDTMSRData *)

Define the MSR callback function to get pointer to the MSR card data

It should be registered using the msr_registerCallBk, this callback function is recommended instead of pMSR_callBack

12.2.3.12 typedef void(* pPIN_callBack)(int, IDTPINData *)

Define the PINPad callback function to get the input PIN Pad data

It should be registered using the pin_registerCallBk,

12.2.3.13 typedef void(* pReadDataLog)(unsigned char *, int)

Define the read response callback function to monitor the reading response from the reader.

It should be registered using the registerLogCallBk,

12.2.3.14 typedef void(* pRKI_callBack)(int, char *)

Define the RKI callback function to get the status of the RKI

It should be registered using the device_registerRKICallBk,

12.2.3.15 typedef void(* pSendDataLog)(unsigned char *, int)

Define the send command callback function to monitor the sending command into the reader.

It should be registered using the registerLogCallBk,

12.2.3.16 typedef void(∗ v4Comm_callBack)(BYTE, BYTE, BYTE ∗, int)

Define the comm callback function to receive the V4 Protocol packets received by the device from an external source (IP/USB/RS-232) It should be registered using the comm_registerV4Callback, Data callback will contain command, sub-command, and data from V4 packet

12.2.4 Function Documentation

12.2.4.1 int config_getBeeperController (int ∗ *firmwareControlBeeper*)

Get the Beeper Controller Status - AUGUSTA Set the Beeper controlled Status by software or firmware

Parameters

firmwareControl-Beeper	1 means firmware control the beeper, 0 means software control beeper.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.2.4.2 int config_getEncryptionControl (int ∗ *msr,* int ∗ *icc*)

Get Encryption Control - AUGUSTA

Get Encryption Control to switch status between MSR and ICC/EMV function. Following Encryption status supported:

- MSR ON, ICC/EMV ON,
- MSR ON, ICC/EMV OFF,
- MSR OFF, ICC/EMV OFF,

Parameters

msr	• 1: enabled MSR with Encryption, • 0: disabled MSR with Encryption,
icc	• 1: enabled ICC with Encryption, • 0: disabled ICC with Encryption,

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.2.4.3 int config_getLEDController (int ∗ *firmwareControlMSRLED,* int ∗ *firmwareControlICCLED*)

Get the LED Controller Status - AUGUSTA Get the MSR / ICC LED controlled status by software or firmware NOTE: The ICC LED always controlled by software.

Parameters

firmwareControl-MSRLED	• 1: firmware control the MSR LED • 0: software control the MSR LED
firmwareControl-ICCLED	• 1: firmware control the ICC LED • 0: software control the ICC LED

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.2.4.4 int config_getModelNumber (OUT char * *sNumber*)

DEPRECATED : please use config_getModelNumber_Len(OUT char* sNumber, IN_OUT int *sNumberLen)

Polls device for Model Number

Parameters

sNumber	Returns Model Number; needs to have at least 64 bytes of memory

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.2.4.5 int config_getModelNumber_Len (OUT char * *sNumber,* IN_OUT int * *sNumberLen*)

Polls device for Model Number

Parameters

sNumber	Returns Model Number
sNumber	length of Model Number

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.2.4.6 int config_getSerialNumber (OUT char * *sNumber*)

DEPRECATED : please use config_getSerialNumber_Len(OUT char* sNumber, IN_OUT int *sNumberLen)

Polls device for Serial Number

Parameters

sNumber	Returns Serial Number; needs to have at least 64 bytes of memory

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.2.4.7 int config_getSerialNumber_Len (OUT char * *sNumber,* IN_OUT int * *sNumberLen*)

Polls device for Serial Number

Parameters

sNumber	Returns Serial Number
sNumberLen	Length of Serial Number

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.2.4.8 int config_setBeeperController (int *firmwareControlBeeper*)

Set the Beeper Controller - AUGUSTA Set the Beeper controlled by software or firmware

Parameters

firmwareControl-Beeper	1 means firmware control the beeper, 0 means software control beeper.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.2.4.9 int config_setCmdTimeOutDuration (IN int *millisecond*)

Set the timeout duration for regular commands The new timeout value will affect all the functions actually send (sync) commands that doesn't need to wait for a callback function, such as device_getFirmwareVersion(), device_pingDevice(), device_SendDataCommandNEO(), device_enablePassThrough(), device_setBurstMode(), device_setPollMode(), device_updateFirmware() . . . etc.

Parameters

millisecond	timeout value in milliseconds

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString

12.2.4.10 int config_setEncryptionControl (int *msr,* int *icc*)

Set Encryption Control - AUGUSTA

Set Encryption Control to switch status between MSR and ICC/EMV function. Following Encryption status supported:

- MSR ON, ICC/EMV ON,
- MSR ON, ICC/EMV OFF,
- MSR OFF, ICC/EMV OFF,

Parameters

msr	• 1: enable MSR with Encryption, • 0: disable MSR with Encryption,
icc	• 1: enable ICC with Encryption, • 0: disable ICC with Encryption,

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.2.4.11 int config_setLEDController (int *firmwareControlMSRLED,* int *firmwareControlICCLED*)

Set the LED Controller - AUGUSTA Set the MSR / ICC LED controlled by software or firmware NOTE: The ICC LED always controlled by software.

Parameters

firmwareControl-MSRLED	• 1: firmware control the MSR LED • 0: software control the MSR LED
firmwareControl-ICCLED	• 1: firmware control the ICC LED • 0: software control the ICC LED

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.2.4.12 int ctls_activateTransaction (IN const int *_timeout,* IN BYTE * *tags,* IN int *tagsLen*)

Start CTLS Transaction Request

Authorizes the CTLS 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 stream. Example, tag 9F02 with amount 0x000000000100 would be 0x9F0206000000000100
tagsLen	The length of tags data buffer.

>>>>>IMPORTANT: parameters for amount and amtOther MUST BE PASSED AS A DOUBLE VALUE WITH DECIMAL POINT. Example, do not pass 1, but instead pass 1.0 or 1.00. Otherwise, results will be unpredictable

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString() Note: if auto poll is on, it will returm the error IDG_P2_STATUS_CODE_COMMAND_NOT_ALLOWED

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 DF01. 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

12.2.4.13 int ctls_cancelTransaction ( )

Cancel EMV Transaction

Cancels the currently executing EMV transaction.

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.2.4.14 int ctls_displayOnlineAuthResult (IN int *statusCode,* IN BYTE * *TLV,* IN int *TLVLen*)

Display Online Authorization Result Use this command to display the status of an online authorization request on the reader's display (OK or NOT OK). Use this command after the reader sends an online request to the issuer.

Parameters

statusCode	1 = OK, 0 = NOT OK, 2 = ARC response 89 for Interac
TLV	Optional TLV for AOSA
TLVLen	TLV Length

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.2.4.15 int ctls_getAllConfigurationGroups (OUT BYTE * *tlv*, IN_OUT int * *tlvLen*)

Retrieve All Configuration Groups

Returns all the Configuration Groups installed on the terminal for CTLS

Parameters

tlv	The TLV elements data
tlvLen	the length of tlv data buffer.

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.2.4.16 int ctls_getConfigurationGroup (IN int *group*, OUT BYTE * *tlv*, OUT int * *tlvLen*)

Get Configuration Group

Retrieves the Configuration for the specified Group.

Parameters

group	Configuration Group
tlv	return data
tlvLen	the length of tlv data buffer

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.2.4.17 int ctls_removeAllApplicationData ( )

Remove All Application Data

Removes all the Application Data

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.2.4.18 int ctls_removeAllCAPK ( )

Remove All Certificate Authority Public Key

Removes all the CAPK

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.2.4.19 int ctls_removeApplicationData (IN BYTE * *AID,* IN int *AIDLen*)

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
AIDLen	the length of AID data buffer

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.2.4.20 int ctls_removeCAPK (IN BYTE * *capk,* IN int *capkLen*)

Remove Certificate Authority Public Key

Removes the CAPK as specified by the RID/Index

Parameters

capk	6 byte CAPK = 5 bytes RID + 1 byte INDEX
capkLen	the length of capk data buffer

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.2.4.21 int 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	Values can be parsed with device_getIDGStatusCodeString()

12.2.4.22 int ctls_retrieveAIDList (OUT BYTE * *AIDList,* IN_OUT int * *AIDListLen*)

Retrieve AID list

Returns all the AID names installed on the terminal for CTLS. .

Parameters

AIDList	array of AID name byte arrays
AIDListLen	the length of AIDList array buffer

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.2.4.23 int ctls_retrieveApplicationData (IN BYTE * *AID,* IN int *AIDLen,* OUT BYTE * *tlv,* IN_OUT int * *tlvLen*)

Retrieve Application Data by AID

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

Parameters

AID	Name of ApplicationID. Must be between 5 and 16 bytes
AIDLen	the length of AID data buffer.
tlv	The TLV elements of the requested AID
tlvLen	the length of tlv data buffer.

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.2.4.24 int ctls_retrieveCAPK (IN BYTE * *capk,* IN int *capkLen,* OUT BYTE * *key,* IN_OUT int * *keyLen*)

Retrieve Certificate Authority Public Key

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

Parameters

capk	6 bytes CAPK = 5 bytes RID + 1 byte Index
capkLen	the length of capk data buffer
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.

keyLen	the length of key data buffer

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.2.4.25 int ctls_retrieveCAPKList (OUT BYTE * *keys*, IN_OUT int * *keysLen*)

Retrieve the Certificate Authority Public Key list

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

Parameters

keys	[key1][key2]...[keyn], each key 6 bytes where key = 5 bytes RID + 1 byte index
keysLen	the length of keys data buffer

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.2.4.26 int ctls_retrieveTerminalData (OUT BYTE * *tlv*, IN_OUT int * *tlvLen*)

Retrieve Terminal Data

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

Parameters

tlv	Response returned as a TLV
tlvLen	the length of tlv data buffer

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.2.4.27 int ctls_setApplicationData (IN BYTE * *tlv*, IN int *tlvLen*)

Set Application Data by AID

Sets the Application Data for CTLS as specified by TLV data

Parameters

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

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

Parameters

tlvLen	the length of tlv data buffer

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.2.4.28 int ctls_setCAPK (IN BYTE * *capk,* IN int *capkLen*)

Set Certificate Authority Public Key

Sets the CAPK as specified by the CAKey structure

Parameters

capk	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.
capkLen	the length of capk data buffer

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.2.4.29 int ctls_setConfigurationGroup (IN BYTE * *tlv*, IN int *tlvLen*)

Set Configuration Group

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

Parameters

tlv	Configuration Group Data in TLV format The first tag of the TLV data must be the group number (FFE4 or DFEE2D). A second tag must exist
tlvLen	the length of tlv data buffer

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.2.4.30 int ctls_setTerminalData (IN BYTE * *tlv*, IN int *tlvLen*)

Set Terminal Data

Sets the Terminal Data for CTLS as specified by the TLV. The first TLV must be Configuration Group Number (Tag FFE4). The terminal global data is group 0, so the first TLV would be FFE40100. Other groups can be defined using this method (1 or greater), and those can be retrieved with ctls_getConfigurationGroup(int group), and deleted with ctls_removeConfigurationGroup(int group). You cannot delete group 0

Parameters

tlv	TerminalData configuration file
tlvLen	the length of tlv data buffer

Return values

RETURN_CODE	Values can be parsed with device_getIDGStatusCodeString()

12.2.4.31 int ctls_startTransaction (IN double *amount,* IN double *amtOther,* IN int *type,* IN const int *_timeout,* IN BYTE * *tags,* IN int *tagsLen*)

Start CTLS Transaction Request

Authorizes the CTLS transaction for an ICC card

The tags will be returned in the callback routine.

Parameters

amount	Transaction amount value (tag value 9F02) - SEE IMPORTANT NOTE BELOW
amtOther	Other amount value, if any (tag value 9F03) - SEE IMPORTANT NOTE BELOW
type	Transaction type (tag value 9C).
timeout	Timeout value in seconds.
tags	Any other tags to be included in the request. Passed as TLV stream. Example, tag 9F0C with amount 0x000000000100 would be 0x9F0C06000000000100 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.
tagsLen	The length of tags data buffer.

>>>>>IMPORTANT: parameters for amount and amtOther MUST BE PASSED AS A DOUBLE VALUE WITH DECIMAL POINT. Example, do not pass 1, but instead pass 1.0 or 1.00. Otherwise, results will be unpredictable

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString() Note: if auto poll is on, it will returm the error IDG_P2_STATUS_CODE_COMMAND_NOT_ALLOWED

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 DF01. 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

12.2.4.32 int device_activateTransaction (IN const int *_timeout,* IN BYTE * *tags,* IN int *tagsLen*)

Start Transaction Request

Authorizes the transaction for an MSR/CTLS/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 a TLV. Example, tag 9F02 with amount 0x000000000100 would be 0x9F0206000000000100 Be sure to include 9F02 (amount)and9-C (transaction type).
tagsLen	The length of tags data buffer.

>>>>>IMPORTANT: parameters for amount and amtOther MUST BE PASSED AS A DOUBLE VALUE WITH DECIMAL POINT. Example, do not pass 1, but instead pass 1.0 or 1.00. Otherwise, results will be unpredictable

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString() Note: if auto poll is on, it will returm the error IDG_P2_STATUS_CODE_COMMAND_NOT_ALLOWED

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 device_startTransaction. Tag FFEE06 must contain tag 9F26 and 9F22, and can optionanally contain tags 9F2B and DF01. 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

12.2.4.33 int device_buzzerOnOff ( )

Use this function to make the buzzer beep once

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.2.4.34 int device_calibrateParameters (BYTE *delta*)

Calibrate reference parameters

Parameters

delta	Delta value (0x02 standard default value)

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.2.4.35 int device_cancelTransaction ( )

Cancel Transaction

Cancels the currently executing transaction.

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.2.4.36 int device_cancelTransactionSilent (int *enable*)

Cancel Transaction Silent

Cancel transaction with or without showing the LCD message

Parameters

enable	0: With LCD message 1: Without LCD message

Returns

success or error code. Values can be parsed with device_getIDGStatusCodeString

12.2.4.37 int device_close ( )

Close the device

Returns

RETURN_CODE: 0: success, 0x0A: failed

12.2.4.38 int device_configureButtons (IN BYTE *done,* IN BYTE *swipe,* IN BYTE *delay*)

Configures the buttons on the ViVOpay Vendi reader

Parameters

done	0x01: the Done switch is enabled 0x00: the Done switch is disabled
swipe	0x01: the Swipe Card switch is enabled 0x00: the Swipe Card switch is disabled
delay	an unsigned delay value (<= 30) in seconds

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.2.4.39 int device_controlBeep (int *index,* int *frequency,* int *duration*)

Control Beep - AUGUSTA

Controls the Beeper

Parameters

index	For Augusta, must be set to 1 (only one beeper)
frequency	Frequency, range 1000-20000 (suggest minimum 3000)
duration	Duration, in milliseconds (range 1 - 65525)

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.2.4.40 int device_controlIndicator (IN int *indicator,* IN int *enable*)

Control Indicators

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

Parameters

indicator	description as follows: • 00h: ICC LED • 01h: Blue MSR • 02h: Red MSR • 03h: Green MSR
enable	TRUE = ON, FALSE = OFF

Returns

success or error code. Values can be parsed with device_getResponseCodeString

See Also

ErrorCode

12.2.4.41 int device_controlLED (byte *indexLED,* byte *control,* int *intervalOn,* int *intervalOff*)

Control MSR LED - AUGUSTA

Controls the LED for the MSR

Parameters

indexLED	For Augusta, must be set to 1 (MSR LED)
control	LED Status: • 00: OFF • 01: RED Solid • 02: RED Blink • 11: GREEN Solid • 12: GREEN Blink • 21: BLUE Solid • 22: BLUE Blink
intervalOn	Blink interval ON, in ms (Range 200 - 2000)
intervalOff	Blink interval OFF, in ms (Range 200 - 2000)

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

intervalOn = 500, int intervalOff = 500

12.2.4.42 int device_controlLED_ICC (int *controlMode,* int *interval*)

Control ICC LED - AUGUSTA

Controls the LED for the ICC card slot

Parameters

controlMode	0 = off, 1 = solid, 2 = blink
interval	Blink interval, in ms (500 = 500 ms)

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.2.4.43 int device_controlLED_MSR (byte *control,* int *intervalOn,* int *intervalOff*)

Control the MSR LED - AUGUSTA

Controls the MSR / ICC LED This API not recommended to control ICC LED

Parameters

control	• 0x00 = off, • 0x01 = RED Solid, • 0x02 = RED Blink, • 0x11 = GREEN Solid, • 0x12 = GREEN Blink, • 0x21 = BLUE Solid, • 0x22 = BLUE Blink,
intervalOn	Blink interval on time last, in ms (500 = 500 ms, valid from 200 to 2000)
intervalOff	Blink interval off time last, in ms (500 = 500 ms, valid from 200 to 2000)

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

int intervalOn = 500, int intervalOff = 500)

12.2.4.44 int device_controlUserInterface (IN BYTE * *values*)

Control User Interface - NEO only

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 device_getIDGStatusCodeString()

12.2.4.45 int device_createDirectory (IN char * *directoryName,* IN int *directoryNameLen*)

Create Directory This command adds a subdirectory to the indicated path.

Parameters

directoryName	Directory Name. The data for this command is ASCII string with the complete path and directory name you want to create. You do not need to specify the root directory. Indicate subdirectories with a forward slash (/).
directoryNameLen	Directory Name Length.

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.2.4.46 int device_deleteDirectory (IN char * *dirName,* IN int *dirNameLen*)

Delete Directory This command deletes an empty directory. For NEO 2 devices, it will delete the directory even the directory is not empty.

Parameters

dirName	Complete path of the directory you want to delete. You do not need to specify the root directory. Indicate subdirectories with a forward slash (/). For NEO 2 devices, to delete the root directory, simply pass "" with 0 for dirNameLen.
dirNameLen	Directory Name Length.

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.2.4.47 int device_deleteFile (IN char * *fileName,* IN int *fileNameLen*)

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 (/).
filenameLen	File Name Length.

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.2.4.48 int device_disableBlueLED ( )

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

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.2.4.49 int device_enableBlueLED (IN BYTE * *data,* IN int *dataLen*)

Use this function to control the blue LED behavior on the Vendi reader

Parameters

data	Sequence data Byte 0 (Cycle): 0 = Cycle once, 1 = Repeat Byte 1 (LEDs): LED State Bitmap Byte 2-3 (Duration): Given in multiples of 10 millisecond Byte 4 (LED): LED State Bitmap Byte 5-6 (Duration): Given in multiples of 10 millisecond Byte 7-24 (Additional LED/Durations): Define up to 8 LED and duration pairs

LED State Bitmap: Bit 8: Left blue LED, 0 = off, 1 = on Bit 7: Center Blue LED, 0 = off, 1 = on Bit 6: Right Blue LED0 = off, 1 = on Bit 5: Yellow LED, 0 = off, 1 = on Bit 4: Reserved for future use Bit 3: Reserved for future use Bit 2: Reserved for future use Bit 1: Reserved for future use

Parameters

dataLen	Length of the sequence data: 0 or 4 to 25 bytes

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.2.4.50 int device_enableExternalLCDMessages (IN int *enableExtLCDMsg*)

Enable or disable the external LCD message It will turn off the external LCD messages including EMV transactions. (For the users who only need MSR and/or CTLS transactions.) The function only works for VP5300

Parameters

enableExtLCD-Msg	1 = ON, 0 = OFF

Returns

success or error code. Values can be parsed with device_getIDGStatusCodeString

See Also

ErrorCode

12.2.4.51 int device_enableL100PassThrough (int *enableL100PassThrough*)

Enable L100 Pass Through

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

Parameters

enableL100-PassThrough	1 = pass through ON, 0 = pass through OFF

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.2.4.52 int device_enablePassThrough (int *enablePassThrough*)

Start Remote Key Injection - AUGUSTA

Starts a remote key injection request with IDTech RKI servers. This function is reserved and not implemented.

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString Enable Pass Through - NEO

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

Parameters

enablePass-Through	1 = pass through ON, 0 = pass through OFF

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.2.4.53 int device_enableRFAntenna (IN int *enableAntenna*)

Enable or disable the RF Antenna

Parameters

enableAntenna	1 = ON, 0 = OFF

Returns

success or error code. Values can be parsed with device_getIDGStatusCodeString

See Also

ErrorCode

12.2.4.54 int device_enhancedPassthrough (IN BYTE * *data,* IN int *dataLen*)

Enables pass through mode for ICC. Required when direct ICC commands are required (power on/off ICC, exchange APDU)

Parameters

data	The data includes Poll Timeout, Flags, Contact Interface to Use, Beep Indicator, LED Status, and Display Strings.
dataLen	length of data

Returns

success or error code. Values can be parsed with device_getIDGStatusCodeString

See Also

ErrorCode

12.2.4.55 int device_enterStopMode ( )

Enter Stop Mode

Set device enter to stop 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

12.2.4.56 int device_getButtonConfiguration (OUT BYTE * *done,* OUT BYTE * *swipe,* OUT BYTE * *delay*)

Reads the button configuration from the ViVOpay Vendi reader

Parameters

done	0x01: the Done switch is enabled 0x00: the Done switch is disabled
swipe	0x01: the Swipe Card switch is enabled 0x00: the Swipe Card switch is disabled
delay	an unsigned delay value in seconds

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.2.4.57 int device_getCurrentDeviceType ( )

Get current active device type

Returns

: return the device type defined as DEVICE_TYPE in the IDTDef.h

12.2.4.58 int device_getDateTime (OUT BYTE * *dateTime*)

DEPRECATED : please use device_getDateTime_Len(OUT BYTE* dateTime, IN_OUT int *dateTimeLen)

Polls device for Date and Time

Parameters

dateTime	Response returned of Date and Time; needs to have at least 6 bytes of memory

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.2.4.59 int device_getDateTime_Len (OUT BYTE * *dateTime,* IN_OUT int * *dateTimeLen*)

Polls device for Date and Time

Parameters

dateTime	Response returned of Date and Time
dateTime	Length of Date and Time

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.2.4.60 int device_getDeviceMemoryUsageInfo (OUT int * *freeHeapSize,* OUT int * *notFreedBlockCnt,* OUT int * *minEverFreeHeapSize*)

Get Device Memory Usage Information

Parameters

freeHeapSize	Free Heap Size: Available heap size
notFreedBlock-Cnt	Memory Not Freed Block Count: Memory in use block count
minEverFree-HeapSize	Minimum Ever Free Heap Size: The lowest ever available heap size

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.2.4.61 int device_getDriveFreeSpace (OUT int * *free*, OUT int * *used*)

Drive Free Space This command returns the free and used disk space on the flash drive.

Parameters

free	Free bytes available on device
used	Used bytes on on device

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.2.4.62 int device_getDRS (BYTE * *codeDRS*, int * *codeDRSLen*)

Get DRS Status - AUGUSTA TTK

Gets the status of DRS(Destructive Reset).

Parameters

codeDRS	the data format is [DRS SourceBlk Number] [SourceBlk1] ... [SourceBlkN] [DRS SourceBlk Number] is 2 bytes, format is NumL NumH. It is Number of [SourceBlkX] [SourceBlkX] is n bytes, Format is [SourceID] [SourceLen] [SourceData] [SourceID] is 1 byte [SourceLen] is 1 byte, it is length of [SourceData]

[SourceID] [SourceLen] [SourceData] 00 1 01 - Application Error 01 1 01 - Application Error 02 1 0x01 - EMV L2 Configuration Check Value Error 0x02 - Future Key Check Value Error 10 1 01 - Battery Error 11 1 Bit 0 - Tamper Switch 1 (0-No, 1-Error) Bit 1 - Tamper Switch 2 (0-No, 1-Error) Bit 2 - Tamper Switch 3 (0-No, 1-Error) Bit 3 - Tamper Switch 4 (0-No, 1-Error) Bit 4 - Tamper Switch 5 (0-No, 1-Error) Bit 5 - Tamper Switch 6 (0-No, 1-Error)

12 1 01 - TemperatureHigh or Low 13 1 01 - Voltage High or Low 1F 4 Reg31∼24bits, Reg23∼16bits, Reg15∼8bits, Reg7∼0bits

Parameters

codeDRSLen	the length of codeDRS

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString() Note: Only support TTK devices

12.2.4.63 int device_getFirmwareVersion (OUT char * *firmwareVersion*)

DEPRECATED : please use device_getFirmwareVersion_Len(OUT char* firmwareVersion, IN_OUT int *firmware-VersionLen)

Polls device for Firmware Version

Parameters

firmwareVersion	Response returned of Firmware Version; needs to have at least 128 bytes of memory

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.2.4.64 int device_getFirmwareVersion_Len (OUT char * *firmwareVersion*, IN_OUT int * *firmwareVersionLen*)

Polls device for Firmware Version

Parameters

firmwareVersion	Response returned of Firmware Version
firmwareVersion-Len	Length of Firmware Version

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.2.4.65 void device_getIDGStatusCodeString (IN int *returnCode*, OUT char * *despcrition*)

Review the return code description.

Parameters

returnCode	the response result.
description	

Return values

the	string for description of response result

- 0: "no error, beginning task";
- 1: "no response from reader";
- 2: "invalid response data";
- 01: " Incorrect Header Tag";
- 02: " Unknown Command";
- 03: " Unknown Sub-Command";
- 04: " CRC Error in Frame";
- 05: " Incorrect Parameter";
- 06: " Parameter Not Supported";
- 07: " Mal-formatted Data";
- 08: " Timeout";
- 0A: " Failed / NACK";
- 0B: " Command not Allowed";
- 0C: " Sub-Command not Allowed";

- 0D: " Buffer Overflow (Data Length too large for reader buffer)";
- 0E: " User Interface Event";
- 10: " Need clear firmware(apply in boot loader only)";
- 11: " Communication type not supported, VT-1, burst, etc. Need encrypted firmware (apply in boot loader only)";
- 12: " Secure interface is not functional or is in an intermediate state.";
- 13: " Data field is not mod 8";
- 14: " Pad 0x80 not found where expected";
- 15: " Specified key type is invalid";
- 16: " Could not retrieve key from the SAM (InitSecureComm)";
- 17: " Hash code problem";
- 18: " Could not store the key into the SAM (InstallKey)";
- 19: " Frame is too large";
- 1A: " Unit powered up in authentication state but POS must resend the InitSecureComm command";
- 1B: " The EEPROM may not be initialized because SecCommInterface does not make sense";
- 1C: " Problem encoding APDU Module-Specific Status Codes ";
- 20: " Unsupported Index (ILM) SAM Transceiver error - problem communicating with the SAM (Key Mgr)";
- 21: " Unexpected Sequence Counter in multiple frames for single bitmap (ILM)Length error in data returned from the SAM (Key Mgr) - 22: " Improper bit map (ILM)"; - 23: " Request Online Authorization"; - 24: " ViVOCard3 raw data read successful"; - 25: " Message index not available (ILM) ViVOcomm activate transaction card type (ViVOcomm)"; - 26: " Version Information Mismatch (ILM)"; - 27: " Not sending commands in correct index message index (ILM)"; - 28: " Time out or next expected message not received (ILM)"; - 29: " ILM languages not available for viewing (ILM)"; - 2A: " Other language not supported (ILM)"; - 41: " from 41 to 4F, Module-specific errors for Key Manager";
- 50: " Auto-Switch OK";
- 51: " Auto-Switch failed";
- 70: " Antenna Error 80h Use another card";
- 81: " Insert or swipe card";
- 90: " Data encryption Key does not exist";
- 91: " Data encryption Key KSN exhausted";

12.2.4.66 int device_getKeyStatus (int * *newFormat,* BYTE * *status,* int * *statusLen*)

Get Key Status

Gets the status of loaded keys

Parameters

status	newFormat for Augusta and miniSmartII only 1: new format of key status 0: reserved format for support previous device
status	For L100, Augusta and miniSmartII: When the newFormat is 0, data format as follows. For Augusta and miniSmartII: byte 0: PIN DUKPT Key, Does not support, always 0 byte 1: PIN Master Key, Does not support, always 0 byte 2: PIN Session Key, Does not support, always 0 byte 3: Account/MSR DUKPT Key, 1 Exists, 0 None, 0xFF STOP byte 4: Account/ICC DUKPT Key, 1 Exists, 0 None, 0xFF STOP byte 5: Admin DUKPT Key, 1 Exists, 0 None, 0xFF STOP For L100: byte 0: PIN DUKPT Key byte 1: PIN Master Key byte 2: Standard PIN Session Key byte 3: Desjardins PIN Session Key byte 4: Account/MSR DUKPT Key, 1 Exists, 0 None, 0xFF STOP, Does not support, always 0 byte 5: Account/ICC DUKPT Key, 1 Exists, 0 None, 0xFF STOP, Does not support, always 0 byte 6: Admin DUKPT Key, 1 Exists, 0 None, 0xFF STOP byte 7: Data DUKPT Key, 1 Exists, 0 None, 0xFF STOP byte 8: MAC DUKPT Key, 1 Exists, 0 None, 0xFF STOP

when the newFormat is 1, data format as follows. [Block Length] [KeyStatusBlock1] [KeyStatusBlock2]...[KeyStatusBlockN] Where: [Block Length] is 2 bytes, format is Len_L Len_H, is KeyStatusBlock Number [KeyStatusBlockX> is 4 bytes, format is [Key Index and Key Name] [key slot] [key status]: [Key Index and Key Name] is 1 byte. Please refer to following table 0x14 LCL-KEK to Encrypt Other Keys 0x02 Data encryption Key to Encrypt ICC/MSR 0x05 MAC DUKPT Key for Host-Device - MAC Verification 0x05 MTK DUKPT Key for TTK Self-Test 0x0C RKI-KEK for Remote Key Injection [key slot] is 2 bytes. Range is 0 - 9999 the MTK DUKPT Key slot is 16, the others are all 0 [key status] is 1 byte. 0 - Not Exist 1 - Exist 0xFF - (Stop. Only Valid for DUKPT Key) For NEO2 and SREDKey2: Each unit of three bytes represents one key's parameters (index and slot). Key Name Index (1 byte): 0x14 - LCL-KEK 0x01 - Pin encryption Key (NEO2 only) 0x02 - Data encryption Key 0x05 - MAC DUKPT Key 0x0A - PCI Pairing Key (NEO2 only) Key Slot (2 bytes): Indicate different slots of a certain Key Name Example: slot =5 (0x00 0x05), slot=300 (0x01 0x2C) For BTPay380, slot is always 0 For example, 0x14 0x00 0x00 0x02 0x00 0x00 0x0A 0x00 0x00 will represent [KeyNameIndex=0x14,KeySlot=0x0000], [KeyNameIndex=0x02,KeySlot=0x0000] and [KeyNameIndex=0x0A,KeySlot=0x0000]

Parameters

statusLen	the length of status

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.2.4.67 int device_getL100PassThroughMode ( )

Get L100 Pass Through Mode

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

Returns

RETURN_CODE: return 1 if L100 Pass Through Mode is TRUE, 0 if L100 Pass Through Mode is FALSE

12.2.4.68 int device_getMerchantRecord (IN int *index*, OUT BYTE * *record*)

DEPRECATED : please use device_getMerchantRecord_Len(IN int index, OUT BYTE * record, IN_OUT int *recordLen)

Get Merchant Record

Gets the merchant record for the device.

Parameters

index	Merchant Record index, valid values 1-6
record;	needs to have at least 99 bytes of memory response data from reader. Merchant Record Index: 1 byte enabled: 1 byte Merchant ID: 32 bytes Length of Merchant URL: 1 byte Merchant URL: 64 bytes

Returns

success or error code. Values can be parsed with device_getIDGStatusCodeString()

See Also

ErrorCode

12.2.4.69 int device_getMerchantRecord_Len (IN int *index,* OUT BYTE * *record,* IN_OUT int * *recordLen*)

Get Merchant Record

Gets the merchant record for the device.

Parameters

index	Merchant Record index, valid values 1-6
record	response data from reader. Merchant Record Index: 1 byte enabled: 1 byte Merchant ID: 32 bytes Length of Merchant URL: 1 byte Merchant URL: 64 bytes
recordLen	Length of record

Returns

success or error code. Values can be parsed with device_getIDGStatusCodeString()

See Also

ErrorCode

12.2.4.70 void device_getResponseCodeString (IN int *returnCode,* OUT char * *despcrition*)

Review the return code description.

Parameters

returnCode	the response result.
description	

Return values

the	string for description of response result

- 0: "no error, beginning task";
- 1: "no response from reader";
- 2: "invalid response data";
- 3: "time out for task or CMD";
- 4: "wrong parameter";

- 5: "SDK is doing MSR or ICC task";
- 6: "SDK is doing PINPad task";
- 7: "SDK is doing CTLS task";
- 8: "SDK is doing EMV task";
- 9: "SDK is doing Other task";
- 10: "err response or data";
- 11: "no reader attached";
- 12: "mono audio is enabled";
- 13: "did connection";
- 14: "audio volume is too low";
- 15: "task or CMD be canceled";
- 16: "UF wrong string format";
- 17: "UF file not found";
- 18: "UF wrong file format";
- 19: "Attempt to contact online host failed";
- 20: "Attempt to perform RKI failed";
- 22: "Buffer size is not enough";
- 0X300: "Key Type(TDES) of Session Key is not same as the related Master Key.";
- 0X400: "Related Key was not loaded.";
- 0X500: "Key Same.";
- 0X501: "Key is all zero";
- 0X502: "TR-31 format error";
- 0X702: "PAN is Error Key.";
- 0X705: "No Internal MSR PAN (or Internal MSR PAN is erased timeout)";
- 0X0C01: "Incorrect Frame Tag";
- 0X0C02: "Incorrect Frame Type";
- 0X0C03: "Unknown Frame Type";
- 0X0C04: "Unknown Command";
- 0X0C05: "Unknown Sub-Command";
- 0X0C06: "CRC Error";
- 0X0C07: "Failed";
- 0X0C08: "Timeout";
- 0X0C0A: "Incorrect Parameter";
- 0X0C0B: "Command Not Supported";
- 0X0C0C: "Sub-Command Not Supported";
- 0X0C0D: "Parameter Not Supported / Status Abort Command";

- 0X0C0F: "Sub-Command Not Allowed";
- 0X0D01: "Incorrect Header Tag";
- 0X0D02: "Unknown Command";
- 0X0D03: "Unknown Sub-Command";
- 0X0D04: "CRC Error in Frame";
- 0X0D05: "Incorrect Parameter";
- 0X0D06: "Parameter Not Supported";
- 0X0D07: "Mal-formatted Data";
- 0X0D08: "Timeout";
- 0X0D0A: "Failed / NACK";
- 0X0D0B: "Command not Allowed";
- 0X0D0C: "Sub-Command not Allowed";
- 0X0D0D: "Buffer Overflow (Data Length too large for reader buffer)";
- 0X0D0E: "User Interface Event";
- 0X0D11: "Communication type not supported, VT-1, burst, etc.";
- 0X0D12: "Secure interface is not functional or is in an intermediate state.";
- 0X0D13: "Data field is not mod 8";
- 0X0D14: "Pad - 0X80 not found where expected";
- 0X0D15: "Specified key type is invalid";
- 0X0D1: "Could not retrieve key from the SAM(InitSecureComm)";
- 0X0D17: "Hash code problem";
- 0X0D18: "Could not store the key into the SAM(InstallKey)";
- 0X0D19: "Frame is too large";
- 0X0D1A: "Unit powered up in authentication state but POS must resend the InitSecureComm command";
- 0X0D1B: "The EEPROM may not be initialized because SecCommInterface does not make sense";
- 0X0D1C: "Problem encoding APDU";
- 0X0D20: "Unsupported Index(ILM) SAM Transceiver error - problem communicating with the SAM(Key Mgr)";
- 0X0D2: "Unexpected Sequence Counter in multiple frames for single bitmap(ILM) Length error in data returned from the SAM(Key Mgr)";
- 0X0D22: "Improper bit map(ILM)";
- 0X0D23: "Request Online Authorization";
- 0X0D24: "ViVOCard3 raw data read successful";
- 0X0D25: "Message index not available(ILM) ViVOcomm activate transaction card type(ViVOcomm)";
- 0X0D26: "Version Information Mismatch(ILM)";
- 0X0D27: "Not sending commands in correct index message index(ILM)";
- 0X0D28: "Time out or next expected message not received(ILM)";

- 0X0D29: "ILM languages not available for viewing(ILM)";
- 0X0D2A: "Other language not supported(ILM)";
- 0X0D41: "Unknown Error from SAM";
- 0X0D42: "Invalid data detected by SAM";
- 0X0D43: "Incomplete data detected by SAM";
- 0X0D44: "Reserved";
- 0X0D45: "Invalid key hash algorithm";
- 0X0D46: "Invalid key encryption algorithm";
- 0X0D47: "Invalid modulus length";
- 0X0D48: "Invalid exponent";
- 0X0D49: "Key already exists";
- 0X0D4A: "No space for new RID";
- 0X0D4B: "Key not found";
- 0X0D4C: "Crypto not responding";
- 0X0D4D: "Crypto communication error";
- 0X0D4E: "Module-specific error for Key Manager";
- 0X0D4F: "All key slots are full (maximum number of keys has been installed)";
- 0X0D50: "Auto-Switch OK";
- 0X0D51: "Auto-Switch failed";
- 0X0D90: "Account DUKPT Key not exist";
- 0X0D91: "Account DUKPT Key KSN exausted";
- 0X0D00: "This Key had been loaded.";
- 0X0E00: "Base Time was loaded.";
- 0X0F00: "Encryption Or Decryption Failed.";
- 0X1000: "Battery Low Warning (It is High Priority Response while Battery is Low.)";
- 0X1800: "Send 'Cancel Command' after send 'Get Encrypted PIN' & 'Get Numeric' & 'Get Amount'; - 0X1900: "Press 'Cancel' key after send 'Get Encrypted PIN' & 'Get Numeric' & 'Get Amount';
- 0X30FF: "Security Chip is not connect";
- 0X3000: "Security Chip is deactivation & Device is In Removal Legally State.";
- 0X3101: "Security Chip is activation & Device is In Removal Legally State.";
- 0X5500: "No Admin DUKPT Key.";
- 0X5501: "Admin DUKPT Key STOP.";
- 0X5502: "Admin DUKPT Key KSN is Error.";
- 0X5503: "Get Authentication Code1 Failed.";
- 0X5504: "Validate Authentication Code Error.";
- 0X5505: "Encrypt or Decrypt data failed.";

- 0X5506: "Not Support the New Key Type.";
- 0X5507: "New Key Index is Error.";
- 0X5508: "Step Error.";
- 0X5509: "KSN Error";
- 0X550A: "MAC Error.";
- 0X550B: "Key Usage Error.";
- 0X550C: "Mode Of Use Error.";
- 0X550F: "Other Error.";
- 0X6000: "Save or Config Failed / Or Read Config Error.";
- 0X6200: "No Serial Number.";
- 0X6900: "Invalid Command - Protocol is right, but task ID is invalid.";
- 0X6A01: "Unsupported Command - Protocol and task ID are right, but command is invalid - In this State";
- 0X6A00: "Unsupported Command - Protocol and task ID are right, but command is invalid.";
- 0X6B00: "Unknown parameter in command - Protocol task ID and command are right, but parameter is invalid.";
- 0X6C00: "Unknown parameter in command - Protocol task ID and command are right, but length is out of the requirement.";
- 0X7200: "Device is suspend (MKSK suspend or press password suspend).";
- 0X7300: "PIN DUKPT is STOP (21 bit 1).";
- 0X7400: "Device is Busy.";
- 0XE100: "Can not enter sleep mode";
- 0XE200: "File has existed";
- 0XE300: "File has not existed";
- 0XE313: "IO line low -- Card error after session start";
- 0XE400: "Open File Error";
- 0XE500: "SmartCard Error";
- 0XE600: "Get MSR Card data is error";
- 0XE700: "Command time out";
- 0XE800: "File read or write is error";
- 0XE900: "Active 1850 error!";
- 0XEA00: "Load bootloader error";
- 0XEF00: "Protocol Error- STX or ETX or check error.";
- 0XEB00: "Picture is not exist";
- 0X2C02: "No Microprocessor ICC seated";
- 0X2C06: "no card seated to request ATR";
- 0X2D01: "Card Not Supported, ";
- 0X2D03: "Card Not Supported, wants CRC";

- 0X690D: "Command not supported on reader without ICC support";
- 0X8100: "ICC error time out on power-up";
- 0X8200: "invalid TS character received - Wrong operation step";
- 0X8300: "Decode MSR Error";
- 0X8400: "TriMagII no Response";
- 0X8500: "No Swipe MSR Card";
- 0X8510: "No Financial Card";
- 0X8600: "Unsupported F, D, or combination of F and D";
- 0X8700: "protocol not supported EMV TD1 out of range";
- 0X8800: "power not at proper level";
- 0X8900: "ATR length too long";
- 0X8B01: "EMV invalid TA1 byte value";
- 0X8B02: "EMV TB1 required";
- 0X8B03: "EMV Unsupported TB1 only 00 allowed";
- 0X8B04: "EMV Card Error, invalid BWI or CWI";
- 0X8B06: "EMV TB2 not allowed in ATR";
- 0X8B07: "EMV TC2 out of range";
- 0X8B08: "EMV TC2 out of range";
- 0X8B09: "per EMV96 TA3 must be $>$ - 0XF";
- 0X8B10: "ICC error on power-up";
- 0X8B11: "EMV T=1 then TB3 required";
- 0X8B12: "Card Error, invalid BWI or CWI";
- 0X8B13: "Card Error, invalid BWI or CWI";
- 0X8B17: "EMV TC1/TB3 conflict-";
- 0X8B20: "EMV TD2 out of range must be T=1";
- 0X8C00: "TCK error";
- 0XA304: "connector has no voltage setting";
- 0XA305: "ICC error on power-up invalid (SBLK(IFSD) exchange";
- 0XE301: "ICC error after session start";
- 0XFF00: "Request to go online";
- 0XFF01: "EMV: Accept the offline transaction";
- 0XFF02: "EMV: Decline the offline transaction";
- 0XFF03: "EMV: Accept the online transaction";
- 0XFF04: "EMV: Decline the online transaction";
- 0XFF05: "EMV: Application may fallback to magstripe technology";
- 0XFF06: "EMV: ICC detected tah the conditions of use are not satisfied";

- 0XFF07: "EMV: ICC didn't accept transaction";
- 0XFF08: "EMV: Transaction was cancelled";
- 0XFF09: "EMV: Application was not selected by kernel or ICC format error or ICC missing data error";
- 0XFF0A: "EMV: Transaction is terminated";
- 0XFF0B: "EMV: Other EMV Error";
- 0XFFFF: "NO RESPONSE";
- 0XF002: "ICC communication timeout";
- 0XF003: "ICC communication Error";
- 0XF00F: "ICC Card Seated and Highest Priority, disable MSR work request";
- 0XF200: "AID List / Application Data is not exist";
- 0XF201: "Terminal Data is not exist";
- 0XF202: "TLV format is error";
- 0XF203: "AID List is full";
- 0XF204: "Any CA Key is not exist";
- 0XF205: "CA Key RID is not exist";
- 0XF206: "CA Key Index it not exist";
- 0XF207: "CA Key is full";
- 0XF208: "CA Key Hash Value is Error";
- 0XF209: "Transaction format error";
- 0XF20A: "The command will not be processing";
- 0XF20B: "CRL is not exist";
- 0XF20C: "CRL number exceed max number";
- 0XF20D: "Amount, Other Amount, Trasaction Type are missing";
- 0XF20E: "The Identification of algorithm is mistake";
- 0XF20F: "No Financial Card";
- 0XF210: "In Encrypt Result state, TLV total Length is greater than Max Length";
- 0X1001: "INVALID ARG";
- 0X1002: "FILE_OPEN_FAILED";
- 0X1003: "FILE OPERATION_FAILED";
- 0X2001: "MEMORY_NOT_ENOUGH";
- 0X3002: "SMARTCARD_FAIL";
- 0X3003: "SMARTCARD_INIT_FAILED";
- 0X3004: "FALLBACK_SITUATION";
- 0X3005: "SMARTCARD_ABSENT";
- 0X3006: "SMARTCARD_TIMEOUT";
- 0X3012: "EMV_RESULT_CODE_MSR_CARD_ERROR_FALLBACK";

- 0X5001: "EMV_PARSING_TAGS_FAILED";
- 0X5002: "EMV_DUPLICATE_CARD_DATA_ELEMENT";
- 0X5003: "EMV_DATA_FORMAT_INCORRECT";
- 0X5004: "EMV_NO_TERM_APP";
- 0X5005: "EMV_NO_MATCHING_APP";
- 0X5006: "EMV_MISSING_MANDATORY_OBJECT";
- 0X5007: "EMV_APP_SELECTION_RETRY";
- 0X5008: "EMV_GET_AMOUNT_ERROR";
- 0X5009: "EMV_CARD_REJECTED";
- 0X5010: "EMV_AIP_NOT_RECEIVED";
- 0X5011: "EMV_AFL_NOT_RECEIVED";
- 0X5012: "EMV_AFL_LEN_OUT_OF_RANGE";
- 0X5013: "EMV_SFI_OUT_OF_RANGE";
- 0X5014: "EMV_AFL_INCORRECT";
- 0X5015: "EMV_EXP_DATE_INCORRECT";
- 0X5016: "EMV_EFF_DATE_INCORRECT";
- 0X5017: "EMV_ISS_COD_TBL_OUT_OF_RANGE";
- 0X5018: "EMV_CRYPTOGRAM_TYPE_INCORRECT";
- 0X5019: "EMV_PSE_NOT_SUPPORTED_BY_CARD";
- 0X5020: "EMV_USER_SELECTED_LANGUAGE";
- 0X5021: "EMV_SERVICE_NOT_ALLOWED";
- 0X5022: "EMV_NO_TAG_FOUND";
- 0X5023: "EMV_CARD_BLOCKED";
- 0X5024: "EMV_LEN_INCORRECT";
- 0X5025: "CARD_COM_ERROR";
- 0X5026: "EMV_TSC_NOT_INCREASED";
- 0X5027: "EMV_HASH_INCORRECT";
- 0X5028: "EMV_NO_ARC";
- 0X5029: "EMV_INVALID_ARC";
- 0X5030: "EMV_NO_ONLINE_COMM";
- 0X5031: "TRAN_TYPE_INCORRECT";
- 0X5032: "EMV_APP_NO_SUPPORT";
- 0X5033: "EMV_APP_NOT_SELECT";
- 0X5034: "EMV_LANG_NOT_SELECT";
- 0X5035: "EMV_NO_TERM_DATA";
- 0X5039: "EMV_PIN_ENTRY_TIMEOUT";

- 0X6001: "CVM_TYPE_UNKNOWN";
- 0X6002: "CVM_AIP_NOT_SUPPORTED";
- 0X6003: "CVM_TAG_8E_MISSING";
- 0X6004: "CVM_TAG_8E_FORMAT_ERROR";
- 0X6005: "CVM_CODE_IS_NOT_SUPPORTED";
- 0X6006: "CVM_COND_CODE_IS_NOT_SUPPORTED";
- 0X6007: "NO_MORE_CVM";
- 0X6008: "PIN_BYPASSED_BEFORE";
- 0X7001: "PK_BUFFER_SIZE_TOO_BIG";
- 0X7002: "PK_FILE_WRITE_ERROR";
- 0X7003: "PK_HASH_ERROR";
- 0X8001: "NO_CARD_HOLDER_CONFIRMATION";
- 0X8002: "GET_ONLINE_PIN";
- 0XD000: "Data not exist";
- 0XD001: "Data access error";
- 0XD100: "RID not exist";
- 0XD101: "RID existed";
- 0XD102: "Index not exist";
- 0XD200: "Maximum exceeded";
- 0XD201: "Hash error";
- 0XD205: "System Busy";
- 0X0E01: "Unable to go online";
- 0X0E02: "Technical Issue";
- 0X0E03: "Declined";
- 0X0E04: "Issuer Referral transaction";
- 0X0F01: "Decline the online transaction";
- 0X0F02: "Request to go online";
- 0X0F03: "Transaction is terminated";
- 0X0F05: "Application was not selected by kernel or ICC format error or ICC missing data error";
- 0X0F07: "ICC didn’t accept transaction";
- 0X0F0A: "Application may fallback to magstripe technology";
- 0X0F0C: "Transaction was cancelled";
- 0X0F0D: "Timeout";
- 0X0F0F: "Other EMV Error";
- 0X0F10: "Accept the offline transaction";
- 0X0F11: "Decline the offline transaction";

- 0X0F21: "ICC detected tah the conditions of use are not satisfied";
- 0X0F22: "No app were found on card matching terminal configuration";
- 0X0F23: "Terminal file does not exist";
- 0X0F24: "CAPK file does not exist";
- 0X0F25: "CRL Entry does not exist";
- 0X0FFE: "code when blocking is disabled";
- 0X0FFF: "code when command is not applicable on the selected device";
- 0XF005: "ICC Encrypted C-APDU Data Structure Length Error Or Format Error.";
- 0XBBE0: "CM100 Success";
- 0XBBE1: "CM100 Parameter Error";
- 0XBBE2: "CM100 Low Output Buffer";
- 0XBBE3: "CM100 Card Not Found";
- 0XBBE4: "CM100 Collision Card Exists";
- 0XBBE5: "CM100 Too Many Cards Exist";
- 0XBBE6: "CM100 Saved Data Does Not Exist";
- 0XBBE8: "CM100 No Data Available";
- 0XBBE9: "CM100 Invalid CID Returned";
- 0XBBEA: "CM100 Invalid Card Exists";
- 0XBBEC: "CM100 Command Unsupported";
- 0XBBED: "CM100 Error In Command Process";
- 0XBBEE: "CM100 Invalid Command";
- 0X9031: "Unknown command";
- 0X9032: "Wrong parameter (such as the length of the command is incorrect)";
- 0X9038: "Wait (the command couldnt be finished in BWT)";
- 0X9039: "Busy (a previously command has not been finished)";
- 0X903A: "Number of retries over limit";
- 0X9040: "Invalid Manufacturing system data";
- 0X9041: "Not authenticated";
- 0X9042: "Invalid Master DUKPT Key";
- 0X9043: "Invalid MAC Key";
- 0X9044: "Reserved for future use";
- 0X9045: "Reserved for future use";
- 0X9046: "Invalid DATA DUKPT Key";
- 0X9047: "Invalid PIN Pairing DUKPT Key";
- 0X9048: "Invalid DATA Pairing DUKPT Key";
- 0X9049: "No nonce generated";

- 0X9949: "No GUID available. Perform getVersion first.";
- 0X9950: "MAC Calculation unsuccessful. Check BDK value.";
- 0X904A: "Not ready";
- 0X904B: "Not MAC data";
- 0X9050: "Invalid Certificate";
- 0X9051: "Duplicate key detected";
- 0X9052: "AT checks failed";
- 0X9053: "TR34 checks failed";
- 0X9054: "TR31 checks failed";
- 0X9055: "MAC checks failed";
- 0X9056: "Firmware download failed";
- 0X9060: "Log is full";
- 0X9061: "Removal sensor unengaged";
- 0X9062: "Any hardware problems";
- 0X9070: "ICC communication timeout";
- 0X9071: "ICC data error (such check sum error)";
- 0X9072: "Smart Card not powered up";

12.2.4.71 int device_getRTCDateTime (IN BYTE * *dateTime,* IN_OUT int * *dateTimeLen*)

get RTC date and time of the device

Parameters

dateTime	<dateTime data=""> is: 6 byte data for YYMMDDHHMMSS in hex. For example 0x171003102547 stands for 2017 Oct 3rd 10:25:47
dateTimeLen	return 6 bytes if successful

Returns

success or error code. Values can be parsed with device_getResponseCodeString

See Also

ErrorCode

12.2.4.72 int device_getSDKWaitTime ( )

Get SDK Wait Time

Get the SDK wait time for transactions

Returns

SDK wait time in seconds

12.2.4.73 int device_getSpectrumProKSN (IN int *type*, OUT BYTE * *KSN*)

DEPRECATED : please use device_getSpectrumProKSN_Len(IN int type, OUT BYTE * KSN, IN_OUT int *KSN-Len)

Get DUKPT KSN

Returns the KSN for the provided key index

Parameters

type	Key type: • 0: Key Encryption Key (Master Key or KEK) • 2: Data Encryption Key (DEK) • 5: MAC Key (MAK) • 10: RKL Key Encryption Key (REK) • 20: HSM DUKPT Key
KSN	Key Serial Number; needs to have at least 10 bytes of memory

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.2.4.74 int device_getSpectrumProKSN_Len (IN int *type*, OUT BYTE * *KSN*, IN_OUT int * *KSNLen*)

Get DUKPT KSN

Returns the KSN for the provided key index

Parameters

type	Key type: • 0: Key Encryption Key (Master Key or KEK) • 2: Data Encryption Key (DEK) • 5: MAC Key (MAK) • 10: RKL Key Encryption Key (REK) • 20: HSM DUKPT Key

KSN	Key Serial Number
KSNLen	Length of KSN

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.2.4.75 int device_getThreadStackSize ( )

Get Thread Stack Size

Get the stack size setting for newly created threads

Returns

Thread Stack Size

12.2.4.76 int device_init ( )

Initial the device by USB

It will detect the device and trying connect.

The connect status can be checked by device_isConnected().

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString() or device_getResponseCodeString()

12.2.4.77 int device_isAttached (int *deviceType*)

Check if the device is attached to the USB port The function device_init() must be called before this function.

Parameters

deviceType,the	device type of the USB device

Returns

1 if the device is attached, or 0 if the device is not attached

12.2.4.78 int device_isConnected ( )

Check the device conntected status

Returns

DEVICE_DISCONNECT=0, or DEVICE_CONNECTED = 1

12.2.4.79 int device_lcdDisplayClear ( )

Use this function to clear the LCD display

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.2.4.80 int device_lcdDisplayLine1Message (IN BYTE * *message,* IN int *messageLen*)

Use this function to display text on the LCD display. On the Vendi reader the LCD is a 2-line character display.

Parameters

message	Valid messages for the first line of text are between 1 and 16 printable characters long. If the text message is greater than 16 bytes but not more than 32 bytes, byte 17 and onward are displayed as a second row of text. All messages are left justified on the LCD display.
messageLen	Length of the message: 1 to 32 bytes

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.2.4.81 int device_lcdDisplayLine2Message (IN BYTE * *message*, IN int *messageLen*)

Use this function to display the message on line 2 of the LCD display. On the Vendi reader the LCD is a 2-line character display.

Parameters

message	Valid messages are between 1 and 16 printable characters long. All messages are left justified on the LCD display.
messageLen	Length of the message: 1 to 16 bytes

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.2.4.82 int device_listDirectory (IN char * *directoryName*, IN int *directoryNameLen*, IN int *recursive*, IN int *onSD*, OUT char * *directory*, IN_OUT int * *directoryLen*)

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
directoryNameLen	Directory Name Length. If null, root directory is listed
recursive	Included sub-directories
onSD	TRUE = use flash storage The returned directory information The returned directory information length

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.2.4.83 int device_pingDevice ( )

Ping Device - NEO only

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

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.2.4.84 int device_pollCardReader (OUT BYTE * *status*)

DEPRECATED : please use device_pollCardReader_Len(OUT BYTE * status, IN_OUT int *statusLen)

Poll Card Reader

Provides information about the state of the Card Reader

Parameters

status	Six bytes indicating card reader information Byte 0: • Bit 0: Device Manufacturing CA data valid • Bit 1: Device Manufacturing Secure data valid • Bit 2: HOST_CR_MASTER_DUKPT Key valid • Bit 3: HOST_CR_MAC Keys valid (Authenticated) • Bit 4: RFU • Bit 5: RFU • Bit 6: DATA_DUKPT Key Valid • Bit 7: Key is initialized (MFK and RSA Key pairs)

Byte 1:

- Bit 0: Firmware Key Valid
- Bit 1: RFU
- Bit 2: CR_PINPAD_MASTER_DUKPT Key valid
- Bit 3: CR_PINPAD_MAC Keys valid (Authenticated)
- Bit 4: DATA Pairing DUKPT Key valid
- Bit 5: PIN Pairing DUKPT Key Valid
- Bit 6: RFU
- Bit 7: RFU

Byte 2:

- Bit 0: RFU
- Bit 1: Tamper Switch #1 Error
- Bit 2: Battery Backup Error
- Bit 3: Temperature Error
- Bit 4: Voltage Sensor Error
- Bit 5: Firmware Authentication Error
- Bit 6: Tamper Switch #2 Error
- Bit 7: Removal Tamper Error

Byte 3:

- Battery Voltage (example 0x32 = 3.2V, 0x24 = 2.4V)

Byte 4:

- Bit 0: Log is Full
- Bit 1: Mag Data Present

- Bit 2: Card Insert
- Bit 3: Removal Sensor connected
- Bit 4: Card Seated
- Bit 5: Latch Mechanism Active
- Bit 6: Removal Sensor Active
- Bit 7: Tamper Detector Active

Byte 5:

- Bit 0: SAM Available
- Bit 1: Chip Card Reader Available
- Bit 2: Host Connected
- Bit 3: Contactless Available
- Bit 4: PINPAD connected
- Bit 5: MSR Header connected
- Bit 6: RFU
- Bit 7: Production Unit

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.2.4.85 int device_pollCardReader_Len (OUT BYTE * *status*, IN_OUT int * *statusLen*)

Poll Card Reader

Provides information about the state of the Card Reader

Parameters

status	Six bytes indicating card reader information Byte 0: • Bit 0: Device Manufacturing CA data valid • Bit 1: Device Manufacturing Secure data valid • Bit 2: HOST_CR_MASTER_DUKPT Key valid • Bit 3: HOST_CR_MAC Keys valid (Authenticated) • Bit 4: RFU • Bit 5: RFU • Bit 6: DATA_DUKPT Key Valid • Bit 7: Key is initialized (MFK and RSA Key pairs)

Byte 1:

- Bit 0: Firmware Key Valid
- Bit 1: RFU

- Bit 2: CR_PINPAD_MASTER_DUKPT Key valid
- Bit 3: CR_PINPAD_MAC Keys valid (Authenticated)
- Bit 4: DATA Pairing DUKPT Key valid
- Bit 5: PIN Pairing DUKPT Key Valid
- Bit 6: RFU
- Bit 7: RFU

Byte 2:

- Bit 0: RFU
- Bit 1: Tamper Switch #1 Error
- Bit 2: Battery Backup Error
- Bit 3: Temperature Error
- Bit 4: Voltage Sensor Error
- Bit 5: Firmware Authentication Error
- Bit 6: Tamper Switch #2 Error
- Bit 7: Removal Tamper Error

Byte 3:

- Battery Voltage (example 0x32 = 3.2V, 0x24 = 2.4V)

Byte 4:

- Bit 0: Log is Full
- Bit 1: Mag Data Present
- Bit 2: Card Insert
- Bit 3: Removal Sensor connected
- Bit 4: Card Seated
- Bit 5: Latch Mechanism Active
- Bit 6: Removal Sensor Active
- Bit 7: Tamper Detector Active

Byte 5:

- Bit 0: SAM Available
- Bit 1: Chip Card Reader Available
- Bit 2: Host Connected
- Bit 3: Contactless Available
- Bit 4: PINPAD connected
- Bit 5: MSR Header connected
- Bit 6: RFU
- Bit 7: Production Unit

Parameters

statusLen	Length of status

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.2.4.86 int device_pollForToken (IN int *timeout*, OUT BYTE * *respData*, IN_OUT int * *respDataLen*)

Poll for Token

Polls for a PICC

Parameters

timeout	timeout in milliseconds, must be multiple of 10 milliseconds. 30, 120, 630, or 1150 for example.
respData	Response data will be stored in respData. 1 byte of card type, and the Serial Number (or the UID) of the PICC if available.
respDataLen	Length of systemCode.

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.2.4.87 int device_queryFile (IN char * *directoryName*, IN int *directoryNameLen*, IN char * *fileName*, IN int *fileNameLen*, OUT int * *isExist*, OUT BYTE * *timeStamp*, IN_OUT int * *timeStampLen*, OUT char * *fileSize*, IN_OUT int * *fileSizeLen*)

Query File This command checks if the specified file exists in NAND Flash..

Parameters

directoryName	Directory name string. No longer than 32 bytes. ASCII string, terminated by 0x00.
directoryNameLen	Directory Name Length.
fileName	File name string. No longer than 32 bytes. ASCII string, terminated by 0x00.
fileNameLen	File Name Length.
isExist	File exists: 1, File not exists 0.
timeStamp	Latest time stamp of the file. 6 bytes BCD code if the file exists.
timeStampLen	Length of timeStamp. 6 if the file exists, 0 if the file does not exist.
fileSize	Zero-terminated ASCII string of the file size.
fileSizeLen	Length of filesSize.

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.2.4.88 int device_rebootDevice ( )

Reboot Device - NGA 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()

12.2.4.89 void device_registerCameraCallBk (pCMR_callBack *pCMRf*)

To register the camera callback function to get the image data. (Pass NULL to disable the callback.)

12.2.4.90 void device_registerCardStatusFrontSwitchCallBk (pCSFS_callBack *pCSFSf*)

To register the card status and front switch callback function to get status. (Pass NULL to disable the callback.)

12.2.4.91 void device_registerFWCallBk (pFW_callBack *pFWf*)

To register the firmware update callback function to get the firmware update status. (Pass NULL to disable the callback.)

12.2.4.92 void device_registerRKICallBk (pRKI_callBack *pRKIf*)

To register the RKI callback function to get the RKI status. (Pass NULL to disable the callback.)

12.2.4.93 int device_selfCheck ( )

Self check for TTK If Self-Test function Failed, then work into De-activation State. If device work into De-activation State, All Sensitive Data will be erased and it need be fixed in Manufacture.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.2.4.94 int device_SendDataCommand (IN BYTE * *cmd*, IN int *cmdLen*, IN BYTE * *data*, IN int *dataLen*, OUT BYTE * *response*, IN_OUT int * *respLen*)

Send a Command to NGA device

Sends a command to the device .

Parameters

cmd	buffer of command to execute.
cmdLen,the	length of the buffer cmd.
data	buffer of IDG command data.
dataLen,the	length of the buffer data.
response	Response data
respLen,the	length of Response data

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.2.4.95 int device_SendDataCommandITP (IN BYTE * *cmd*, IN int *cmdLen*, IN BYTE * *data*, IN int *dataLen*, OUT BYTE * *response*, IN_OUT int * *respLen*)

Send a Command to ITP device

Sends a command to the device .

Parameters

cmd	buffer of command to execute.
cmdLen,the	length of the buffer cmd.
data	buffer of ITP command data.
dataLen,the	length of the buffer data.
response	Response data
respLen,the	length of Response data

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.2.4.96 int device_SendDataCommandNEO (IN int *cmd*, IN int *subCmd*, IN BYTE * *data*, IN int *dataLen*, OUT BYTE * *response*, IN_OUT int * *respLen*)

Send a Command to NEO device

Sends a command to the NEO device .

Parameters

cmd	command to execute.
subCmd,sub	command to execute.
data	buffer of NEO command data.
dataLen,the	length of the buffer data.
response	Response data
respLen,the	length of Response data

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.2.4.97 int device_setBurstMode (IN BYTE *mode*)

Send Burst Mode - NEO

Sets the burst mode for the device.

Parameters

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

Returns

success or error code. Values can be parsed with device_getIDGStatusCodeString

See Also

ErrorCode

12.2.4.98 int device_setCancelTransactionMode (int *mode*)

Set Cancel Transaction Mode

Set the cancel transaction mode to be with or without LCD message

Parameters

mode	0: With LCD message 1: Without LCD message

Returns

success or error code. 1: Success, 0: Failed

12.2.4.99 int device_setConfigPath (const char * *path*)

Set the path to the config xml file(s) if any

Parameters

path	The path to the config xml files (such as "NEO2_Devices.xml" which contains the information of NEO2 devices). Only need to specify the path to the folder which contains the config files. File names are not needed. The maximum length of path is 200 characters including the '\0' at the end.

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.2.4.100 int device_setCurrentDevice (int *deviceType*)

Sets the current device to talk to

The connect status can be checked by device_isConnected().

Parameters

deviceType	Device to connect to
	
	`enum DEVICE_TYPE {`
	`  IDT_DEVICE_UNKNOWN=0,`
	`  IDT_DEVICE_AUGUSTA_HID,`
	`  IDT_DEVICE_AUGUSTA_KB,`
	`  IDT_DEVICE_AUGUSTA_S_HID,`
	`  IDT_DEVICE_AUGUSTA_S_KB,`
	`  IDT_DEVICE_AUGUSTA_S_TTK_HID,`
	`  IDT_DEVICE_SPECTRUM_PRO,`
	`  IDT_DEVICE_MINISMART_II,`
	`  IDT_DEVICE_L100,`
	`  IDT_DEVICE_UNIPAY,`
	`  IDT_DEVICE_UNIPAY_I_V,`
	`  IDT_DEVICE_VP3300_AJ,`
	`  IDT_DEVICE_KIOSK_III,`
	`  IDT_DEVICE_KIOSK_III_S,`
	`  IDT_DEVICE_PIP_READER,`
	`  IDT_DEVICE_VENDI,`
	`  IDT_DEVICE_VP3300_USB,`
	`  IDT_DEVICE_UNIPAY_I_V_TTK,`
	`  IDT_DEVICE_VP3300_BT,`
	`  IDT_DEVICE_VP8800,`
	`  IDT_DEVICE_SREDKEY2_HID,`
	`  IDT_DEVICE_SREDKEY2_KB,`
	`  IDT_DEVICE_NEO2,`
	`  IDT_DEVICE_MINISMART_II_COM = IDT_DEVICE_NEO2+5,`
	`  IDT_DEVICE_SPECTRUM_PRO_COM,`
	`  IDT_DEVICE_KIOSK_III_COM,`
	`  IDT_DEVICE_KIOSK_III_S_COM,`
	`  IDT_DEVICE_VP3300_COM,`
	`  IDT_DEVICE_NEO2_COM,`
	`  IDT_DEVICE_MAX_DEVICES = IDT_DEVICE_NEO2_COM+5`
	`};`

Returns

RETURN_CODE: 1: success, 0: failed

12.2.4.101 int device_setMerchantRecord (int *index*, int *enabled*, char * *merchantID*, char * *merchantURL*)

Set Merchant Record - NEO 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 device_getIDGStatusCodeString()

12.2.4.102 int device_setNEO2DevicesConfigs (IN const char * *configs*, IN int *len*)

Pass the content of the config xml file ("NEO2_Devices.xml") as a string to the SDK instead of reading the config xml file by the SDK It needs to be called before device_init(), otherwise the SDK will try to read the config xml file.

Parameters

configs	The content read from the config xml file ("NEO2_Devices.xml" which contains the information of NEO2 devices).
len	The length of the string configs. The maximum length is 5000 bytes.

12.2.4.103 int device_setPollMode (IN BYTE *mode*)

Set Poll Mode - NEO

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 device_getIDGStatusCodeString()

12.2.4.104 int device_setRTCDateTime (IN BYTE * *dateTime*, IN int *dateTimeLen*)

set RTC date and time of the device

Parameters

dateTime	<dateTime data=""> is: 6 byte data for YYMMDDHHMMSS in hex. For example 0x171003102547 stands for 2017 Oct 3rd 10:25:47
dateTimeLen	should be always 6 bytes

Returns

success or error code. Values can be parsed with device_getResponseCodeString

See Also

ErrorCode

12.2.4.105 void device_setSDKWaitTime (int *waitTime*)

Set SDK Wait Time

Set the SDK wait time for transactions

Parameters

waitTime	The SDK wait time for transaction in seconds

12.2.4.106 int 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

12.2.4.107 int device_setSystemLanguage (char * *language*)

Set Model Number for the device

Parameters

sNumber	Model Number

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString Set System Language Sets the language for the message displayed in the LCD screen

Parameters

language	2-byte ASCII code, can be "EN" or "JP"

Returns

success or error code. Values can be parsed with device_getIDGStatusCodeString

See Also

ErrorCode

12.2.4.108 void device_setThreadStackSize (int *threadSize*)

Set Thread Stack Size

Set the stack size setting for newly created threads

12.2.4.109 void device_setTransactionExponent (int *exponent*)

Sets the transaction exponent to be used with device_startTransaction. Default value is 2

Parameters

exponent,The	exponent to use when calling device_startTransaction

12.2.4.110 int device_startListenNotifications ( )

Start Listen Notifications This function enables Card Status and Front Switch notifications.

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.2.4.111 int device_startQRCodeScan (IN int *_timeout*)

Start QR Code Scanning

Enables QR Code scanning, waiting for the QR code.

Parameters

timeout	QR Code Scan Timeout Value. Between 30 and 65536 seconds.

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString() Note: if auto poll mode is on, it will return command not allowed error

12.2.4.112 int device_startRKI (const char * *caPath*)

Start remote key injection.

Parameters

caPath	The path to ca-certificates.crt

Returns

success or error code.

See Also

ErrorCode

12.2.4.113 int device_startTakingPhoto (IN int *_timeout*)

Start Taking Photo

Enables the camera to take a photo.

Parameters

timeout	Photo taking Timeout Value. Between 30 and 65536 seconds.

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString() Note: if auto poll mode is on, it will return command not allowed error

12.2.4.114 int device_startTransaction (IN double *amount,* IN double *amtOther,* IN int *type,* IN const int *_timeout,* IN BYTE * *tags,* IN int *tagsLen*)

Start Transaction Request

Authorizes the transaction for an MSR/CTLS/ICC card

The tags will be returned in the callback routine.

Parameters

amount	Transaction amount value (tag value 9F02) • SEE IMPORTANT NOTE BELOW
amtOther	Other amount value, if any (tag value 9F03) • SEE IMPORTANT NOTE BELOW
type	Transaction type (tag value 9C).
timeout	Timeout value in seconds.
tags	Any other tags to be included in the request. Passed as TLV. 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.
tagsLen	The length of tags data buffer.

>>>>>IMPORTANT: parameters for amount and amtOther MUST BE PASSED AS A DOUBLE VALUE WITH DECIMAL POINT. Example, do not pass 1, but instead pass 1.0 or 1.00. Otherwise, results will be unpredictable

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString() Note: if auto poll is on, it will returm the error IDG_P2_STATUS_CODE_COMMAND_NOT_ALLOWED

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 device_startTransaction. Tag FFEE06 must contain tag 9F26 and 9F22, and can optionanally contain tags 9F2B and DF01. 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

12.2.4.115 int device_stopListenNotifications ( )

Stop Listen Notifications This function disables Card Status and Front Switch notifications.

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.2.4.116 int device_stopQRCodeScan ( )

Stop QR Code Scanning Cancels QR Code scanning request.

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.2.4.117 int device_stopTakingPhoto ( )

Stop Taking Photo Cancels Photo Taking request.

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.2.4.118 void device_toSDCard (int *forSDCard*)

To SD Card

Set the destination of the file or directory function

Parameters

forSDCard	0: for internal memory, 1: for SD card

12.2.4.119 int device_transferFile (IN char * *fileName,* IN int *fileNameLen,* IN BYTE * *file,* IN int *fileLen*)

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 (/)
filenameLen	File Name Length.
file	The data file.
fileLen	File Length.

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.2.4.120 int device_turnOffYellowLED ( )

Use this function to 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_getIDGStatusCodeString()

12.2.4.121 int device_turnOnYellowLED ( )

Use this function to 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_getIDGStatusCodeString()

12.2.4.122 int device_updateFirmware (IN BYTE * *firmwareData,* IN int *firmwareDataLen,* IN char * *firmwareName,* IN int *encryptionType,* IN BYTE * *keyBlob,* IN int *keyBlobLen*)

Update Firmware - NGA Updates the firmware of the Spectrum Pro K21 HUB or Maxq1050.

Parameters

firmwareData	Signed binary data of a firmware file provided by IDTech
firmwareDataLen	Length of firmwareData
firmwareName	Firmware name. Must be one of the following two strings (with appropriate version information) • "SP K21 APP Vx.xx.xxx" • "SP MAX APP Vx.xx.xxx"
encryptionType	Encryption type • 0 : Plaintext • 1 : TDES ECB, PKCS#5 padding • 2 : TDES CBC, PKCS#5, IV is all 0
keyBlob	Encrypted firmware session key blob, TR-31 Rev B, wrapped by FW Key (Optional, none if firmware is plaintext)
keyBlobLen	Length of keyBlob

Returns

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

Firmware update status is returned in the callback with the following values: sender = SPECTRUM_PRO state = DeviceState.FirmwareUpdate data = File Progress. Two bytes, with byte[0] = current block, and byte[1] = total blocks. 0x0310 = block 3 of 16 transactionResultCode:

- RETURN_CODE_DO_SUCCESS = Firmware Update Completed Successfully
- RETURN_CODE_BLOCK_TRANSFER_SUCCESS = Current block transferred successfully
- Any other return code represents an error condition

12.2.4.123 int device_verifyBackdoorKey ( )

Verify Backdoor Key to Unlock Security

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

Note: The function is only for TTK devices.

12.2.4.124 int emv_activateTransaction (IN int *timeout,* IN BYTE * *tags,* IN int *tagsLen,* IN int *forceOnline*)

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	Tags to be included in the request. Passed as a TLV stream. Example, tag 9F02 with amount 0x000000000100 would be 0x9F0206000000000100
tagsLen	Length of tags
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 = DFEE1A079F029F36959F37

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString >>>>>IMPORTANT: parameters for amount and amtOther MUST BE PASSED AS A DOUBLE VALUE WITH DECIMAL POINT. Example, do not pass 1, but instead pass 1.0 or 1.00. Otherwise, results will be unpredictable.

12.2.4.125 void emv_allowFallback (IN int *allow*)

Allow fallback for EMV transactions. Default is TRUE

Parameters

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

12.2.4.126 int emv_authenticateTransaction (IN BYTE * *updatedTLV,* IN int *updatedTLVLen*)

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.

Parameters

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, 9F37 to be included in response = DFEE1A079F029F36959F37

updatedTLVLen	

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.2.4.127 int emv_authenticateTransactionWithTimeout (IN int *timeout,* IN BYTE * *updatedTLV,* IN int *updatedTLVLen*)

Authenticate EMV Transaction Request with Timeout

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.

Parameters

timeout	Timeout value in seconds.
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, 9F37 to be included in response = DFEE1-A079F029F36959F37
updatedTLVLen	

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.2.4.128 int emv_callbackResponseLCD (IN int *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_DISPLAY_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 device_getResponseCodeString()

12.2.4.129 int emv_callbackResponseMSR (IN BYTE * *MSR*, IN_OUT int *MSRLen*)

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
MSRLen	the length of Swiped track data

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.2.4.130 int emv_cancelTransaction ( )

Cancel EMV Transaction

Cancels the currently executing EMV transaction.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.2.4.131 int emv_completeTransaction (IN int *commError*, IN BYTE * *authCode*, IN int *authCodeLen*, IN BYTE * *iad*, IN int *iadLen*, IN BYTE * *tlvScripts*, IN int *tlvScriptsLen*, IN BYTE * *tlv*, IN int *tlvLen*)

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(1) if host was unreachable, or FALSE(0) 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
authCodeLen	the length of authCode
iad	Issuer Authentication Data, if any. Example 0x11223344556677883030 (tag value 91).
iadLen	the length of iadLen
tlvScripts	71/72 scripts, if any
tlvScriptsLen	the length of tlvScriptsLen
tlv	Additional TLV data to return with transaction results (if any)
tlvLen	the length of tlv

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.2.4.132 int emv_getAutoAuthenticateTransaction ( )

Gets auto authenticate value for EMV transactions.

Returns

RETURN_CODE: TRUE = auto authenticate, FALSE = manually authenticate

12.2.4.133 int emv_getAutoCompleteTransaction ( )

Gets auto complete value for EMV transactions.

Returns

RETURN_CODE: TRUE = auto complete, FALSE = manually complete

12.2.4.134 int emv_getEMVConfigurationCheckValue (OUT BYTE * *checkValue*, IN_OUT int * *checkValueLen*)

Get EMV Kernel configuration check value info

Parameters

checkValue	Response returned of Kernel configuration check value info
checkValueLen	the length of checkValue

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.2.4.135 int emv_getEMVKernelCheckValue (OUT BYTE * *checkValue*, IN_OUT int * *checkValueLen*)

Get EMV Kernel check value info

Parameters

checkValue	Response returned of Kernel check value info
checkValueLen	the length of checkValue

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.2.4.136 int emv_getEMVKernelVersion (OUT char * *version*)

DEPRECATED : please use emv_getEMVKernelVersion_Len(OUT char* version, IN_OUT int *versionLen)

Polls device for EMV Kernel Version

Parameters

version	Response returned of Kernel Version; needs to have at least 128 bytes of memory.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.2.4.137 int emv_getEMVKernelVersion_Len (OUT char * *version*, IN_OUT int * *versionLen*)

Polls device for EMV Kernel Version

Parameters

version	Response returned of Kernel Version
versionLen	Length of version

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.2.4.138 void emv_registerCallBk (pEMV_callBack *pEMVf*)

To register the emv callback function to get the EMV processing response. (Pass NULL to disable the callback.)

12.2.4.139 int emv_removeAllApplicationData ( )

Remove All Application Data

Removes all the Application Data

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.2.4.140 int emv_removeAllCAPK ( )

Remove All Certificate Authority Public Key

Removes all the CAPK

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.2.4.141 int emv_removeAllCRL ( )

Remove All Certificate Revocation List Entries

Removes all CRLEntry entries

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.2.4.142 int emv_removeApplicationData (IN BYTE * *AID*, IN int *AIDLen*)

Remove Application Data by AID

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

Parameters

AID	Name of ApplicationID Must be between 5 and 16 bytes
AIDLen	the length of AID data buffer

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.2.4.143 int emv_removeCAPK (IN BYTE * *capk*, IN int *capkLen*)

Remove Certificate Authority Public Key

Removes the CAPK as specified by the RID/Index

Parameters

capk	6 byte CAPK = 5 bytes RID + 1 byte INDEX
capkLen	the length of capk data buffer

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.2.4.144 int emv_removeCRL (IN BYTE * *list*, IN int *lsLen*)

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
lsLen	the length of list data buffer

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.2.4.145 int emv_removeTerminalData ( )

Remove Terminal Data

Removes the Terminal Data

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.2.4.146 int emv_retrieveAIDList (OUT BYTE * *AIDList*, IN_OUT int * *AIDListLen*)

Retrieve AID list

Returns all the AID names installed on the terminal.

Parameters

AIDList	array of AID name byte arrays
AIDListLen	the length of AIDList array buffer

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.2.4.147 int emv_retrieveApplicationData (IN BYTE * *AID*, IN int *AIDLen*, OUT BYTE * *tlv*, IN_OUT int * *tlvLen*)

Retrieve Application Data by AID

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

Parameters

AID	Name of ApplicationID. Must be between 5 and 16 bytes
AIDLen	the length of AID data buffer.
tlv	The TLV elements of the requested AID
tlvLen	the length of tlv data buffer.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.2.4.148 int emv_retrieveCAPK (IN BYTE * *capk*, IN int *capkLen*, OUT BYTE * *key*, IN_OUT int * *keyLen*)

Retrieve Certificate Authority Public Key

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

Parameters

capk	6 bytes CAPK = 5 bytes RID + 1 byte Index
capkLen	the length of capk data buffer
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.
keyLen	the length of key data buffer

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.2.4.149 int emv_retrieveCAPKList (OUT BYTE * *keys*, IN_OUT int * *keysLen*)

Retrieve the Certificate Authority Public Key list

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

Parameters

keys	[key1][key2]...[keyn], each key 6 bytes where key = 5 bytes RID + 1 byte index
keysLen	the length of keys data buffer

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.2.4.150 int emv_retrieveCRL (OUT BYTE * *list,* IN_OUT int * *lssLen*)

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
lssLen	the length of list data buffer

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.2.4.151 int emv_retrieveTerminalData (OUT BYTE * *tlv,* IN_OUT int * *tlvLen*)

Retrieve Terminal Data

Retrieves the Terminal Data.

Parameters

tlv	Response returned as a TLV
tlvLen	the length of tlv data buffer

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.2.4.152 int emv_retrieveTerminalID (OUT char * *terminalID*)

DEPRECATED : please use emv_retrieveTerminalID_Len(OUT char* terminalID, IN_OUT int *terminalIDLen)

Gets the terminal ID as printable characters .

Parameters

terminalID	Terminal ID string; needs to have at least 30 bytes of memory

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.2.4.153 int emv_retrieveTerminalID_Len (OUT char * *terminalID,* IN_OUT int * *terminalIDLen*)

Gets the terminal ID as printable characters .

Parameters

terminalID	Terminal ID string
terminalIDLen	Length of terminalID

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.2.4.154 int emv_retrieveTransactionResult (IN BYTE * *tags*, IN int *tagsLen*, IDTTransactionData * *cardData*)

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
tagsLen	Length of tag list
cardData	All requested tags returned as unencrypted, encrypted and masked TLV data in IDTTransactionData object

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.2.4.155 int emv_setApplicationData (IN BYTE * *name*, IN int *nameLen*, IN BYTE * *tlv*, IN int *tlvLen*)

Set Application Data by AID

Sets the Application Data as specified by the application name and TLV data

Parameters

name	Application name, 10-32 ASCII hex characters representing 5-16 bytes Example "a0000000031010"
nameLen	the length of name data buffer of Application name,
tlv	Application data in TLV format
tlvLen	the length of tlv data buffer

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.2.4.156 int emv_setApplicationDataTLV (IN BYTE * *tlv*, IN int *tlvLen*)

Set Application Data by TLV

Sets the Application Data as specified by the 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) Example valid TLV, for Group #2, AID a0000000035010- : "dfee2d01029f0607a0000000051010ffe10101ffe50110ffe30114ffe20106"
tlvLen	the length of tlv data buffer

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.2.4.157 void emv_setAutoAuthenticateTransaction (IN int *authenticate*)

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

Parameters

authenticate	TRUE = auto authenticate, FALSE = manually authenticate

12.2.4.158 void emv_setAutoCompleteTransaction (IN int *complete*)

Enables complete for EMV transactions. If a emv_authenticateTranaction results in code 0x0004 (go online), then emv_completeTransaction can automatically execute if parameter is set to TRUE

Parameters

complete	TRUE = auto complete, FALSE = manually complete

12.2.4.159 int emv_setCAPK (IN BYTE * *capk*, IN int *capkLen*)

Set Certificate Authority Public Key

Sets the CAPK as specified by the CAKey structure

Parameters

capk	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.
capkLen	the length of capk data buffer

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.2.4.160 int emv_setCRL (IN BYTE * *list*, IN int *lsLen*)

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]
lsLen	the length of list data buffer

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.2.4.161 int emv_setTerminalData (IN BYTE * *tlv*, IN int *tlvLen*)

Set Terminal Data

Sets the Terminal Data as specified by the TerminalData structure passed as a parameter

Parameters

tlv	TerminalData configuration file
tlvLen	the length of tlv data buffer

Return values

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

12.2.4.162 int emv_setTerminalID (IN char * *terminalID*)

Sets the terminal ID as printable characters .

Parameters

terminalID	Terminal ID to set

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.2.4.163 int emv_setTerminalMajorConfiguration (IN int *configuration*)

Sets the terminal major configuration in ICS .

Parameters

configuration	A configuration value, range 1-23 • 1 = 1C • 2 = 2C • 3 = 3C • 4 = 4C • 5 = 5C ... • 23 = 23C

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.2.4.164 void emv_setTransactionParameters (IN double *amount,* IN double *amtOther,* IN int *type,* IN int *timeout,* IN BYTE * *tags,* IN int *tagsLen*)

Set EMV Transaction Parameters

Set the parameters to be used on EMV transactions for an ICC card when Auto Poll is on

The tags will be returned in the callback routine.

Parameters

amount	Transaction amount value (tag value 9F02)
amtOther	Other amount value, if any (tag value 9F03)
type	Transaction type (tag value 9C).
timeout	Timeout value in seconds.
tags	Any other tags to be included in the request (Maximum Length = 500 bytes). Passed as a string. Example, tag 9F0C with amount 0x000000000100 would be "9F0C06000000000100" 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. 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 = DFEE1A079F029F36959F37

tagsLen	the length of tags

12.2.4.165 int emv_startTransaction (IN double *amount*, IN double *amtOther*, IN int *exponent*, IN int *type*, IN int *timeout*, IN BYTE * *tags*, IN int *tagsLen*, IN int *forceOnline*)

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) - SEE IMPORTANT NOTE BELOW
amtOther	Other amount value, if any (tag value 9F03) - SEE IMPORTANT NOTE BELOW
exponent	Number of characters after decimal point
type	Transaction type (tag value 9C).
timeout	Timeout value in seconds.
tags	Any other tags to be included in the request. Passed as a TLV stream. Example, tag 9F0-C with amount 0x000000000100 would be 0x9F0C06000000000100 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.
tagsLen	Length of tags
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 = DFEE1A079F029-F36959F37

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString >>>>>IMPORTANT: parameters for amount and amtOther MUST BE PASSED AS A DOUBLE VALUE WITH DECIMAL POINT. Example, do not pass 1, but instead pass 1.0 or 1.00. Otherwise, results will be unpredictable

12.2.4.166 int felica_authentication (IN BYTE * *key*, IN int *keyLen*)

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

Parameters

key	16-byte key used for MAC generation of Read or Write with MAC
keyLen	length of key, must be 16 bytes

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.2.4.167 int felica_poll (IN BYTE * *systemCode*, IN int *systemCodeLen*, OUT BYTE * *respData*, OUT int * *respDataLen*)

FeliCa Poll for Card

Polls for a Felica Card

Parameters

systemCode	System Code.
systemCodeLen	Length of systemCode. Must be 2 bytes
respData	response data will be stored in respData. Poll response as explained in FeliCA Lite-S User's Manual
respDataLen	Length of systemCode.

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.2.4.168 int felica_read (IN BYTE * *serviceCodeList*, IN int *serviceCodeListLen*, IN int *blockCnt*, IN BYTE * *blockList*, IN int *blockListLen*, OUT BYTE * *blockData*, OUT int * *blockDataLen*)

FeliCa Read

Reads up to 4 blocks.

Parameters

serviceCodeList	Service Code List. Each service code in Service Code List = 2 bytes of data
serviceCodeList-Len	Length of serviceCodeList
blockCnt	Number of blocks in blockList. Maximum 4 block requests
blockList	Block to read. Each block in blockList = 2 or 3 bytes of data.
blockListLen	Length of blockList.
blockData	Blocks read will be stored in blockData. Each block 16 bytes.
blockDataLen	Length of blockData.

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.2.4.169 int felica_readWithMac (IN int *blockCnt*, IN BYTE * *blockList*, IN int *blockListLen*, OUT BYTE * *blockData*, OUT int * *blockDataLen*)

FeliCa Read with MAC Generation

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

Parameters

blockCnt	Number of blocks in blockList. Maximum 3 block requests
blockList	Block to read. Each block in blockList = 2 or 3 bytes of data.
blockListLen	Length of blockList.
blockData	Blocks read will be stored in blockData. Each block is 16 bytes.
blockDataLen	Length of blockData.

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.2.4.170 int felica_requestService (IN BYTE * *nodeCode*, IN int *nodeCodeLen*, OUT BYTE * *respData*, OUT int * *respDataLen*)

FeliCa Request Service

Request Service for a Felica Card

Parameters

nodeCode	Node Code List. Each node 2 bytes
nodeCodeLen	Length of nodeCode.
respData	response data will be stored in respData. Response as explained in FeliCA Lite-S User's Manual
respDataLen	Length of respData.

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.2.4.171 int felica_SendCommand (IN BYTE * *command*, IN int *commandLen*, OUT BYTE * *respData*, OUT int * *respDataLen*)

FeliCa Send Command

Send a Felica Command

Parameters

command	Command data from settlement center to be sent to felica card
commandLen	Length of command data
respData	Response data from felica card.
respDataLen	Length of respData.

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.2.4.172 int felica_write (IN BYTE * *serviceCodeList*, IN int *serviceCodeListLen*, IN int *blockCnt*, IN BYTE * *blockList*, IN int *blockListLen*, IN BYTE * *blockData*, IN int *blockDataLen*, OUT BYTE * *statusFlag*, OUT int * *statusFlagLen*)

FeliCa Write

Writes a block

Parameters

serviceCodeList	Service Code List. Each service code in Service Code List = 2 bytes of data
serviceCodeList-Len	Length of serviceCodeList
blockCnt	Number of blocks in blockList. Currently only support 1 block.
blockList	Block list. Each block in blockList = 2 or 3 bytes of data.
blockListLen	Length of blockData.
blockData	Block to write.
blockDataLen	Length of blockData. Must be 16 bytes.
respData	If successful, the Status Flag (2 bytes) is stored in respData.resData. Status flag response as explained in FeliCA Lite-S User's Manual, Section 4.5

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.2.4.173 int felica_writeWithMac (IN BYTE *blockNum*, IN BYTE * *blockData*, IN int *blockDataLen*)

FeliCa Write with MAC Generation

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

Parameters

blockNum	Number of block
blockData	Block to write.
blockDataLen	Length of blockData. Must be 16 bytes.

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.2.4.174 int icc_disable ( )

ICC Function enable/disable - AUGUSTA Disable ICC function

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.2.4.175 int icc_enable (IN int *withNotification*)

ICC Function enable/disable - AUGUSTA Enable ICC function with or without seated notification

Parameters

withNotification	• 1: with notification when ICC seated status changed, • 0: without notification.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.2.4.176 int icc_exchangeAPDU (IN BYTE * *c_APDU,* IN int *cLen,* OUT BYTE * *reData,* IN_OUT int * *reLen*)

Exchange APDU with plain text - AUGUSTA For Non-SRED Augusta Only

Sends an APDU packet to the ICC. If successful, response is the APDU data in response parameter.

Parameters

c_APDU	APDU data packet
cLen	APDU data packet length
reData	Unencrypted APDU response
reLen	Unencrypted APDU response data length

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.2.4.177 int icc_exchangeEncryptedAPDU (IN BYTE * *c_APDU,* IN int *cLen,* OUT BYTE * *reData,* IN_OUT int * *reLen*)

Exchange APDU with encrypted data - AUGUSTA For SRED Augusta Only

Sends an APDU packet to the ICC. If successful, response is the APDU data in response parameter.

Parameters

c_APDU	KSN + encytpted APDU data packet, or no KSN (use last known KSN) + encrypted APDU data packet With KSN: [0A][KSN][Encrypted C-APDU] Without KSN: [00][Encrypted C-APDU]

The format of Raw C-APDU Data Structure of [m-bytes Encrypted C-APDU] is below:

- m = 2 bytes Valid C-APDU Length + x bytes Valid C-APDU + y bytes Padding (0x00) Note: For TDES mode: 2+x should be multiple of 8. If it was not multiple of 8, unit should padded y bytes 0x00 automatically (2+x+y should be multiple of 8). For AES mode: 2+x should be multiple of 16. If it was not multiple of 16, unit should padded y bytes 0x00 automatically (2+x+y should be multiple of 16).

  Parameters

cLen	data packet length
reData	response encrypted APDU response. Can be three options:

[00] + [Plaintext R-APDU]

- [01] + [0A] + [KSN] + [n bytes Encrypted R-APDU without Status Bytes] + [2 bytes Status Bytes]
- [01] + [00] + [n bytes Encrypted R-APDU without Status Bytes] + [2 bytes Status Bytes]

  The KSN, when provided, will be 10 bytes. The KSN will only be provided when it has changed since the last provided KSN. Each card Power-On generates a new KSN. During a sequence of commands where the KSN is identical, the first response will have a KSN length set to [0x0A] followed by the KSN, while subsequent commands with the same KSN value will have a KSN length of [0x00] followed by the Encrypted R-APDU without Status Bytes.

  Parameters

reLen	encrypted APDU response data length

  Returns

  RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.2.4.178 int icc_getAPDU_KSN (OUT BYTE * *KSN,* IN_OUT int * *inLen*)

Get APDU KSN - AUGUSTA

Retrieves the KSN used in ICC Encypted APDU usage

Parameters

KSN	Returns the encrypted APDU packet KSN
inLen	KSN data length

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.2.4.179 int icc_getFunctionStatus (OUT int * *enabled,* OUT int * *withNotification*)

Get ICC Function status - AUGUSTA Get ICC Function status about enable/disable and with or without seated notification

Parameters

enabled	• 1: ICC Function enabled, • 0: means disabled.
withNotification	1 means with notification when ICC seated status changed. 0 means without notification.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.2.4.180 int icc_getICCReaderStatus (OUT BYTE * *status*)

Get Reader Status - AUGUSTA

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

12.2.4.181 int icc_getKeyFormatForICCDUKPT (OUT BYTE * *format*)

Get Key Format For DUKPT - AUGUSTA

Specifies how data will be encrypted with Data Key or PIN key (if DUKPT key loaded). This applies to both MSR and ICC

Parameters

format	Response returned from method: • 'TDES': Encrypted card data with TDES if DUKPT Key had been loaded.(default) • 'AES': Encrypted card data with AES if DUKPT Key had been loaded. • 'NONE': No Encryption.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.2.4.182 int icc_getKeyTypeForICCDUKPT (OUT BYTE * *type*)

Get Key Type for DUKPT - AUGUSTA

Specifies the key type used for ICC DUKPT encryption This applies to both MSR and ICC

Parameters

type	Response returned from method: • 'DATA': Encrypted card data with Data Key DUKPT Key had been loaded. (default) • 'PIN': Encrypted card data with PIN Key if DUKPT Key had been loaded.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.2.4.183 int icc_powerOffICC ( )

Power Off ICC

Powers down the ICC

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

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

12.2.4.184 int icc_powerOnICC (OUT BYTE * *ATR,* IN_OUT int * *inLen*)

Power On ICC

Power up the currently selected microprocessor card in the ICC reader

Parameters

ATR,the	ATR data response when succeeded power on ICC,
inLen,the	length of ATR data

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.2.4.185 int icc_setKeyFormatForICCDUKPT (IN BYTE *format*)

Set Key Format for DUKPT - AUGUSTA

Sets how data will be encrypted, with either TDES or AES (if DUKPT key loaded) This applies to both MSR and ICC

Parameters

format	encryption Encryption Type • 00: Encrypt with TDES • 01: Encrypt with AES • 02: Encrypt with TransArmor - AUGUSTA only

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.2.4.186 int icc_setKeyTypeForICCDUKPT (IN BYTE *type*)

Set Key Type for DUKPT Key - AUGUSTA

Sets which key the data will be encrypted with, with either Data Key or PIN key (if DUKPT key loaded) This applies to both MSR and ICC

Parameters

type	Encryption Type • 00: Encrypt with Data Key • 01: Encrypt with PIN Key

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.2.4.187 int iso8583_deserializeFromXML (IN BYTE * *serializedMessage*, IN int *serializedMessageLength*, OUT DL_ISO8583_HANDLER * *ISOHandler*, OUT DL_ISO8583_MSG * *ISOMessage*)

Deserialize the XML-formatted ISO8583 message.

Parameters

serialized-Message	- The XML-formatted message
serialized-MessageLength	- The length of the XML-formatted message
ISOHandler	- A null ISO8583 handler
ISOMessage	- The ISO8583 message structure

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.2.4.188 int iso8583_displayMessage (IN DL_ISO8583_HANDLER * *ISOHandler*, IN DL_ISO8583_MSG * *ISOMessage*)

Display the messages in a formatted manner on the screen for verifying results.

Parameters

ISOHandler	- The ISO8583 handler
ISOMessage	- The ISO8583 message structure

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.2.4.189 int iso8583_freeMessage (IN DL_ISO8583_MSG * *ISOMessage*)

Deallocate the ISO8583 message structure's memory.

Parameters

ISOMessage	- The ISO8583 message structure

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.2.4.190 int iso8583_get1987Handler (OUT DL_ISO8583_HANDLER * *ISOHandler*)

Get the ISO8583 1987 version handler.

Parameters

ISOHandler	A handler with knowledge of the ISO8583 1987 version fields

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.2.4.191 int iso8583_get1993Handler (OUT DL_ISO8583_HANDLER * *ISOHandler*)

Get the ISO8583 1993 version handler.

Parameters

ISOHandler	A handler with knowledge of the ISO8583 1993 version fields

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.2.4.192 int iso8583_get2003Handler (OUT DL_ISO8583_HANDLER * *ISOHandler*)

Get the ISO8583 2003 version handler.

Parameters

ISOHandler	A handler with knowledge of the ISO8583 2003 version fields

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.2.4.193 int iso8583_getField (IN DL_UINT16 *dataField,* IN DL_ISO8583_HANDLER * *ISOHandler,* OUT DL_ISO8583_FIELD_DEF * *field*)

Get the specified field's information using the data field.

Parameters

dataField	- The data field number

ISOHandler	- The ISO8583 handler
field	- The requested field

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.2.4.194 int iso8583_getMessageField (IN DL_UINT16 *dataField*, IN DL_ISO8583_MSG * *ISOMessage*, OUT DL_ISO8583_MSG_FIELD * *messageField*)

Get the specified message field using the data field.

Parameters

dataField	- The data field number
ISOMessage	- The ISO8583 message structure
messageField	- The requested message field

Returns

0 if the if the setting was applied; otherwise, return -1 on failure

12.2.4.195 int iso8583_initializeMessage (OUT DL_ISO8583_MSG * *ISOMessage*)

Initialize the ISO8583 message structure.

Parameters

ISOMessage	- The initialized ISO8583 message structure

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.2.4.196 int iso8583_packMessage (IN const DL_ISO8583_HANDLER * *ISOHandler*, IN const DL_ISO8583_MSG * *ISOMessage*, OUT DL_UINT8 * *packedData*, OUT DL_UINT16 * *packedDataLength*)

Pack the message fields into an array.

Parameters

ISOHandler	- The ISO8583 handler
ISOMessage	- The ISO8583 message structure
packedData	- The packaged data
packedData-Length	- The packaged data's length

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.2.4.197 int iso8583_removeMessageField (IN DL_UINT16 *dataField*, OUT DL_ISO8583_MSG * *ISOMessage*)

Remove the specified message field.

Parameters

dataField	- The data field number
ISOMessage	- The ISO8583 message structure

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.2.4.198 int iso8583_serializeToXML (IN DL_ISO8583_HANDLER * *ISOHandler,* IN DL_ISO8583_MSG * *ISOMessage,* OUT BYTE * *serializedMessage,* OUT int * *serializedMessageLength*)

Serialize the message fields into an XML format.

Parameters

ISOHandler	- The ISO8583 handler
ISOMessage	- The ISO8583 message structure
serialized-Message	- The XML-formatted message
serialized-MessageLength	- The XML message's length

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.2.4.199 int iso8583_setMessageField (IN DL_UINT16 *dataField,* IN const DL_UINT8 * *data,* OUT DL_ISO8583_MSG * *ISOMessage*)

Set the specified message field.

Parameters

dataField	- The data field number
data	- The data to apply
ISOMessage	- The ISO8583 message structure

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.2.4.200 int iso8583_unpackMessage (IN const DL_ISO8583_HANDLER * *ISOHandler,* IN const DL_UINT8 * *packedData,* IN DL_UINT16 *packedDataLength,* OUT DL_ISO8583_MSG * *ISOMessage*)

Unpack the message field array into the ISO8583 message structure.

Parameters

ISOHandler	- The ISO8583 handler
packedData	- The packaged data
packedData-Length	- The packaged data's length
ISOMessage	- The ISO8583 message structure

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.2.4.201 int lcd_addButton (IN char * *screenName,* IN int *screenNameLen,* IN char * *buttonName,* IN int *buttonNameLen,* IN BYTE *type,* IN BYTE *alignment,* IN int *xCord,* IN int *yCord,* IN char * *label,* IN int *labelLen,* OUT IDTLCDItem * *returnItem*)

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
screenNameLen	Length of screenName
buttonName	Button name that will be the target of add button
buttonNameLen	Length of buttonName
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
label	Label to show on the button. Required for Large/Medium buttons. Not used for Invisible buttons.
labelLen	Length of label
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

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.2.4.202 int lcd_addEthernet (IN char * *screenName,* IN int *screenNameLen,* IN char * *objectName,* IN int *objectNameLen,* IN BYTE *alignment,* IN int *xCord,* IN int *yCord,* OUT IDTLCDItem * *returnItem*)

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
screenNameLen	Length of screenName
objectName	Object name that will be the target of add widget
objectNameLen	Length of objectName
alignment	Position for widget • 0 = Display object at the horizon center of specified y, while x ignored • 1 = Display object at specified x and y • 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

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

12.2.4.203 int lcd_addImage (IN char * *screenName,* IN int *screenNameLen,* IN char * *objectName,* IN int *objectNameLen,* IN BYTE *alignment,* IN int *xCord,* IN int *yCord,* IN char * *filename,* IN int *filenameLen,* OUT IDTLCDItem * *returnItem*)

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
screenNameLen	Length of screenName
objectName	Object name that will be the target of add image
objectNameLen	Length of objectName
alignment	Position for Image • 0 = Display object at the horizon center of specified y, while x ignored • 1 = Display object at specified x and y • 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.
filenameLen	Length of filename
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

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

12.2.4.204 int lcd_addItemToList (IN BYTE * *listGraphicsID,* IN char * *itemName,* IN char * *itemID,* IN int *selected*)

Adds an item to an existing list.

Custom Display Mode must be enabled for custom text.

Parameters

listGraphicsID	Existing list's graphics ID (4 byte array) that is provided during creation
itemName	Item name (Maximum: 127 characters)
itemID	Identifier for the item (Maximum: 31 characters)
selected	If the item should be selected

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.2.4.205 int lcd_addLED (IN char * *screenName,* IN int *screenNameLen,* IN char * *objectName,* IN int *objectNameLen,* IN BYTE *alignment,* IN int *xCord,* IN int *yCord,* OUT IDTLCDItem * *returnItem,* IN BYTE * *LED,* IN int *LEDLen*)

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
screenNameLen	Length of screenName
objectName	Object name that will be the target of add LED
objectNameLen	Length of objectName
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
LEDLen	Length of LED

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

12.2.4.206 int lcd_addText (IN char * *screenName,* IN int *screenNameLen,* IN char * *objectName,* IN int *objectNameLen,* IN BYTE *alignment,* IN int *xCord,* IN int *yCord,* IN int *width,* IN int *height,* IN BYTE *fontID,* IN BYTE * *color,* IN int *colorLen,* IN char * *label,* IN int *labelLen,* OUT IDTLCDItem * *returnItem*)

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
screenNameLen	Length of screenName
objectName	Object name that will be the target of add text
objectNameLen	Length of objectName
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
colorLen	Length of color
label	Label to show on the text
labelLen	Length of label
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

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
14	RoundCondensed-Medium_12	RoundCondensed-Medium_0.ttf	12
15	RoundCondensed-Medium_18	RoundCondensed-Medium_0.ttf	18
16	RoundCondensed-Medium_24	RoundCondensed-Medium_0.ttf	24
17	RoundCondensed-Medium_36	RoundCondensed-Medium_0.ttf	36
18	RoundCondensed-Medium_48	RoundCondensed-Medium_0.ttf	48
19	RoundCondensed-Medium_60	RoundCondensed-Medium_0.ttf	60
20	RoundCondensed-Medium_72	RoundCondensed-Medium_0.ttf	72
21	RoundCondensed-Semibold_12	RoundCondensed-Semibold.ttf	12
22	RoundCondensed-Semibold_18	RoundCondensed-Semibold.ttf	18
23	RoundCondensed-Semibold_24	RoundCondensed-Semibold.ttf	24
24	RoundCondensed-Semibold_36	RoundCondensed-Semibold.ttf	36
25	RoundCondensed-Semibold_48	RoundCondensed-Semibold.ttf	48
26	RoundCondensed-Semibold_60	RoundCondensed-Semibold.ttf	60
27	RoundCondensed-Semibold_72	RoundCondensed-Semibold.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

12.2.4.207 int lcd_addVideo (IN char * *screenName,* IN int *screenNameLen,* IN char * *objectName,* IN int *objectNameLen,* IN BYTE *alignment,* IN int *xCord,* IN int *yCord,* IN char * *filename,* IN int *filenameLen,* OUT IDTLCDItem * *returnItem*)

Add Video

Adds a video 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 video
screenNameLen	Length of screenName
objectName	Object name that will be the target of add video
objectNameLen	Length of objectName
alignment	Position for Video • 0 = Display object at the horizon center of specified y, while x ignored • 1 = Display object at specified x and y • 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 Video, range 0-271
yCord	y-coordinate for Video, range 0-479
filename	Filename of the video. Must be available in the sd card.
filenameLen	Length of filename
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 video

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
video	1

12.2.4.208 int lcd_cancelSlideShow (OUT BYTE * *statusCode,* IN_OUT int * *statusCodeLen*)

Cancel slide show Cancel the slide show currently running

Parameters

statusCode	If the return code is not Success (0), the kernel may return a four-byte Extended Status Code
statusCodeLen	the length of the Extended Status Code (should be 4 bytes)

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.2.4.209 int lcd_captureSignature (IN int *timeout*)

Enables Signature Capture This command executes the signature capture screen. Once a signature is captured, it is sent to the callback with DeviceState.Signature, and the data will contain a .png of the signature

Parameters

timeout	Timeout waiting for the signature capture

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.2.4.210 int lcd_clearDisplay (IN BYTE *control*)

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

Parameters

control	for L100 only. 0:First Line 1:Second Line 2:Third Line 3:Fourth Line 0xFF: All Screen

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.2.4.211 int lcd_clearEventQueue ( )

Removes all entries from the event queue.

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.2.4.212 int lcd_clearScreenInfo ( )

Clear Screen Info

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

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.2.4.213 int lcd_cloneScreen (IN char * *screenName,* IN int *screenNameLen,* IN char * *cloneName,* IN int *cloneNameLen,* OUT int * *cloneID*)

Clone Screen

Clones an existing screen.

Parameters

screenName	Screen name to clone.
screenNameLen	Length of screenName.
cloneName	Name of the cloned screen.
cloneNameLen	Length of cloneName.
cloneID	Screen ID of the cloned screen

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.2.4.214 int lcd_createInputField (IN BYTE * *specs*, IN int *specsLen*, OUT BYTE * *graphicId*)

DEPRECATED : please use lcd_createInputField_Len(IN BYTE *specs, IN int specsLen, OUT BYTE *graphicId, IN_OUT int *graphicIdLen)

Create an input field on the screen.

Parameters

specs	The specs of the input field:

Length (bytes)	Description
2 - 4	X coordinate in pixels, zero terminated ASCII
----------—	----------—
2 - 4	Y coordinate in pixels, zero terminated ASCII
----------—	----------—
2 - 4	Width in pixels, zero terminated ASCII. Set to 0 (30h) for calculated width.
----------—	----------—
2 - 4	Height in pixels, zero terminated ASCII. Set to 0 (30h) for calculated height.
----------—	----------—
2	Font designation. Default font = 1, zero terminated ASCII
----------—	----------—
2 - 3	Zero terminated ASCII Font ID
----------—	----------—
3	Zero terminated ASCII hexadecimal display option flag
	Bit 0 0 = No Border
	1 = Show Border
	Bit 1 0 = Characters are first displayed on the leftmost area of the screen.
	1 = The first character entered is displayed on the rightmost area of
	the screen, and, as further digits are entered, characters scroll

	from the right to the left.
	Bit 2 - 15 Reserved
----------——	----------——
1 or 9	Foreground color, zero terminated ASCII hexadecimal
----------——	----------——
1 or 9	Background color, zero terminated ASCII hexadecimal
----------——	----------——
1 or 9	Border color, zero terminated ASCII hexadecimal
----------——	----------——
1 - 65	Prefill String, zero terminated ASCII
----------——	----------——
1 - 65	Format String, zero terminated ASCII

Parameters

specsLen	The length of specs
graphicsID	The graphicID of the event (required to be 4 bytes)

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.2.4.215 int lcd_createInputField_Len (IN BYTE * *specs*, IN int *specsLen*, OUT BYTE * *graphicId*, IN_OUT int * *graphicIdLen*)

Create an input field on the screen.

Parameters

specs	The specs of the input field:

Length (bytes)	Description
2 - 4	X coordinate in pixels, zero terminated ASCII
----------——	----------——
2 - 4	Y coordinate in pixels, zero terminated ASCII
----------——	----------——
2 - 4	Width in pixels, zero terminated ASCII. Set to 0 (30h) for calculated width.
----------——	----------——
2 - 4	Height in pixels, zero terminated ASCII. Set to 0 (30h) for calculated height.
----------——	----------——
2	Font designation. Default font = 1, zero terminated ASCII
----------——	----------——

2 - 3	Zero terminated ASCII Font ID
----------—	----------—
3	Zero terminated ASCII hexadecimal display option flag
	Bit 0 0 = No Border
	1 = Show Border
	Bit 1 0 = Characters are first displayed on the leftmost area of the screen.
	1 = The first character entered is displayed on the rightmost area of
	the screen, and, as further digits are entered, characters scroll
	from the right to the left.
	Bit 2 - 15 Reserved
----------—	----------—
1 or 9	Foreground color, zero terminated ASCII hexadecimal
----------—	----------—
1 or 9	Background color, zero terminated ASCII hexadecimal
----------—	----------—
1 or 9	Border color, zero terminated ASCII hexadecimal
----------—	----------—
1 - 65	Prefill String, zero terminated ASCII
----------—	----------—
1 - 65	Format String, zero terminated ASCII

Parameters

specsLen	The length of specs
graphicsID	The graphicID of the event (required to be 4 bytes)
graphicsIDLen	Length of graphicsID

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.2.4.216 int lcd_createList (IN int *posX,* IN int *posY,* IN int *numOfColumns,* IN int *numOfRows,* IN int *fontDesignation,* IN int *fontID,* IN int *verticalScrollArrowsVisible,* IN int *borderedListItems,* IN int *borderdScrollArrows,* IN int *touchSensitive,* IN int *automaticScrolling,* OUT BYTE ∗ *graphicsID*)

DEPRECATED : please use lcd_createList_Len(IN int posX, IN int posY, IN int numOfColumns, IN int numOfRows, IN int fontDesignation, IN int fontID, IN int verticalScrollArrowsVisible, IN int borderedListItems, IN int borderdScrollArrows, IN int touchSensitive, IN int automaticScrolling, OUT BYTE ∗graphicsID, IN_OUT int ∗graphicsIDLen)

Creates a display list.

Parameters

posX	X coordinate in pixels
posY	Y coordinate in pixels

numOfColumns	Number of columns to display
numOfRows	Number of rows to display
fontDesignation	Font Designation 1 - Default font
fontID	Font styling | Font ID	Height in pixels	Font Properties
---	---	---
1	13	Regular
2	17	Regular
3	17	Bold
4	22	Regular
5	20	Regular
6	20	Bold
7	29	Regular
8	38	Regular
9	38	Bold
10	58	Regular
11	58	Bold, mono-space
12	14	Regular, mono-space, 8 pixels wide
13	15	Regular, mono-space, 9 pixels wide
14	17	Regular, mono-space, 9 pixels wide
15	20	Regular, mono-space, 11 pixels wide
16	21	Regular, mono-space, 12 pixels wide
17	25	Regular, mono-space, 14 pixels wide
18	30	Regular, mono-space, 17 pixels wide	
verticalScroll-ArrowsVisible	Display vertical scroll arrows by default
borederedList-Items	Draw border around list items
borederedScroll-Arrows	Draw border around scroll arrows (if visible)
touchSensitive	List items are touch enabled
automatic-Scrolling	Enable automatic scrolling of list when new items exceed display area
graphicsID	A four byte array containing the ID of the created element (optional) if graphicsID is NULL, the SDK will not return graphicsID if graphicsID is not NULL, the SDK will return graphicsID, but it will need 4 bytes of memory

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.2.4.217 int lcd_createList_Len (IN int *posX*, IN int *posY*, IN int *numOfColumns*, IN int *numOfRows*, IN int *fontDesignation*, IN int *fontID*, IN int *verticalScrollArrowsVisible*, IN int *borderedListItems*, IN int *borderdScrollArrows*, IN int *touchSensitive*, IN int *automaticScrolling*, OUT BYTE * *graphicsID*, IN_OUT int * *graphicsIDLen*)

Creates a display list.

Parameters

posX	X coordinate in pixels
posY	Y coordinate in pixels
numOfColumns	Number of columns to display
numOfRows	Number of rows to display
fontDesignation	Font Designation 1 - Default font
fontID	Font styling | Font ID | Height in pixels | Font Properties | | ------- | ---------------- | ---------- | | 1 | 13 | Regular | | 2 | 17 | Regular | | 3 | 17 | Bold | | 4 | 22 | Regular | | 5 | 20 | Regular | | 6 | 20 | Bold | | 7 | 29 | Regular | | 8 | 38 | Regular | | 9 | 38 | Bold | | 10 | 58 | Regular | | 11 | 58 | Bold, mono-space | | 12 | 14 | Regular, mono-space, 8 pixels wide | | 13 | 15 | Regular, mono-space, 9 pixels wide | | 14 | 17 | Regular, mono-space, 9 pixels wide | | 15 | 20 | Regular, mono-space, 11 pixels wide | | 16 | 21 | Regular, mono-space, 12 pixels wide | | 17 | 25 | Regular, mono-space, 14 pixels wide | | 18 | 30 | Regular, mono-space, 17 pixels wide |
verticalScroll-ArrowsVisible	Display vertical scroll arrows by default
borederedList-Items	Draw border around list items
borederedScroll-Arrows	Draw border around scroll arrows (if visible)
touchSensitive	List items are touch enabled
automatic-Scrolling	Enable automatic scrolling of list when new items exceed display area
graphicsID	A four byte array containing the ID of the created element (optional)
graphicsIDLen	Length of graphicsID (optional)

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.2.4.218 int lcd_createScreen (IN char * *screenName*, IN int *screenNameLen*, OUT int * *ScreenID*)

Create Screen

Creates a new screen.

Parameters

screenName	Screen name to use.
screenNameLen	Length of screenName.
screenID	Screen ID that was created.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.2.4.219 int lcd_customDisplayMode (IN int *enable*)

Custom Display Mode Controls the LCD display mode to custom display. Keyboard entry is limited to the Cancel, Clear, Enter and the function keys, if present. PIN entry is not permitted while the reader is in Custom Display Mode

Parameters

enable	TRUE = enabled, FALSE = disabled

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.2.4.220 int lcd_destroyScreen (IN char * *screenName,* IN int *screenNameLen*)

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.
screenNameLen	Length of screenName.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.2.4.221 int lcd_displayButton (IN int *posX,* IN int *posY,* IN int *buttonWidth,* IN int *buttonHeight,* IN int *fontDesignation,* IN int *fontID,* IN int *displayPosition,* IN char * *buttonLabel,* IN int *buttonTextColorR,* IN int *buttonTextColorG,* IN int *buttonTextColorB,* IN int *buttonBackgroundColorR,* IN int *buttonBackgroundColorG,* IN int *buttonBackgroundColorB,* OUT BYTE * *graphicsID*)

DEPRECATED : please use lcd_displayButton_Len(IN int posX, IN int posY, IN int buttonWidth, IN int buttonHeight, IN int fontDesignation, IN int fontID, IN int displayPosition, IN char *buttonLabel, IN int buttonTextColorR, IN int buttonTextColorG, IN int buttonTextColorB, IN int buttonBackgroundColorR, IN int buttonBackgroundColorG, IN int buttonBackgroundColorB, OUT BYTE *graphicsID, IN_OUT int *graphicsIDLen)

Displays an interactive button.

Parameters

posX	X coordinate in pixels
posY	Y coordinate in pixels
buttonWidth	Width of the button
buttonHeight	Height of the button
fontDesignation	Font designation 1 - Default
Font	ID Font styling | Font ID | Height in pixels | Font Properties | | --- | --- | --- | | 1 | 13 | Regular | | 2 | 17 | Regular | | 3 | 17 | Bold | | 4 | 22 | Regular | | 5 | 20 | Regular | | 6 | 20 | Bold | | 7 | 29 | Regular | | 8 | 38 | Regular | | 9 | 38 | Bold | | 10 | 58 | Regular | | 11 | 58 | Bold, mono-space | | 12 | 14 | Regular, mono-space, 8 pixels wide | | 13 | 15 | Regular, mono-space, 9 pixels wide | | 14 | 17 | Regular, mono-space, 9 pixels wide | | 15 | 20 | Regular, mono-space, 11 pixels wide | | 16 | 21 | Regular, mono-space, 12 pixels wide | | 17 | 25 | Regular, mono-space, 14 pixels wide | | 18 | 30 | Regular, mono-space, 17 pixels wide |
displayPosition	Button display position 0 - Center on line Y without clearing screen and without word wrap 1 - Center on line Y after clearing screen and without word wrap 2 - Display at (X, Y) without clearing screen and without word wrap 3 - Display at (X, Y) after clearing screen and without word wrap 4 - Center button on screen without clearing screen and without word wrap 5 - Center button on screen after clearing screen and without word wrap 64 - Center on line Y without clearing screen and with word wrap 65 - Center on line Y after clearing the screen and with word wrap 66 - Display at (X, Y) without clearing screen and with word wrap 67 - Display at (X, Y) after clearing screen and with word wrap 68 - Center button on screen without clearing screen and with word wrap 69 - Center button on screen after clearing screen and with word wrap
buttonLabel	Button label text (Maximum: 31 characters)
buttonTextColorR	- Red component for foreground color (0 - 255)
buttonTextColorG	- Green component for foreground color (0 - 255)
buttonTextColorB	- Blue component for foreground color (0 - 255)
buttonBackgroundColorR	- Red component for background color (0 - 255)
buttonBackgroundColorG	- Green component for background color (0 - 255)

buttonBackgroundColorB	- Blue component for background color (0 - 255)
graphicsID	A four byte array containing the ID of the created element (optional) if graphicsID is NULL, the SDK will not return graphicsID if graphicsID is not NULL, the SDK will return graphicsID, but it will need 4 bytes of memory

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.2.4.222 int lcd_displayButton_Len (IN int *posX*, IN int *posY*, IN int *buttonWidth*, IN int *buttonHeight*, IN int *fontDesignation*, IN int *fontID*, IN int *displayPosition*, IN char * *buttonLabel*, IN int *buttonTextColorR*, IN int *buttonTextColorG*, IN int *buttonTextColorB*, IN int *buttonBackgroundColorR*, IN int *buttonBackgroundColorG*, IN int *buttonBackgroundColorB*, OUT BYTE * *graphicsID*, IN_OUT int * *graphicsIDLen*)

Displays an interactive button.

Parameters

posX	X coordinate in pixels
posY	Y coordinate in pixels
buttonWidth	Width of the button
buttonHeight	Height of the button
fontDesignation	Font designation 1 - Default
Font	ID Font styling

Font ID	Height in pixels	Font Properties
1	13	Regular
2	17	Regular
3	17	Bold
4	22	Regular
5	20	Regular
6	20	Bold
7	29	Regular
8	38	Regular
9	38	Bold
10	58	Regular
11	58	Bold, mono-space
12	14	Regular, mono-space, 8 pixels wide
13	15	Regular, mono-space, 9 pixels wide
14	17	Regular, mono-space, 9 pixels wide
15	20	Regular, mono-space, 11 pixels wide
16	21	Regular, mono-space, 12 pixels wide
17	25	Regular, mono-space, 14 pixels wide
18	30	Regular, mono-space, 17 pixels wide

displayPosition	Button display position 0 - Center on line Y without clearing screen and without word wrap 1 - Center on line Y after clearing screen and without word wrap 2 - Display at (X, Y) without clearing screen and without word wrap 3 - Display at (X, Y) after clearing screen and without word wrap 4 - Center button on screen without clearing screen and without word wrap 5 - Center button on screen after clearing screen and without word wrap 64 - Center on line Y without clearing screen and with word wrap 65 - Center on line Y after clearing the screen and with word wrap 66 - Display at (X, Y) without clearing screen and with word wrap 67 - Display at (X, Y) after clearing screen and with word wrap 68 - Center button on screen without clearing screen and with word wrap 69 - Center button on screen after clearing screen and with word wrap
buttonLabel	Button label text (Maximum: 31 characters)
buttonTextColor-R	- Red component for foreground color (0 - 255)
buttonTextColor-G	- Green component for foreground color (0 - 255)
buttonTextColor-B	- Blue component for foreground color (0 - 255)
button-Background-ColorR	- Red component for background color (0 - 255)
button-Background-ColorG	- Green component for background color (0 - 255)
button-Background-ColorB	- Blue component for background color (0 - 255)
graphicsID	A four byte array containing the ID of the created element (optional)
graphicsIDLen	Length of graphicsID (optional)

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.2.4.223 int lcd_displayMessage (int *lineNumber,* char * *message,* int *messageLen*)

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
messageLen	length of message

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.2.4.224 int lcd_displayParagraph (IN int *posX,* IN int *posY,* IN int *displayWidth,* IN int *displayHeight,* IN int *fontDesignation,* IN int *fontID,* IN int *displayProperties,* IN char * *displayText*)

Displays text with scroll feature.

Custom Display Mode must be enabled.

Parameters

posX	X coordinate in pixels
posY	Y coordinate in pixels
displayWidth	Width of the display area in pixels (Minimum: 40px) 0 or NULL - Use the full width to display text
displayHeight	Height of the display area in pixels (Minimum: 100px) 0 or NULL - Use the full height to display text
fontDesignation	Font designation 1 - Default
fontID	Font styling | Font ID | Height in pixels | Font Properties | | --- | --- | --- | | 1 | 13 | Regular | | 2 | 17 | Regular | | 3 | 17 | Bold | | 4 | 22 | Regular | | 5 | 20 | Regular | | 6 | 20 | Bold | | 7 | 29 | Regular | | 8 | 38 | Regular | | 9 | 38 | Bold | | 10 | 58 | Regular | | 11 | 58 | Bold, mono-space | | 12 | 14 | Regular, mono-space, 8 pixels wide | | 13 | 15 | Regular, mono-space, 9 pixels wide | | 14 | 17 | Regular, mono-space, 9 pixels wide | | 15 | 20 | Regular, mono-space, 11 pixels wide | | 16 | 21 | Regular, mono-space, 12 pixels wide | | 17 | 25 | Regular, mono-space, 14 pixels wide | | 18 | 30 | Regular, mono-space, 17 pixels wide |
display-Properties	Display properties for the text 0 - Center on line Y without clearing screen 1 - Center on line Y after clearing screen 2 - Display at (X, Y) without clearing screen 3 - Display at (X, Y) after clearing screen 4 - Center on screen without clearing screen 5 - Center on screen after clearing screen
displayText	Display text (Maximum: 3999 characters)

12.2.4.225 int 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

12.2.4.226 int lcd_displayText (IN int *posX,* IN int *posY,* IN int *displayWidth,* IN int *displayHeight,* IN int *fontDesignation,* IN int *fontID,* IN int *screenPosition,* IN char * *displayText,* OUT BYTE * *graphicsID*)

DEPRECATED : please use lcd_displayText_Len(IN int posX, IN int posY, IN int displayWidth, IN int displayHeight, IN int fontDesignation, IN int fontID, IN int screenPosition, IN char *displayText, OUT BYTE *graphicsID, IN_OUT int *graphicsIDLen)

Displays text.

Custom Display Mode must be enabled for custom text. PIN pad entry is not allowed in Custom Display Mode but the Cancel, OK, and Clear keys remain active.

Parameters

posX	X coordinate in pixels
posY	Y coordinate in pixels
displayWidth	Width of the display area in pixels (optional)
displayHeight	Height of the display area in pixels (optional)
fontDesignation	Font designation 1 - Default
Font	ID Font styling | Font ID | Height in pixels | Font Properties | | --- | --- | --- | | 1 | 13 | Regular | | 2 | 17 | Regular | | 3 | 17 | Bold | | 4 | 22 | Regular | | 5 | 20 | Regular | | 6 | 20 | Bold | | 7 | 29 | Regular | | 8 | 38 | Regular | | 9 | 38 | Bold | | 10 | 58 | Regular | | 11 | 58 | Bold, mono-space | | 12 | 14 | Regular, mono-space, 8 pixels wide | | 13 | 15 | Regular, mono-space, 9 pixels wide | | 14 | 17 | Regular, mono-space, 9 pixels wide | | 15 | 20 | Regular, mono-space, 11 pixels wide | | 16 | 21 | Regular, mono-space, 12 pixels wide | | 17 | 25 | Regular, mono-space, 14 pixels wide | | 18 | 30 | Regular, mono-space, 17 pixels wide |
screenPosition	Display position 0 - Center on line Y without clearing screen 1 - Center on line Y after clearing screen 2 - Display at (X, Y) without clearing screen 3 - Display at (X, Y) after clearing screen 4 - Display at center of screen without clearing screen 5 - Display at center of screen after clearing screen 6 - Display text right-justified without clearing screen 7 - Display text right-justified after clearing screen
displayText	Display text (Maximum: 1900 characters)
graphicsID	A four byte array containing the ID of the created element (optional) if graphicsID is NULL, the SDK will not return graphicsID if graphicsID is not NULL, the SDK will return graphicsID, but it will need 4 bytes of memory

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.2.4.227 int lcd_displayText_Len (IN int *posX*, IN int *posY*, IN int *displayWidth*, IN int *displayHeight*, IN int *fontDesignation*, IN int *fontID*, IN int *screenPosition*, IN char * *displayText*, OUT BYTE * *graphicsID*, IN_OUT int * *graphicsIDLen*)

Displays text.

Custom Display Mode must be enabled for custom text. PIN pad entry is not allowed in Custom Display Mode but the Cancel, OK, and Clear keys remain active.

Parameters

posX	X coordinate in pixels
posY	Y coordinate in pixels
displayWidth	Width of the display area in pixels (optional)

displayHeight	Height of the display area in pixels (optional)
fontDesignation	Font designation 1 - Default
Font	ID Font styling | Font ID | Height in pixels | Font Properties | | --- | --- | --- | | 1 | 13 | Regular | | 2 | 17 | Regular | | 3 | 17 | Bold | | 4 | 22 | Regular | | 5 | 20 | Regular | | 6 | 20 | Bold | | 7 | 29 | Regular | | 8 | 38 | Regular | | 9 | 38 | Bold | | 10 | 58 | Regular | | 11 | 58 | Bold, mono-space | | 12 | 14 | Regular, mono-space, 8 pixels wide | | 13 | 15 | Regular, mono-space, 9 pixels wide | | 14 | 17 | Regular, mono-space, 9 pixels wide | | 15 | 20 | Regular, mono-space, 11 pixels wide | | 16 | 21 | Regular, mono-space, 12 pixels wide | | 17 | 25 | Regular, mono-space, 14 pixels wide | | 18 | 30 | Regular, mono-space, 17 pixels wide |
screenPosition	Display position 0 - Center on line Y without clearing screen 1 - Center on line Y after clearing screen 2 - Display at (X, Y) without clearing screen 3 - Display at (X, Y) after clearing screen 4 - Display at center of screen without clearing screen 5 - Display at center of screen after clearing screen 6 - Display text right-justified without clearing screen 7 - Display text right-justified after clearing screen
displayText	Display text (Maximum: 1900 characters)
graphicsID	A four byte array containing the ID of the created element (optional)
graphicsIDLen	Length of graphicsID (optional)

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.2.4.228 int lcd_enableBacklight (int *enable*)

Enable/Disable LCD Backlight Trns 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

12.2.4.229 int lcd_getActiveScreen (OUT char * *screenName*, IN_OUT int * *screenNameLen*)

Get Active Screen

Returns the active screen ID.

Parameters

screenName	Screen name this is active.
screenNameLen	Length of screenName.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.2.4.230 int lcd_getAllObjects (IN char * *screenName*, IN int *screenNameLen*, IN_OUT int * *objectNumbers*, OUT IDTObjectInfo * *objectInfo*)

Get All Objects on Screen

Get all created objects' name on certain screen

Parameters

screenName	Screen name to get all objects
screenNameLen	Length of screenName
objectNumbers	Number of created objects
returnObjects	Array of all created objects each element includes -objectID of a created object -objectName of a created object -objectNameLen of objectName

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.2.4.231 int lcd_getAllScreens (IN_OUT int * *screenNumbers*, OUT IDTScreenInfo * *screenInfo*)

Get All Screens

Get all created screens' name

Parameters

screenNumbers	Number of created screens
screenInfo	Array of all created screens each element includes -screenID of a created screen -screenName of a created screen -screenNameLen of screenName

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.2.4.232 int lcd_getBacklightStatus (int * *enabled*)

Get Backlight Status

Returns the status of the LCD back lighting.

Parameters

enabled	1 = Backlight is ON, 0 = Backlight is OFF

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.2.4.233 int lcd_getButtonEvent (OUT int * *screenID,* OUT int * *objectID,* OUT char * *screenName,* IN_OUT int * *screenNameLen,* OUT char * *objectName,* IN_OUT int * *objectNameLen,* OUT int * *isLongPress*)

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
screenNameLen	Length of screenName
objectName	Button Name of the last clicked button
objectNameLen	Length of objectName
isLongPress	1 = Long Press, 0 = Short Press
ip	Optional: IP address to execute command on (for IP connected devices)

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.2.4.234 int lcd_getInputEvent (IN int *timeout,* OUT int * *dataReceived,* OUT BYTE * *eventType,* OUT BYTE * *graphicsID,* OUT BYTE * *eventData*)

DEPRECATED : please use lcd_getInputEvent_Len(IN int timeout, OUT int *dataReceived, OUT BYTE *eventType, IN_OUT int *eventTypeLen, OUT BYTE *graphicsID, IN_OUT int *graphicsIDLen, OUT BYTE *eventData, IN_OUT int *eventDataLen)

Requests input from the reader.

Parameters

timeout	Timeout amount in seconds 0 - No timeout
dataReceived	Indicates if an event occurred and data was received 0 - No data received 1 - Data received
eventType	The event type (required to be at least 4 bytes), see table below
graphicsID	The graphicID of the event (required to be at least 4 bytes)
eventData	The event data, see table below (required to be at least 73 bytes)

Event Type	Value (4 bytes)	Event Specific Data
Button Event	00030000h	Length = Variable
		Byte 1: State (1 = Pressed, other values RFU)
		Byte 2 - n: Null terminated caption
----------	----------	----------
Checkbox Event	00030001h	Length = 1 byte
		Byte 1: State (1 = Checked, 0 = Unchecked)
----------	----------	----------
Line Item Event	00030002h	Length = 5 bytes
		Byte 1: State (1 = Item Selected, other values RFU)
		Byte 2 - n: Caption of the selected item

Keypad Event	00030003h	Length - 3 bytes
		Byte 1: State (1 = key pressed, 2 = key released, other values RFU)
		Byte 2 - 3: Key pressed and Key release
		0030h - KEYPAD_KEY_0
		0031h - KEYPAD_KEY_1
		0032h - KEYPAD_KEY_2
		0033h - KEYPAD_KEY_3
		0034h - KEYPAD_KEY_4
		0035h - KEYPAD_KEY_5
		0036h - KEYPAD_KEY_6
		0037h - KEYPAD_KEY_7
		0038h - KEYPAD_KEY_8
		0039h - KEYPAD_KEY_9
		Byte 2 - 3: Only Key pressed
		000Dh - KEYPAD_KEY_ENTER
		0008h - KEYPAD_KEY_CLEAR
		001Bh - KEYPAD_KEY_CANCEL
		0070h - FUNC_KEY_F1 (Vend III)
		0071h - FUNC_KEY_F2 (Vend III)
		0072h - FUNC_KEY_F3 (Vend III)
		0073h - FUNC_KEY_F4 (Vend III)
Touchscreen Event	00030004h	Length = 1 - 33 bytes
		Byte 1: State (not used)
		Byte 2 - 33: Image name (zero terminated)
Slideshow Event	00030005h	Length = 1 byte
		Byte 1: State (not used)
Transaction Event	00030006h	Length = 9 bytes
		Byte 1: State (not used)
		Byte 2 - 5: Card type (0 = unknown)
		Byte 6 - 9: Status - A four byte, big-endian field
		Byte 9 is used to store the 1-byte status code
		00 - SUCCESS
		08 - TIMEOUT
		0A - FAILED
		This is not related to the extended status codes

----------—	----------—	---------- ----------—
Radio Button Event	00030007h	Length = 73 bytes
		Byte 1: State (1 = Change ins selected button, other values RFU)
		Byte 2 - 33: Null terminated group name
		Byte 34 - 65: Radio button caption

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.2.4.235 int lcd_getInputEvent_Len (IN int *timeout*, OUT int * *dataReceived*, OUT BYTE * *eventType*, IN_OUT int * *eventTypeLen*, OUT BYTE * *graphicsID*, IN_OUT int * *graphicsIDLen*, OUT BYTE * *eventData*, IN_OUT int * *eventDataLen*)

Requests input from the reader.

Parameters

timeout	Timeout amount in seconds 0 - No timeout
dataReceived	Indicates if an event occurred and data was received 0 - No data received 1 - Data received
eventType	The event type (required to be at least 4 bytes), see table below
eventTypeLen	Length of eventType
graphicsID	The graphicID of the event (required to be at least 4 bytes)
graphicsIDLen	length of graphicID
eventData	The event data, see table below (required to be at least 73 bytes)

Event Type	Value (4 bytes)	Event Specific Data
Button Event	00030000h	Length = Variable
		Byte 1: State (1 = Pressed, other values RFU)
		Byte 2 - n: Null terminated caption
----------—	----------—	---------- ----------—
Checkbox Event	00030001h	Length = 1 byte
		Byte 1: State (1 = Checked, 0 = Unchecked)
----------—	----------—	---------- ----------—
Line Item Event	00030002h	Length = 5 bytes
		Byte 1: State (1 = Item Selected, other values RFU)
		Byte 2 - n: Caption of the selected item
----------—	----------—	---------- ----------—
Keypad Event	00030003h	Length - 3 bytes
		Byte 1: State (1 = key pressed, 2 = key released, other values RFU)
		Byte 2 - 3: Key pressed and Key release

		0030h - KEYPAD_KEY_0
		0031h - KEYPAD_KEY_1
		0032h - KEYPAD_KEY_2
		0033h - KEYPAD_KEY_3
		0034h - KEYPAD_KEY_4
		0035h - KEYPAD_KEY_5
		0036h - KEYPAD_KEY_6
		0037h - KEYPAD_KEY_7
		0038h - KEYPAD_KEY_8
		0039h - KEYPAD_KEY_9
		Byte 2 - 3: Only Key pressed
		000Dh - KEYPAD_KEY_ENTER
		0008h - KEYPAD_KEY_CLEAR
		001Bh - KEYPAD_KEY_CANCEL
		0070h - FUNC_KEY_F1 (Vend III)
		0071h - FUNC_KEY_F2 (Vend III)
		0072h - FUNC_KEY_F3 (Vend III)
		0073h - FUNC_KEY_F4 (Vend III)
----------	----------	----------
Touchscreen Event	00030004h	Length = 1 - 33 bytes
		Byte 1: State (not used)
		Byte 2 - 33: Image name (zero terminated)
----------	----------	----------
Slideshow Event	00030005h	Length = 1 byte
		Byte 1: State (not used)
----------	----------	----------
Transaction Event	00030006h	Length = 9 bytes
		Byte 1: State (not used)
		Byte 2 - 5: Card type (0 = unknown)
		Byte 6 - 9: Status - A four byte, big-endian field
		Byte 9 is used to store the 1-byte status code
		00 - SUCCESS
		08 - TIMEOUT
		0A - FAILED
		This is not related to the extended status codes
----------	----------	----------
Radio Button Event	00030007h	Length = 73 bytes
		Byte 1: State (1 = Change ins selected button, other values RFU)
		Byte 2 - 33: Null terminated group name

		Byte 34 - 65: Radio button caption

Parameters

eventDataLen	Length of eventData

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.2.4.236 int lcd_getInputFieldValue (IN BYTE * *graphicId*, OUT BYTE * *retData*, IN_OUT int * *retDataLen*)

Get the keypad data that was entered into the specified Input Field.

Parameters

graphicsID	The graphicID of the input field (required to be 4 bytes)
retData	return keypad data
retDataLen	The length of retData

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.2.4.237 int lcd_getSelectedListItem (IN BYTE * *listGraphicsID*, OUT char * *itemID*)

DEPRECATED : please use lcd_getSelectedListItem_Len(IN BYTE *listGraphicsID, OUT char *itemID, IN_OUT int *itemIDLen)

Retrieves the selected item's ID.

Parameters

listGraphicsID	Existing list's graphics ID (4 byte array) that is provided during creation
itemID	The selected item's ID (Maximum: 32 characters) Need 33 bytes of memory including '\0'

12.2.4.238 int lcd_getSelectedListItem_Len (IN BYTE * *listGraphicsID*, OUT char * *itemID*, IN_OUT int * *itemIDLen*)

Retrieves the selected item's ID.

Parameters

listGraphicsID	Existing list's graphics ID (4 byte array) that is provided during creation
itemID	The selected item's ID (Maximum: 32 characters) Need 33 bytes of memory including '\0'
itemIDLen	Length of itemID

12.2.4.239 int lcd_loadScreenInfo ( )

Load Screen Info

Load all current screen information from RAM to flash

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.2.4.240 int lcd_queryObjectbyID (IN int *objectID*, OUT int * *objectNumbers*, OUT IDTScreenInfo * *screenInfo*)

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
objectNumbers	Number of the checked object
screenInfo	screen names containing the item

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.2.4.241 int lcd_queryObjectbyName (IN char * *objectName*, IN int *objectNameLen*, IN_OUT int * *objectNumbers*, OUT IDTScreenInfo * *screenInfo*)

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
objectNameLen	Length of objectName
objectNumbers	Number of the checked object
screenInfo	Array of all the screen names that contain the object

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.2.4.242 int lcd_queryScreenbyID (IN int *screenID*, OUT int * *result*, OUT int * *screenName*, IN_OUT int * *screenNameLen*)

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
screenNameLen	Length of screenName
ip	Optional: IP address to execute command on (for IP connected devices)

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.2.4.243 int lcd_queryScreenbyName (IN char * *screenName*, IN int *screenNameLen*, OUT int * *result*)

Queery Screen by Name

Check if the given screen exists or not

Parameters

screenName	Name of the checked screen
screenNameLen	Length of screenName
result	the checking result

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.2.4.244 void lcd_registerCallBk (pLCD_callBack *pLCDf*)

To register the lcd callback function to get the LCDItem. (Pass NULL to disable the callback.)

12.2.4.245 int lcd_removeItem (IN char * *screenName*, IN int *screenNameLen*, IN char * *objectName*, IN int *objectNameLen*)

Removed Item

Removes a component.

Parameters

screenName	Screen name where to remove the target from.
screenNameLen	Length of screenName.
objectName	Identifier of the component
objectNameLen	Length of objectName.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.2.4.246 int lcd_resetInitialState ( )

Reset to Initial State. This command places the reader UI into the idle state and displays the appropriate idle display.

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.2.4.247 int lcd_savePrompt (int *promptNumber*, char * *prompt*, int *promptLen*)

Save Prompt

Saves a message prompt to L100 memory.

Parameters

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

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.2.4.248 int lcd_setBackgroundImage (IN char * *file*, IN int *fileLen*, IN int *enable*)

Set Background Image You must send images to the reader??s memory and send a Start Custom Mode command to the reader before it will respond to Image commands. Image files must be in .bmp or .png format.

Parameters

file	Complete path and file name of the file you want to use. Example "file.png" will put in root directory, while "ss/file.png" will put in ss directory (which must exist)
fileLen	Length of files
enable	TRUE = Use Background Image, FALSE = Use Background Color

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.2.4.249 int lcd_setBacklight (IN BYTE *backlightVal*)

Set Backlight

Set backlight percentage. If the percent $>$ 100, it will be rejected. If 0 $<$ percent $<$ 10, backlight percent will be set to 10. If percent == 0, backlight will be turned off

Parameters

backlightVal	Backlight percent value to be sat

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.2.4.250 int lcd_setDisplayImage (IN char * *file,* IN int *fileLen,* IN int *posX,* IN int *posY,* IN int *posMode,* IN int *touchEnable,* IN int *clearScreen*)

Set Display Image You must send images to the reader's memory and send a Start Custom Mode command to the reader before it will respond to Image commands. Image files must be in .bmp or .png format.

Parameters

file	Complete path and file name of the file you want to use. Example "file.png" will put in root directory, while "ss/file.png" will put in ss directory (which must exist)
fileLen	Length of files
posX	X coordinate in pixels, Range 0-271
posY	Y coordinate in pixels, Range 0-479
posMode	Position Mode • 0 = Center on Line Y • 1 = Display at (X, Y) • 2 - Center on screen

touchEnable	TRUE = Image is touch sensitive
clearScreen	TRUE = Clear screen

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.2.4.251 int lcd_setForeBackColor (IN BYTE * *foreRGB,* IN int *foreRGBLen,* IN BYTE * *backRGB,* IN int *backRGBLen*)

Set Foreground and Background Color This command sets the foreground and background colors of the LCD.

Parameters

foreRGB	Foreground RGB. 000000 = black, FF0000 = red, 00FF00 = green, 0000FF = blue, FFFFFF = white
Length	of foreRGB. Must be 3.
backRGB	Background RGB. 000000 = black, FF0000 = red, 00FF00 = green, 0000FF = blue, FFFFFF = white
Length	of backRGB. Must be 3.

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.2.4.252 int lcd_showScreen (IN char * *screenName,* IN int *screenNameLen*)

Show Screen

Displays and makes active a previously created screen.

Parameters

screenName	Screen to display. The screen name is defined with lcd_createScreen.
screenNameLen	Length of screenName.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.2.4.253 int lcd_startSlideShow (IN char * *files,* IN int *filesLen,* IN int *posX,* IN int *posY,* IN int *posMode,* IN int *touchEnable,* IN int *recursion,* IN int *touchTerminate,* IN int *delay,* IN int *loops,* IN int *clearScreen*)

Start slide show You must send images to the reader's memory and send a Start Custom Mode command to the reader before it will respond to this commands. Image files must be in .bmp or .png format.

Parameters

files	Complete paths and file names of the files you want to use, separated by commas. If a directory is specified, all files in the dirctory are displayed
filesLen	Length of files
posX	X coordinate in pixels, Range 0-271
posY	Y coordinate in pixels, Range 0-479

posMode	Position Mode • 0 = Center on Line Y • 1 = Display at (X, Y) • 2 - Center on screen
touchEnable	TRUE = Image is touch sensitive
recursion	TRUE = Recursively follow directorys in list
touchTerminate	TRUE = Terminate slideshow on touch (if touch enabled)
delay	Number of seconds between image displays
loops	Number of display loops. A zero indicates continuous display
clearScreen	TRUE = Clear screen

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.2.4.254 int lcd_storeScreenInfo ( )

Store Screen Info

Store all current screen information from RAM to flash

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.2.4.255 int lcd_updateColor (IN char * *screenName*, IN int *screenNameLen*, IN char * *objectName*, IN int *objectNameLen*, IN BYTE * *color*, IN int *colorLen*)

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
screenNameLen	Length of screenName.
objecName	Identifier of the component
objectNameLen	Length of objectName.
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
colorLen	Length of color.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.2.4.256 int lcd_updateLabel (IN char * *screenName,* IN int *screenNameLen,* IN char * *objectName,* IN int *objectNameLen,* IN char * *label,* IN int *labelLen*)

Update Label

Updates the component label.

Parameters

screenName	Screen name that will be the target of update label
screenNameLen	Length of screenName.
objectName	Identifier of the component
objectNameLen	Length of objectName.
label	Label to show on the component
labelLen	Length of label.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.2.4.257 int lcd_updatePosition (IN char * *screenName,* IN int *screenNameLen,* IN char * *objectName,* IN int *objectNameLen,* IN BYTE *alignment,* IN int *new_xCord,* IN int *new_yCord*)

Update Position

Updates the component position.

Parameters

screenName	Screen Name that will be the target of update position
screenNameLen	Length of screenName.
objectName	Identifier of the component
objectNameLen	Length of objectName.
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

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.2.4.258 int loyalty_cancelTransaction ( )

Cancel Loyalty Transaction Only for VP6800

Cancels the currently executing transaction.

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.2.4.259 int loyalty_cancelTransactionSilent (int *enable*)

Cancel Loyalty Transaction Silent Only for VP6800

Cancel transaction with or without showing the LCD message

Parameters

enable	0: With LCD message 1: Without LCD message

Returns

success or error code. Values can be parsed with device_getIDGStatusCodeString

12.2.4.260 void loyalty_registerCallBk (pEMV_callBack *pEMVf*)

To register the loyalty callback function to get the EMV processing response. (Pass NULL to disable the callback.) Only for VP6800

12.2.4.261 int loyalty_startTransaction (IN double *amount*, IN double *amtOther*, IN int *type*, IN const int *_timeout*, IN BYTE * *tags*, IN int *tagsLen*, IN const int *cardType*, IN const int *iccReadType*)

Start Loyalty Transaction Request Only for VP6800

Authorizes the transaction for an MSR/ICC card

The tags will be returned in the callback routine.

Parameters

amount	Transaction amount value (tag value 9F02) • SEE IMPORTANT NOTE BELOW

amtOther	Other amount value, if any (tag value 9F03) • SEE IMPORTANT NOTE BELOW
type	Transaction type (tag value 9C).
timeout	Timeout value in seconds.
tags	Any other tags to be included in the request. Passed as TLV. 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.
tagsLen	The length of tags data buffer.

>>>>>IMPORTANT: parameters for amount and amtOther MUST BE PASSED AS A DOUBLE VALUE WITH DECIMAL POINT. Example, do not pass 1, but instead pass 1.0 or 1.00. Otherwise, results will be unpredictable

Parameters

cardType	Set available card to accept. 0 = MSR only. 1 = MSR and ICC.
iccReadType	In case of ICC reading, this is how to behave. 0 = Same as device_startTransaction 1 = When reading ICC, read DF4F(JIS2EquivalentData) in ReadRecord.

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString() Note: if auto poll is on, it will returm the error IDG_P2_STATUS_CODE_COMMAND_NOT_ALLOWED

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 device_startTransaction. Tag FFEE06 must contain tag 9F26 and 9F22, and can optionanally contain tags 9F2B and DF01. 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

12.2.4.262 int msr_cancelMSRSwipe ( )

Disable MSR Swipe Cancels MSR swipe request.

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.2.4.263 int msr_captureMode (int *isBufferMode*, int *withNotification*)

Set MSR Capture Mode.

For Non-SRED Augusta Only

Switch between Auto mode and Buffer mode. Auto mode only available on KB interface

Parameters

isBufferMode	• 1: Enter into Buffer mode. • 0: Enter into Auto mode. KB mode only. Swipes automatically captured and sent to keyboard buffer
withNotification	• 1: with notification when swiped MSR data. • 0: without notification when swiped MSR data.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.2.4.264 int msr_clearMSRData ( )

Clear MSR Data Clears the MSR Data buffer

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.2.4.265 int msr_disable ( )

Disable MSR Disable MSR functions.

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.2.4.266 int msr_flushTrackData ( )

Flush Track Data Clears any track data being retained in memory by future PIN Block request.

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.2.4.267 int msr_getClearPANID (BYTE * *value*)

Get Clear PAN ID.

Returns the number of digits that begin the PAN that will be in the clear

Parameters

value	4901 <Setting value>="">. setting Value: Number of digits in clear. Values are char '0' - '6'

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.2.4.268 int msr_getExpirationMask (BYTE * *value*)

Get MSR expiration date mask.

Parameters

value	5001 <Setting value>="">. setting Value: '0' = masked, '1' = not-masked

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.2.4.269 int msr_getFunctionStatus (int * *enable*, int * *isBufferMode*, int * *withNotification*)

Get MSR Function Status.

Gets the MSR function status

Parameters

enable	1 = MSR enabled, 0 = MSR disabled
isBufferMode	1 = buffer mode, 0 = auto mode

withNotification	1 = with notification, 0 = without notification

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.2.4.270 int msr_getKeyFormatForICCDUKPT (OUT BYTE * *format*)

Get Key Format For DUKPT

Specifies how data will be encrypted with Data Key or PIN key (if DUKPT key loaded). This applies to both MSR and ICC

Parameters

format	Response returned from method: • 'TDES': Encrypted card data with TDES if DUKPT Key had been loaded. (default) • 'AES': Encrypted card data with AES if DUKPT Key had been loaded. • 'NONE': No Encryption.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.2.4.271 int msr_getKeyTypeForICCDUKPT (OUT BYTE * *type*)

Get Key Type for DUKPT

Specifies the key type used for ICC DUKPT encryption. This applies to both MSR and ICC.

Parameters

type	Response returned from method: • 'DATA': Encrypted card data with Data Key DUKPT Key had been loaded. (default) • 'PIN': Encrypted card data with PIN Key if DUKPT Key had been loaded.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.2.4.272 int msr_getMSRData (OUT BYTE * *reData,* IN_OUT int * *reLen*)

Get MSR Data Reads the MSR Data buffer

Parameters

reData	Card data

reLen	Card data length

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.2.4.273 int msr_getSwipeForcedEncryptionOption (BYTE * *option*)

Get MSR Swipe Forced Encryption Option.

Parameters

option	8401 <Setting value>="">. Setting Value Byte using lower four bits as flags. 0 = Force Encryption Off, 1 = Force Encryption On bit0 = Track 1 bit1 = Track 2 bit2 = Track 3 bit4 = Track 3 Card Option 0

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.2.4.274 int msr_getSwipeMaskOption (BYTE * *option*)

Get MSR Swipe Mask Option.

Gets the swipe mask/clear data sending option

Parameters

option	8601 <Setting value>="">. Setting Value Byte using lower three bits as flags. 0 = Mask Option Off, 1 = Mask Option On bit0 = Track 1 bit1 = Track 2 bit2 = Track 3 Example: Response 0x03 = Track1/Track2 Masked Option ON, Track3 Masked Option Off

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.2.4.275 void msr_registerCallBk (**pMSR_callBack** *pMSRf*)

To register the msr callback function to get the MSR card data. (Pass NULL to disable the callback.)

12.2.4.276 void msr_registerCallBkp (**pMSR_callBackp** *pMSRf*)

To register the msr callback function to get the MSR card data pointer. (Pass NULL to disable the callback.)

12.2.4.277 int msr_setClearPANID (BYTE *val*)

Set Clear PAN ID.

Parameters

val	Set Clear PAN ID to value: Number of digits to show in clear. Range 0-6.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.2.4.278 int msr_setExpirationMask (int *mask*)

Set Expiration Masking

Sets the flag to mask the expiration date

Parameters

mask	TRUE = mask expiration

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.2.4.279 int msr_setKeyFormatForICCDUKPT (IN BYTE *format*)

Set Key Format for DUKPT

Sets how data will be encrypted, with either TDES or AES (if DUKPT key loaded) This applies to both MSR and ICC

Parameters

format	encryption Encryption Type • 00: Encrypt with TDES • 01: Encrypt with AES

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.2.4.280 int msr_setKeyTypeForICCDUKPT (IN BYTE *type*)

Set Key Type for DUKPT Key

Sets which key the data will be encrypted with, with either Data Key or PIN key (if DUKPT key loaded) This applies to both MSR and ICC

Parameters

type	Encryption Type • 00: Encrypt with Data Key • 01: Encrypt with PIN Key

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.2.4.281 int msr_setSwipeForcedEncryptionOption (int *track1,* int *track2,* int *track3,* int *track3card0*)

Set MSR Swipe Forced Encryption Option.

Parameters

tarck1	Set track1 encryption to true or false.
tarck2	Set track2 encryption to true or false.
tarck3	Set track3 encryption to true or false.
tarck3card0	Set track3 card0 encryption to true or false.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.2.4.282 int msr_setSwipeMaskOption (int *track1*, int *track2*, int *track3*)

Set MSR Swipe Mask Option.

Sets the swipe mask/clear data sending option

Parameters

tarck1	Set track1 mask to true or false.
tarck2	Set track2 mask to true or false.
tarck3	Set track3 mask to true or false.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.2.4.283 int msr_startMSRSwipe (IN int *_timeout*)

Start MSR Swipe Enables MSR, waiting for swipe to occur. Allows track selection. Returns IDTMSRData instance to MSR_callBack()

Parameters

timeout	Swipe Timeout Value

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString() Note: if auto poll mode is on, it will return command not allowed error

12.2.4.284 void parseMSRData (IN BYTE * *resData*, IN int *resLen*, IN_OUT IDTMSRData * *cardData*)

Parser the MSR data from the buffer into IDTMSTData structure

Parameters

resData	MSR card data buffer
resLen	the length of resData
cardData	the parser result with IDTMSTData structure

12.2.4.285 void parsePINBlockData (IN BYTE * *resData*, IN int *resLen*, IN_OUT IDTPINData * *cardData*)

Parse the PIN block data from the buffer into IDTPINData structure

Parameters

resData	PIN card data buffer
resLen	the length of resData
cardData	the parser result with IDTPINData structure

12.2.4.286 void parsePINData (IN BYTE * *resData*, IN int *resLen*, IN_OUT IDTPINData * *cardData*)

Parse the PIN data from the buffer into IDTPINData structure

Parameters

resData	PIN card data buffer
resLen	the length of resData
cardData	the parser result with IDTPINData structure

12.2.4.287 int pin_cancelPINEntry ( )

Cancel PIN Entry

Cancels PIN entry request

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.2.4.288 int pin_capturePin (IN int *timeout*, IN int *type*, IN char * *PAN*, IN int *PANLen*, IN int *minPIN*, IN int *maxPIN*, IN char * *message*, IN int *messageLen*)

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)
PANLen	Length of PAN
minPIN	Minimum PIN Length
maxPIN	Maximum PIN Length
message	LCD Message
messageLen	Length of message

Returns

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

12.2.4.289 int pin_capturePinExt (IN int *type*, IN char * *PAN*, IN int *PANLen*, IN int *minPIN*, IN int *maxPIN*, IN char * *message*, IN int *messageLen*, IN char * *verify*, IN int *verifyLen*)

- Capture PIN Ext

Parameters

type	PAN and Key Type • 00h: MKSK to encrypt PIN, Internal PAN (from MSR or Manual PAN Entry or Contactless EMV Transaction) • 01h: DUKPT to encrypt PIN, Internal PAN (from MSR or Manual PAN Entry or Contactless EMV Transaction) • 10h: MKSK to encrypt PIN, External Plaintext PAN • 11h: DUKPT to encrypt PIN, External Plaintext PAN • 20h: MKSK to encrypt PIN, External Ciphertext PAN (for PIN pad only) • 21h: DUKPT to encrypt PIN, External Ciphertext PAN (for PIN pad only) • 80h: MKSK to encrypt PIN, Internal PAN, Verify PIN (from MSR or Manual PAN Entry or Contactless EMV Transaction) • 81h: DUKPT to encrypt PIN, Internal PAN, Verify PIN (from MSR or Manual PAN Entry or Contactless EMV Transaction) • 90h: MKSK to encrypt PIN, External Plaintext PAN, Verify PIN • 91h: DUKPT to encrypt PIN, External Plaintext PAN, Verify PIN

PAN	Personal Account Number (if internal, value is 0)
PANLen	Length of PAN
minPIN	Minimum PIN Length
maxPIN	Maximum PIN Length
message	LCD Message Up to 2 lines of text, each line 1-16, separated by 0x00
messageLen	Length of 1st scenario LCD message, valid in 00h∼21h (0∼33).If no LCD message input, length is 00h, and display default msg “PLEASE ENTER PIN”
verify	LCD Message Up to 2 lines of text, each line 1-16, separated by 0x00
verifyLen	Length of 2nd Scenario LCD message.valid in 00h∼21h (0∼33).This field is present only when PAN and Key Type has Verify PIN.If no LCD message input, length is 00h, and display default msg “ ENTER PIN AGAIN”

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.2.4.290 int pin_getEncryptedOnlinePIN (IN int *keyType,* IN int *timeout*)

Get Encrypted DUKPT PIN

Requests PIN Entry for online authorization. PIN block and KSN returned in callback function DeviceState.-TransactionData with cardData.pin_pinblock. A swipe must be captured first before this function can execute

Parameters

keyType	PIN block key type. Valid values 0, 3 for TDES, 4 for AES
timeout	PIN entry timout

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.2.4.291 int pin_getEncryptedPIN (int *keyType,* char ∗ *PAN,* int *PANLen,* char ∗ *message,* int *messageLen,* 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
PANLen	length of PAN
message	Message to display
messageLen	length of message
timeout	PIN entry timeout

Returns

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

12.2.4.292 int pin_getFunctionKey (int *timeout*)

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

@param timeout Timeout, in seconds
```

Returns

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

12.2.4.293 int pin_getPAN (IN int *getCSC,* IN int *timeout*)

Get PAN

Requests PAN Entry on pinpad

Parameters

getCSC	Include Customer Service Code (also known as CVV, CVC)
timeout	PAN entry timout

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.2.4.294 int pin_getPanEntry (IN int *csc,* IN int *expDate,* IN int *ADR,* IN int *ZIP,* IN int *mod10CK,* IN int *timeout,* IN int *encPANOnly*)

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

csc	Request CSS
expDate	Request Expiration Date
ADR	Request Address
ZIP	Request Zip
mod10CK	Validate entered PAN passes MOD-10 checking before accepting
timeout	Number of seconds that the reader waits for the data entry session to complete, stored as a big-endian number. 0 = no timeout
encPANOnly	Output only encrypted PAN

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.2.4.295 int pin_getPIN (IN int *mode*, IN int *PANSource*, IN char * *iccPAN*, IN int *iccPANLen*, IN int *startTimeout*, IN int *entryTimeout*, IN char * *language*, IN int *languageLen*)

Get Encrypted PIN

Requests PIN Entry

Parameters

mode	• 0x00- Cancel: Cancels PIN entry = also can execute pin_cancelPINEntry(). All other parameters for this method will be ignored • 0x01- Online PIN DUKPT • 0x02- Online PIN MKSK • 0x03- Offline PIN (No need to define PAN Source or ICC PAN)
PANSource	• 0x00- ICC: PAN Captured from ICC and must be provided in iccPAN parameter • 0x01- MSR: PAN Captured from MSR swipe and will be inserted by Spectrum Pro. No need to provide iccPAN parameter.
iccPAN	PAN captured from ICC. When PAN captured from MSR, this parameter will be ignored
iccPANLen	the length of iccPAN
startTimeout	The amount of time allowed to start PIN entry before timeout
entryTimeout	The amount of time to enter the PIN after first digit selected before timeout
language	Valid values "EN" for English, "ES" for Spanish, "ZH" for Chinese, "FR" for French
languageLen	the length of language

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.2.4.296 int pin_inputFromPrompt (BYTE *mask*, BYTE *preClearText*, BYTE *postClearText*, int *minLen*, int *maxLen*, char * *lang*, BYTE *promptID*, char * *defaultResponse*, int *defaultResponseLen*, int *timeout*)

Get PIN Input from Prompt Results returned to PIN Callback. If successful function key capture, data is returned as IDTPINData.keyString.

Parameters

mask	0 = no masking, 1 = Display "∗" for numeric key according to Pre-Cleartext and Post-Cleartext display
preClearText	Range 0-6 Characters to mask at start of text if masking is on;
postClearText	Range 0-6 Characters to mask at end of text if masking is on;
minLen	Minimum number of digits required to input
maxLen	Maximum number of digits allowed to be input
lang	Valid values; "EN" is English display message "JP" is Japanese display message "ES" is Spanish display message "FR" is French display message "ZH" is Chinese display message "PT" is Portuguese display message
promptID	Valid values: 0x00: Enter Phone Number 0x01: Enter IP Address 0x02: Enter Subnet Mask 0x03: Enter Default Gateway 0x04: Enter Odometer Reading/Mileage 0x05: Enter Employee ID 0x06: Enter Password for Default Factory Setting 0x07: Enter Email Address (Full keyboard)
defaultResponse	Default String on input field
defaultResponseLen	Length of defaultResponse
timeout	Timeout, in seconds

Returns

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

12.2.4.297 int pin_promptCreditDebit (IN char ∗ *currencySymbol,* IN int *currencySymbolLen,* IN char ∗ *displayAmount,* IN int *displayAmountLen,* IN int *timeout,* OUT BYTE ∗ *retData,* IN_OUT int ∗ *retDataLen*)

Prompt for Credit or Debit

Requests prompt for Credit or Debit. Response returned in callback function as DeviceState.MenuItem with data MENU_SELECTION_CREDIT = 0, MENU_SELECTION_DEBIT = 1

Parameters

currencySymbol	Allowed values are $ (0x24), ???(0xA5), ???(0xA3), ???(0xA4), or NULL
currencySymbolLen	length of currencySymbol
displayAmount	Amount to display (can be NULL)
displayAmountLen	length of displayAmount
timeout	Menu entry timout. Valid values 2-20 seconds
retData	If successful, the return code is zero and the data is 1 byte (0: Credit 1: Debit). If the return code is not zero, it may be a four-byte Extended Status Code
currencySymbolLen	length of currencySymbol

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.2.4.298 int pin_promptForAmount (IN int *timeout,* IN int *minLen,* IN int *maxLen,* IN char ∗ *message,* IN int *messageLen,* BYTE ∗ *signature,* IN int *signatureLen*)

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
messageLen	Length of message
signature	Display message signed by Numeric Private Key using RSAPSS algorithm: 1. Calculate 32 bytes Hash for ?斤??Display Flag><Key max="" length>=""><Key Min Length><Plaintext display="" message>=""><?斤?? 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
signatureLen	Length of signature

Returns

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

12.2.4.299 int pin_promptForAmountInput (int *messageID,* int *languageID,* int *minLen,* int *maxLen,* int *timeout*)

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
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	N?MERO PO	NUMERO PO	No COMMANDE
8	DRIVER ID	LICEN?A	LICENCIA	ID CONDUCTEUR
9	ODOMETER	ODOMETER	ODOMETRO	ODOMETRE
10	ID NUMBER	N?MERO 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 N?MERO	NUMERO EMP	No TRAVAIL
14	WORK ORDER	TRABALHO ORDEM	ORDEN TRABAJO	FICHE TRAVAIL
15	VEHICLE ID	ID VE?CULO	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	N?MERO PATH	RUTA NUMERO	No ROUTE
23	ENTER USER ID	ENTER USER ID	INGRESE ID USUARIO	ID UTILISATEUR
24	FLEET NUMBER	N?MERO DE FROTA	FLOTA NUMERO	No PARC AUTO
25	ENTER PRODUCT	ADICIONAR PRODUTO	INGRESE PRODUCTO	ENTREZ PRODUIT
26	DRIVER NUMBER	N?MERO DRIVER	CONDUCTOR NUMERO	No CONDUCTEUR
27	ENTER LICENSE	ENTER LICEN?A	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 VE?CULO	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	N?MERO DO VE?CULO	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 USU?RIO	INGRESE INFO USUARIO	INFO UTILISATEUR
39	ENTER CUST CODE	ENTER CODE. CLIENTE	INGRESE COD. CLIENTE	ENTR CODE CLIENT
40	ENTER EMPLOYEE	ENTER FUNCION?RIO	INGRESE EMPLEADO	ENTREZ EMPLOYE
41	ENTER ID NUMBER	ENTER N?MERO 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	N?MERO 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 VE?CULO	REINGRESE VEHICULO	RE-ENTR VEHICULE
48	ENTER VEHICLE ID	ENTER VE?CULO 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 REFER?NCIA	INGRESE REFERENCIA	ENTREZ REFERENCE
53	ENTER AUTH NUMBR	ENTER N?MERO 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	QUILOMETRAGE-M	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 #	VE?CULO #	VEHICULO #	# VEHICULE
68	ENTER VEHICLE #	ENTER VE?CULO #	INGRESE VEHICULO #	ENT # VEHICULE
69	JOB #	TRABALHO #	TRABAJO #	# TRAVAIL
70	ENTER JOB #	ENTER JOB #	INGRESE TRABAJO #	ENTREZ # TRAVAIL
71	DEPT NUMBER	N?MERO DEPT	NUMERO DEPTO	No DEPARTEMENT

72	DEPARTMENT #	DEPARTAMENTO #	DEPARTAMENTO #	DEPARTEMENT
73	ENTER DEPT #	ENTER DEPT #	INGRESE DEPTO #	ENT# DEPARTEMENT
74	LICENSE NUMBER	N?MERO DE LICEN?A	NUMERO LICENCIA	No PERMIS
75	LICENSE #	LICEN?A #	LICENCIA #	# PERMIS
76	ENTER LICENSE #	ENTER LICEN?A #	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 N?MERO DE CART?O	INGRESE NUM TARJETA	ENTREZ NO CARTE
86	EXP DATE(YYMM)	VALIDADE VAL (AAMM)	FECHA EXP (AAMM)	DATE EXPIR(AAMM)
87	PHONE NUMBER	TELEFONE	NUMERO TELEFONO	NO TEL
88	CVV START DATE	CVV DATA DE IN?CIO	CVV FECHA INICIO	CVV DATE DE DEBUT
89	ISSUE NUMBER	N?MERO DE EMISS?O	NUMERO DE EMISION	NO DEMISSION
90	START DATE (MMYY)	DATA DE IN?CIO (AAMM)	FECHA INICIO (AAMM)	DATE DE DEBUT-AAMM

```
@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
@param timeout Timout value, in seconds


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

12.2.4.300 int pin_promptForKeyInput (int *messageID,* int *languageID,* int *maskInput,* int *minLen,* int *maxLen,* int *timeout*)

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	N?MERO PO	NUMERO PO	No COMMANDE
8	DRIVER ID	LICEN?A	LICENCIA	ID CONDUCTEUR
9	ODOMETER	ODOMETER	ODOMETRO	ODOMETRE
10	ID NUMBER	N?MERO 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 N?MERO	NUMERO EMP	No TRAVAIL
14	WORK ORDER	TRABALHO ORDEM	ORDEN TRABAJO	FICHE TRAVAIL
15	VEHICLE ID	ID VE?CULO	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	N?MERO PATH	RUTA NUMERO	No ROUTE
23	ENTER USER ID	ENTER USER ID	INGRESE ID USUARIO	ID UTILISATEUR
24	FLEET NUMBER	N?MERO DE FROTA	FLOTA NUMERO	No PARC AUTO
25	ENTER PRODUCT	ADICIONAR PRODUTO	INGRESE PRODUCTO	ENTREZ PRODUIT
26	DRIVER NUMBER	N?MERO DRIVER	CONDUCTOR NUMERO	No CONDUCTEUR
27	ENTER LICENSE	ENTER LICEN?A	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 VE?CULO	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	N?MERO DO VE?CULO	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 USU?RIO	INGRESE INFO USUARIO	INFO UTILISATEUR
39	ENTER CUST CODE	ENTER CODE. CLIENTE	INGRESE COD. CLIENTE	ENTR CODE CLIENT
40	ENTER EMPLOYEE	ENTER FUNCION?RIO	INGRESE EMPLEADO	ENTREZ EMPLOYE
41	ENTER ID NUMBER	ENTER N?MERO 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	N?MERO 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 VE?CULO	REINGRESE VEHICULO	RE-ENTR VEHICULE
48	ENTER VEHICLE ID	ENTER VE?CULO 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 REFER?NCIA	INGRESE REFERENCIA	ENTREZ REFERENCE
53	ENTER AUTH NUMBR	ENTER N?MERO 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	QUILOMETRAGE-M	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 #	VE?CULO #	VEHICULO #	# VEHICULE
68	ENTER VEHICLE #	ENTER VE?CULO #	INGRESE VEHICULO #	ENT # VEHICULE
69	JOB #	TRABALHO #	TRABAJO #	# TRAVAIL
70	ENTER JOB #	ENTER JOB #	INGRESE TRABAJO #	ENTREZ # TRAVAIL
71	DEPT NUMBER	N?MERO DEPT	NUMERO DEPTO	No DEPARTEMENT
72	DEPARTMENT #	DEPARTAMENTO #	DEPARTAMENTO #	DEPARTEMENT
73	ENTER DEPT #	ENTER DEPT #	INGRESE DEPTO #	ENT# DEPARTEMENT
74	LICENSE NUMBER	N?MERO DE LICEN?A	NUMERO LICENCIA	No PERMIS
75	LICENSE #	LICEN?A #	LICENCIA #	# PERMIS
76	ENTER LICENSE #	ENTER LICEN?A #	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 N?MERO DE CART?O	INGRESE NUM TARJETA	ENTREZ NO CARTE
86	EXP DATE(YYMM)	VALIDADE VAL (AAMM)	FECHA EXP (AAMM)	DATE EXPIR(AAMM)
87	PHONE NUMBER	TELEFONE	NUMERO TELEFONO	NO TEL
88	CVV START DATE	CVV DATA DE IN?CIO	CVV FECHA INICIO	CVV DATE DE DEBUT
89	ISSUE NUMBER	N?MERO DE EMISS?O	NUMERO DE EMISION	NO DEMISSION
90	START DATE (MMYY)	DATA DE IN?CIO (AAMM)	FECHA INICIO (AAMM)	DATE DE DEBUT-AAMM

```
@param messageID  Message (1-90)
@param languageID 0=English Prompt, 1=Portuguese Prompt, 2=Spanish Prompt, 3=French Prompt
@param maskInput  1 = entry is masked with '*', 0 = 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
@param timeout Timout value, in seconds


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

12.2.4.301 int pin_promptForNumericKey (IN int *timeout,* IN int *maskInput,* IN int *minLen,* IN int *maxLen,* IN char * *message,* IN int *messageLen,* BYTE * *signature,* IN int *signatureLen*)

Capture Numeric Input

Parameters

timeout	Timeout, in seconds. Value of 0 will use system timeout, if any
maskInput	• 0 = Display numeric for numeric key on LCD • 1 = Display ?∗? for numeric key on LCD
minLen	Minimum input Length
maxLen	Maximum input Length
message	Plaintext Display Message. - 16 chars max
messageLen	Length of 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
signatureLen	Length of signature

Returns

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

12.2.4.302 int pin_promptForNumericKeyWithSwipe (IN int *timeout,* IN BYTE *function,* IN int *minLen,* IN int *maxLen,* IN char ∗ *line1,* IN int *line1Len,* IN char ∗ *line2,* IN int *line2Len,* BYTE ∗ *signature,* IN int *signatureLen*)

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
line1	Line 1 of LCD Message - 16 chars max
line1Len	Length of line1
line2	Line 2 of LCD Message - 16 chars max
line2Len	Length of line2
signature	Display message signed by Numeric Private Key using RSAPSS algorithm: 1. Calculate 32 bytes Hash for ?斤??Display Flag><Key max="" length>=""><Key Min Length><Plaintext display="" message>=">?斤?? 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
signatureLen	Length of signature

Returns

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

12.2.4.303 void pin_registerCallBk (pPIN_callBack *pPINf*)

To register the pin callback function to get the PINPad data. (Pass NULL to disable the callback.)

12.2.4.304 int pin_sendBeep (int *frequency,* int *duration*)

Send Beep

Executes a beep request.

Parameters

frequency	Frequency, range 200-20000Hz Not used for NEO 2 devices
duration	Duration, range 16-65535ms Not used for NEO 2 devices

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.2.4.305 int pin_setKeyValues (int *mode*)

Set Key Values

Set return key values on or off

Parameters

mode	On: 1, Off: 0

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.2.4.306 void registerHotplugCallBk (pMessageHotplug *pMsgHotplug*)

To register the USB HID hot-plug callback function which implemented in the application to monitor the hotplug message from the SDK.

12.2.4.307 void registerLogCallBk (pSendDataLog *pFSend,* pReadDataLog *pFRead*)

To register the log callback function which implemented in the application to monitor sending/reading data between application and reader.

12.2.4.308 int rs232_device_init (int *deviceType,* int *port_number,* int *brate*)

Initial the device by RS232

It will try to connect to the device with provided deviceType, port_number, and brate.

Parameters

deviceType	Device to connect to
port_number	Port number of the device

Port nr. | **Linux** | **Windows**

| 0 | ttyS0 | COM1 | | 1 | ttyS1 | COM2 | | 2 | ttyS2 | COM3 | | 3 | ttyS3 | COM4 | | 4 | ttyS4 | COM5 | | 5 | ttyS5 | COM6 | | 6 | ttyS6 | COM7 | | 7 | ttyS7 | COM8 | | 8 | ttyS8 | COM9 | | 9 | ttyS9 | COM10 | | 10 | ttyS10 | COM11 | | 11 | ttyS11 | COM12 | | 12 | ttyS12 | COM13 | | 13 | ttyS13 | COM14 | | 14 | ttyS14 | COM15 | | 15 | ttyS15 | COM16 | | 16 | ttyUSB0 | n.a. | | 17 | ttyUSB1 | n.a. | | 18 | ttyUSB2 | n.a. | | 19 | ttyUSB3 | n.a. | | 20 | ttyUSB4 | n.a. | | 21 | ttyUSB5 | n.a. | | 22 | ttyAMA0 | n.a. | | 23 | ttyAMA1 | n.a. | | 24 | ttyACM0 | n.a. | | 25 | ttyACM1 | n.a. | | 26 | rfcomm0 | n.a. | | 27 | rfcomm1 | n.a. | | 28 | ircomm0 | n.a. | | 29 | ircomm1 | n.a. | | 30 | cuau0 | n.a. | | 31 | cuau1 | n.a. | | 32 | cuau2 | n.a. | | 33 | cuau3 | n.a. | | 34 | cuaU0 | n.a. | | 35 | cuaU1 | n.a. | | 36 | cuaU2 | n.a. | | 37 | cuaU3 | n.a. |

Parameters

brate	Bitrate of the device

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.2.4.309 char* SDK_Version ( )

To Get SDK version

Returns

return the SDK version string

12.2.4.310 int setAbsoluteLibraryPath (const char * *absoluteLibraryPath*)

Set the path to use when searching for ID TECH's libraries. If this is not set, the libraries will be searched for with the system's default procedures.

Parameters

absoluteLibrary-Path	The absolute path to ID TECH's libraries.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.2.4.311 int ws_deleteSSLCert (IN char * *name*, IN int *nameLen*)

Delete SSL Certificate Deletes a SSL Certificate by name

Parameters

name	Name of certificate to delete
nameLen	Certificate Name Length

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.2.4.312 int ws_getCertChainType (OUT int * *type*)

Get Certificate Chain Type Returns indicator for using test/production certificate chain

Parameters

type	0 = test certificate chain, 1 = production certificate chain

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.2.4.313 int ws_loadSSLCert (IN char * *name*, IN int *nameLen*, IN char * *dataDER*, IN int *dataDERLen*)

Load SSL Certificate Loads a SSL certificate

Parameters

name	Certificate Name
nameLen	Certificate Name Length
dataDER	DER encoded certificate data
dataDERLen	DER encoded certificate data length

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.2.4.314 int ws_requestCSR (OUT RequestCSR * *csr*)

Request CSR Requests 3 sets of public keys: encrypting Keys, signing/validating keys, signing/validating 3rd party apps

Parameters

csr	RequestCSR structure to return the data

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.2.4.315 int ws_revokeSSLCert (IN char * *name*, IN int *nameLen*)

Revoke SSL Certificate Revokes a SSL Certificate by name

Parameters

name	Name of certificate to revoke
nameLen	Certificate Name Length

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.2.4.316 int ws_updateRootCertificate (IN char * *name*, IN int *nameLen*, IN char * *dataDER*, IN int *dataDERLen*, IN char * *signature*, IN int *signatureLen*)

Update Root Certificate Updates the root certificate

Parameters

name	Certificate Name
nameLen	Certificate Name Length
dataDER	DER encoded certificate data
dataDERLen	DER encoded certificate data length
signature	Future Root CA signed (RSASSA PSS SHA256) by current Root CA
signature	length

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.3 Source_C/libIDT_KioskIII.h File Reference

KioskIII API.

```
#include "IDTDef.h"
```

Macros

- #define IN
- #define OUT
- #define IN_OUT

Typedefs

- typedef void(∗ pMessageHotplug)(int, int)
- typedef void(∗ pSendDataLog)(unsigned char ∗, int)
- typedef void(∗ pReadDataLog)(unsigned char ∗, int)
- typedef void(∗ pEMV_callBack)(int, int, unsigned char ∗, int, IDTTransactionData ∗, EMV_Callback ∗, int)
- typedef void(∗ pMSR_callBack)(int, IDTMSRData)
- typedef void(∗ pMSR_callBackp)(int, IDTMSRData ∗)
- typedef void(∗ pPIN_callBack)(int, IDTPINData ∗)
- typedef void(∗ pCMR_callBack)(int, IDTCMRData ∗)
- typedef void(∗ pCSFS_callBack)(BYTE status)
- typedef void(∗ ftpComm_callBack)(int, int, int)
- typedef void(∗ httpComm_callBack)(BYTE ∗, int)
- typedef void(∗ v4Comm_callBack)(BYTE, BYTE, BYTE ∗, int)

Functions

- void registerHotplugCallBk (pMessageHotplug pMsgHotplug)
- void registerLogCallBk (pSendDataLog pFSend, pReadDataLog pFRead)
- void emv_registerCallBk (pEMV_callBack pEMVf)
- void ctls_registerCallBk (pMSR_callBack pCTLSf)
- void ctls_registerCallBkp (pMSR_callBackp pCTLSf)
- void pin_registerCallBk (pPIN_callBack pPINf)
- void device_registerCameraCallBk (pCMR_callBack pCMRf)
- void device_registerCardStatusFrontSwitchCallBk (pCSFS_callBack pCSFSf)
- char ∗ SDK_Version ()
- int setAbsoluteLibraryPath (const char ∗absoluteLibraryPath)
- int device_init ()
- int rs232_device_init (int deviceType, int port_number, int brate)
- int device_setCurrentDevice (int deviceType)
- int device_close ()
- void device_getIDGStatusCodeString (IN int returnCode, OUT char ∗despcrition)
- int device_isConnected ()
- int device_isAttached (int deviceType)
- int device_getFirmwareVersion (OUT char ∗firmwareVersion)
- int device_getFirmwareVersion_Len (OUT char ∗firmwareVersion, IN_OUT int ∗firmwareVersionLen)
- int device_pingDevice ()
- int device_controlUserInterface (IN BYTE ∗values)
- int device_getCurrentDeviceType ()
- int device_SendDataCommandNEO (IN int cmd, IN int subCmd, IN BYTE ∗data, IN int dataLen, OUT BYTE ∗response, IN_OUT int ∗respLen)
- int device_enablePassThrough (int enablePassThrough)
- int device_setBurstMode (IN BYTE mode)
- int device_setPollMode (IN BYTE mode)
- int device_setMerchantRecord (int index, int enabled, char ∗merchantID, char ∗merchantURL)
- int device_getTransactionResults (IDTMSRData ∗cardData)
- int device_getMerchantRecord (IN int index, OUT BYTE ∗record)
- int device_getMerchantRecord_Len (IN int index, OUT BYTE ∗record, IN_OUT int ∗recordLen)
- int device_getSDKWaitTime ()
- void device_setSDKWaitTime (int waitTime)
- int device_getThreadStackSize ()
- void device_setThreadStackSize (int threadSize)
- int config_getSerialNumber (OUT char ∗sNumber)
- int config_getSerialNumber_Len (OUT char ∗sNumber, IN_OUT int ∗sNumberLen)

- int ctls_startTransaction (IN double amount, IN double amtOther, IN int type, IN const int _timeout, IN BYTE ∗tags, IN int tagsLen)
- int ctls_activateTransaction (IN const int _timeout, IN BYTE ∗tags, IN int tagsLen)
- int ctls_cancelTransaction ()
- int ctls_retrieveApplicationData (IN BYTE ∗AID, IN int AIDLen, OUT BYTE ∗tlv, IN_OUT int ∗tlvLen)
- int ctls_setApplicationData (IN BYTE ∗tlv, IN int tlvLen)
- int ctls_removeApplicationData (IN BYTE ∗AID, IN int AIDLen)
- int ctls_removeAllApplicationData ()
- int ctls_retrieveAIDList (OUT BYTE ∗AIDList, IN_OUT int ∗AIDListLen)
- int ctls_retrieveTerminalData (OUT BYTE ∗tlv, IN_OUT int ∗tlvLen)
- int ctls_setTerminalData (IN BYTE ∗tlv, IN int tlvLen)
- int ctls_retrieveCAPK (IN BYTE ∗capk, IN int capkLen, OUT BYTE ∗key, IN_OUT int ∗keyLen)
- int ctls_setCAPK (IN BYTE ∗capk, IN int capkLen)
- int ctls_removeCAPK (IN BYTE ∗capk, IN int capkLen)
- int ctls_removeAllCAPK ()
- int ctls_retrieveCAPKList (OUT BYTE ∗keys, IN_OUT int ∗keysLen)
- int ctls_setConfigurationGroup (IN BYTE ∗tlv, IN int tlvLen)
- int ctls_getConfigurationGroup (IN int group, OUT BYTE ∗tlv, OUT int ∗tlvLen)
- int ctls_getAllConfigurationGroups (OUT BYTE ∗tlv, IN_OUT int ∗tlvLen)
- int ctls_removeConfigurationGroup (int group)
- void parseMSRData (IN BYTE ∗resData, IN int resLen, IN_OUT IDTMSRData ∗cardData)

12.3.1 Detailed Description

KioskIII API. KioskIII Global API methods.

12.3.2 Macro Definition Documentation

12.3.2.1 #define IN

INPUT parameter.

12.3.2.2 #define IN_OUT

INPUT / OUTPUT PARAMETER.

12.3.2.3 #define OUT

OUTPUT parameter.

12.3.3 Typedef Documentation

12.3.3.1 typedef void(∗ ftpComm_callBack)(int, int, int)

Define the comm callback function to get FTP file transfer status

It should be passed as a parameter in a FTP request, Signature (int, int, int) = response code, current block, total blocks RESPONSE CODES: 100 = FILE DOWNLOAD STARTED 101 = FILE BLOCK XX OF XX RECEIVED 102 = FILE DOWNLOAD COMPLETED 103 = FILE DOWNLOAD TERMINATED PREMATURELY

12.3.3.2 typedef void(∗ httpComm_callBack)(BYTE ∗, int)

Define the comm callback function to get the async url data

It should be registered using the comm_registerHTTPCallback

12.3.3.3 typedef void(∗ pCMR_callBack)(int, IDTCMRData ∗)

Define the camera callback function to get the image data

It should be registered using the device_registerCameraCallBk,

12.3.3.4 typedef void(∗ pCSFS_callBack)(BYTE status)

Define the card status and front switch callback function to get card and front switch status

It should be registered using the device_registerCardStatusFrontSwitchCallBk,

12.3.3.5 typedef void(∗ pEMV_callBack)(int, int, unsigned char ∗, int, IDTTransactionData ∗, EMV_Callback ∗, int)

Define the EMV callback function to get the transaction message/data/result.

It should be registered using the emv_registerCallBk,

12.3.3.6 typedef void(∗ pMessageHotplug)(int, int)

Define the USB hot-plug callback function to monitor the info when plug in/out the reader.

It should be registered using the registerHotplugCallBk, The first integer parameter is device type, and the second integer parameter is either 0: Device Plugged Out or 1: Device Plugged In

12.3.3.7 typedef void(∗ pMSR_callBack)(int, IDTMSRData)

Define the MSR callback function to get the MSR card data

It should be registered using the msr_registerCallBk, this callback function is for backward compatibility

12.3.3.8 typedef void(∗ pMSR_callBackp)(int, IDTMSRData ∗)

Define the MSR callback function to get pointer to the MSR card data

It should be registered using the msr_registerCallBk, this callback function is recommended instead of pMSR_callBack

12.3.3.9 typedef void(∗ pPIN_callBack)(int, IDTPINData ∗)

Define the PINPad callback function to get the input PIN Pad data

It should be registered using the pin_registerCallBk,

12.3.3.10 typedef void(∗ pReadDataLog)(unsigned char ∗, int)

Define the read response callback function to monitor the reading response from the reader.

It should be registered using the registerLogCallBk,

12.3.3.11 typedef void(∗ pSendDataLog)(unsigned char ∗, int)

Define the send command callback function to monitor the sending command into the reader.

It should be registered using the registerLogCallBk,

12.3.3.12 typedef void(∗ v4Comm_callBack)(BYTE, BYTE, BYTE ∗, int)

Define the comm callback function to receive the V4 Protocol packets received by the device from an external source (IP/USB/RS-232) It should be registered using the comm_registerV4Callback, Data callback will contain command, sub-command, and data from V4 packet

12.3.4 Function Documentation

12.3.4.1 int config_getSerialNumber (OUT char ∗ *sNumber*)

DEPRECATED : please use config_getSerialNumber_Len(OUT char∗ sNumber, IN_OUT int ∗sNumberLen)

Polls device for Serial Number

Parameters

sNumber	Returns Serial Number; needs to have at least 64 bytes of memory

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString

12.3.4.2 int config_getSerialNumber_Len (OUT char ∗ *sNumber,* IN_OUT int ∗ *sNumberLen*)

Polls device for Serial Number

Parameters

sNumber	Returns Serial Number
sNumberLen	Length of Serial Number

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString

12.3.4.3 int ctls_activateTransaction (IN const int *_timeout,* IN BYTE ∗ *tags,* IN int *tagsLen*)

Start CTLS Transaction Request

Authorizes the CTLS 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 stream. Example, tag 9F0C with amount 0x000000000100 would be 0x9F0C06000000000100
tagsLen	The length of tags data buffer.

>>>>>IMPORTANT: parameters for amount and amtOther MUST BE PASSED AS A DOUBLE VALUE WITH DECIMAL POINT. Example, do not pass 1, but instead pass 1.0 or 1.00. Otherwise, results will be unpredictable

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString() Note: if auto poll is on, it will returm the error IDG_P2_STATUS_CODE_COMMAND_NOT_ALLOWED

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

12.3.4.4 int ctls_cancelTransaction ( )

Cancel EMV Transaction

Cancels the currently executing EMV transaction.

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.3.4.5 int ctls_getAllConfigurationGroups (OUT BYTE * *tlv*, IN_OUT int * *tlvLen*)

Retrieve All Configuration Groups

Returns all the Configuration Groups installed on the terminal for CTLS

Parameters

tlv	The TLV elements data
tlvLen	the length of tlv data buffer.

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.3.4.6 int ctls_getConfigurationGroup (IN int *group*, OUT BYTE * *tlv*, OUT int * *tlvLen*)

Get Configuration Group

Retrieves the Configuration for the specified Group.

Parameters

group	Configuration Group
tlv	return data
tlvLen	the length of tlv data buffer

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.3.4.7 void ctls_registerCallBk (pMSR_callBack *pCTLSf*)

To register the ctls callback function to get the MSR card data. (Pass NULL to disable the callback.)

12.3.4.8 void ctls_registerCallBkp (pMSR_callBackp *pCTLSf*)

To register the ctls callback function to get the MSR card data pointer. (Pass NULL to disable the callback.)

12.3.4.9 int ctls_removeAllApplicationData ( )

Remove All Application Data

Removes all the Application Data

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.3.4.10 int ctls_removeAllCAPK ( )

Remove All Certificate Authority Public Key

Removes all the CAPK

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.3.4.11 int ctls_removeApplicationData (IN BYTE * *AID,* IN int *AIDLen*)

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
AIDLen	the length of AID data buffer

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.3.4.12 int ctls_removeCAPK (IN BYTE * *capk*, IN int *capkLen*)

Remove Certificate Authority Public Key

Removes the CAPK as specified by the RID/Index

Parameters

capk	6 byte CAPK = 5 bytes RID + 1 byte INDEX
capkLen	the length of capk data buffer

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.3.4.13 int 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	Values can be parsed with device_getIDGStatusCodeString()

12.3.4.14 int ctls_retrieveAIDList (OUT BYTE * *AIDList*, IN_OUT int * *AIDListLen*)

Retrieve AID list

Returns all the AID names installed on the terminal for CTLS. .

Parameters

AIDList	array of AID name byte arrays
AIDListLen	the length of AIDList array buffer

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.3.4.15 int ctls_retrieveApplicationData (IN BYTE * *AID*, IN int *AIDLen*, OUT BYTE * *tlv*, IN_OUT int * *tlvLen*)

Retrieve Application Data by AID

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

Parameters

AID	Name of ApplicationID. Must be between 5 and 16 bytes
AIDLen	the length of AID data buffer.
tlv	The TLV elements of the requested AID
tlvLen	the length of tlv data buffer.

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.3.4.16 int ctls_retrieveCAPK (IN BYTE * *capk*, IN int *capkLen*, OUT BYTE * *key*, IN_OUT int * *keyLen*)

Retrieve Certificate Authority Public Key

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

Parameters

capk	6 bytes CAPK = 5 bytes RID + 1 byte Index
capkLen	the length of capk data buffer
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.

keyLen	the length of key data buffer •

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.3.4.17 int ctls_retrieveCAPKList (OUT BYTE * *keys*, IN_OUT int * *keysLen*)

Retrieve the Certificate Authority Public Key list

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

Parameters

keys	[key1][key2]...[keyn], each key 6 bytes where key = 5 bytes RID + 1 byte index
keysLen	the length of keys data buffer

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.3.4.18 int ctls_retrieveTerminalData (OUT BYTE * *tlv*, IN_OUT int * *tlvLen*)

Retrieve Terminal Data

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

Parameters

tlv	Response returned as a TLV
tlvLen	the length of tlv data buffer

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.3.4.19 int ctls_setApplicationData (IN BYTE * *tlv*, IN int *tlvLen*)

Set Application Data by AID

Sets the Application Data for CTLS as specified by TLV data

Parameters

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

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

Parameters

tlvLen	the length of tlv data buffer

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.3.4.20 int ctls_setCAPK (IN BYTE * *capk*, IN int *capkLen*)

Set Certificate Authority Public Key

Sets the CAPK as specified by the CAKey structure

Parameters

capk	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.
capkLen	the length of capk data buffer

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.3.4.21 int ctls_setConfigurationGroup (IN BYTE * *tlv*, IN int *tlvLen*)

Set Configuration Group

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

Parameters

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

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.3.4.22 int ctls_setTerminalData (IN BYTE * *tlv*, IN int *tlvLen*)

Set Terminal Data

Sets the Terminal Data for CTLS as specified by the TLV. The first TLV must be Configuration Group Number (Tag FFE4). The terminal global data is group 0, so the first TLV would be FFE40100. Other groups can be defined using this method (1 or greater), and those can be retrieved with ctls_getConfigurationGroup(int group), and deleted with ctls_removeConfigurationGroup(int group). You cannot delete group 0.

Parameters

tlv	TerminalData configuration file
tlvLen	the length of tlv data buffer

Return values

RETURN_CODE	Values can be parsed with device_getIDGStatusCodeString()

12.3.4.23 int ctls_startTransaction (IN double *amount,* IN double *amtOther,* IN int *type,* IN const int *_timeout,* IN BYTE * *tags,* IN int *tagsLen*)

Start CTLS Transaction Request

Authorizes the CTLS transaction for an ICC card

The tags will be returned in the callback routine.

Parameters

amount	Transaction amount value (tag value 9F02) - SEE IMPORTANT NOTE BELOW
amtOther	Other amount value, if any (tag value 9F03) - SEE IMPORTANT NOTE BELOW
type	Transaction type (tag value 9C).
timeout	Timeout value in seconds.
tags	Any other tags to be included in the request. Passed as TLV stream. Example, tag 9F0C with amount 0x000000000100 would be 0x9F0C06000000000100 If tags 9F02 (amount),9-F03 (other amount), or 9C (transaction type) are included, they will take priority over these values supplied as individual parameters to this method.
tagsLen	The length of tags data buffer.

>>>>>IMPORTANT: parameters for amount and amtOther MUST BE PASSED AS A DOUBLE VALUE WITH DECIMAL POINT. Example, do not pass 1, but instead pass 1.0 or 1.00. Otherwise, results will be unpredictable

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString() Note: if auto poll is on, it will returm the error IDG_P2_STATUS_CODE_COMMAND_NOT_ALLOWED

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

12.3.4.24 int device_close ( )

Close the device

Returns

RETURN_CODE: 0: success, 0x0A: failed

12.3.4.25 int device_controlUserInterface (IN BYTE * *values*)

Control User Interface

Controls the User Interface: Display, Beep, LED

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

```
- 08h: One long beep of 800 ms
Byte[2] = LED Number
- 00h: LED 0 (Power LED) 01h: LED 1
- 02h: LED 2
- 03h: LED 3
- FFh: All LEDs
Byte[3] = LED Status
- 00h: LED Off
- 01h: LED On
```

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.3.4.26 int device_enablePassThrough (int *enablePassThrough*)

Start Remote Key Injection

Starts a remote key injection request with IDTech RKI servers. This function is reserved and not implemented.

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

Enable Pass Through - NEO

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

Parameters

enablePass-Through	1 = pass through ON, 0 = pass through OFF

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.3.4.27 int device_getCurrentDeviceType ( )

Get current active device type

Returns

: return the device type defined as DEVICE_TYPE in the IDTDef.h

12.3.4.28 int device_getFirmwareVersion (OUT char * *firmwareVersion*)

DEPRECATED : please use device_getFirmwareVersion_Len(OUT char* firmwareVersion, IN_OUT int *firmware-VersionLen)

Polls device for Firmware Version

Parameters

firmwareVersion	Response returned of Firmware Version; needs to have at least 128 bytes of memory

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.3.4.29 int device_getFirmwareVersion_Len (OUT char * *firmwareVersion,* IN_OUT int * *firmwareVersionLen*)

Polls device for Firmware Version

Parameters

firmwareVersion	Response returned of Firmware Version
firmwareVersion-Len	Length of Firmware Version

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.3.4.30 void device_getIDGStatusCodeString (IN int *returnCode,* OUT char * *despcrition*)

Review the return code description.

Parameters

returnCode	the response result.
description	

Return values

the	string for description of response result

- 0: "no error, beginning task";
- 1: "no response from reader";
- 2: "invalid response data";
  - 01: " Incorrect Header Tag";
  - 02: " Unknown Command";
  - 03: " Unknown Sub-Command";
  - 04: " CRC Error in Frame";
  - 05: " Incorrect Parameter";
  - 06: " Parameter Not Supported";
  - 07: " Mal-formatted Data";
  - 08: " Timeout";
  - 0A: " Failed / NACK";
  - 0B: " Command not Allowed";
  - 0C: " Sub-Command not Allowed";
  - 0D: " Buffer Overflow (Data Length too large for reader buffer)";
  - 0E: " User Interface Event";
  - 10: " Need clear firmware(apply in boot loader only)";
  - 11: " Communication type not supported, VT-1, burst, etc. Need encrypted firmware (apply in boot loader only)";
  - 12: " Secure interface is not functional or is in an intermediate state.";
  - 13: " Data field is not mod 8";
  - 14: " Pad 0x80 not found where expected";
  - 15: " Specified key type is invalid";
  - 16: " Could not retrieve key from the SAM (InitSecureComm)";
  - 17: " Hash code problem";
  - 18: " Could not store the key into the SAM (InstallKey)";

- 19: " Frame is too large";
- 1A: " Unit powered up in authentication state but POS must resend the InitSecureComm command";
- 1B: " The EEPROM may not be initialized because SecCommInterface does not make sense";
- 1C: " Problem encoding APDU Module-Specific Status Codes ";
- 20: " Unsupported Index (ILM) SAM Transceiver error - problem communicating with the SAM (Key Mgr)";
- 21: " Unexpected Sequence Counter in multiple frames for single bitmap (ILM)Length error in data returned from the SAM (Key Mgr) - 22: " Improper bit map (ILM)"; - 23: " Request Online Authorization"; - 24: " ViVOCard3 raw data read successful"; - 25: " Message index not available (ILM) ViVOcomm activate transaction card type (ViVOcomm)"; - 26: " Version Information Mismatch (ILM)"; - 27: " Not sending commands in correct index message index (ILM)"; - 28: " Time out or next expected message not received (ILM)"; - 29: " ILM languages not available for viewing (ILM)"; - 2A: " Other language not supported (ILM)"; - 41: " from 41 to 4F, Module-specific errors for Key Manager";
- 50: " Auto-Switch OK";
- 51: " Auto-Switch failed";
- 70: " Antenna Error 80h Use another card";
- 81: " Insert or swipe card";
- 90: " Data encryption Key does not exist";
- 91: " Data encryption Key KSN exhausted";

12.3.4.31 int device_getMerchantRecord (IN int *index,* OUT BYTE * *record*)

DEPRECATED : please use device_getMerchantRecord_Len(IN int index, OUT BYTE * record, IN_OUT int *recordLen)

Get Merchant Record

Gets the merchant record for the device.

Parameters

index	Merchant Record index, valid values 1-6
record;	needs to have at least 99 bytes of memory response data from reader. Merchant Record Index: 1 byte enabled: 1 byte Merchant ID: 32 bytes Length of Merchant URL: 1 byte Merchant URL: 64 bytes

Returns

success or error code. Values can be parsed with device_getIDGStatusCodeString()

See Also

ErrorCode

12.3.4.32 int device_getMerchantRecord_Len (IN int *index,* OUT BYTE * *record,* IN_OUT int * *recordLen*)

Get Merchant Record

Gets the merchant record for the device.

Parameters

index	Merchant Record index, valid values 1-6
record	response data from reader. Merchant Record Index: 1 byte enabled: 1 byte Merchant ID: 32 bytes Length of Merchant URL: 1 byte Merchant URL: 64 bytes
recordLen	Length of record

Returns

success or error code. Values can be parsed with device_getIDGStatusCodeString()

See Also

ErrorCode

12.3.4.33 int device_getSDKWaitTime ( )

Get SDK Wait Time

Get the SDK wait time for transactions

Returns

SDK wait time in seconds

12.3.4.34 int device_getThreadStackSize ( )

Get Thread Stack Size

Get the stack size setting for newly created threads

Returns

Thread Stack Size

12.3.4.35 int device_getTransactionResults (IDTMSRData * *cardData*)

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

Parameters

cardData	The transaction results

Returns

success or error code. Values can be parsed with device_getResponseCodeString

See Also

ErrorCode

12.3.4.36 int device_init ( )

Initial the device by USB

It will detect the device and trying connect.

The connect status can be checked by device_isConnected().

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.3.4.37 int device_isAttached (int *deviceType*)

Check if the device is attached to the USB port The function device_init() must be called before this function.

Parameters

deviceType,the	device type of the USB device

Returns

1 if the device is attached, or 0 if the device is not attached

12.3.4.38 int device_isConnected ( )

Check the device conntected status

Returns

DEVICE_DISCONNECT=0, or DEVICE_CONNECTED = 1

12.3.4.39 int device_pingDevice ( )

Ping Device

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

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.3.4.40 void device_registerCameraCallBk (pCMR_callBack *pCMRf*)

To register the camera callback function to get the image data. (Pass NULL to disable the callback.)

12.3.4.41 void device_registerCardStatusFrontSwitchCallBk (pCSFS_callBack *pCSFSf*)

To register the card status and front switch callback function to get status. (Pass NULL to disable the callback.)

12.3.4.42 int device_SendDataCommandNEO (IN int *cmd,* IN int *subCmd,* IN BYTE * *data,* IN int *dataLen,* OUT BYTE * *response,* IN_OUT int * *respLen*)

Send a Command to NEO device

Sends a command to the NEO device .

Parameters

cmd	command to execute.
subCmd,sub	command to execute.
data	buffer of NEO command data.
dataLen,the	length of the buffer data.
response	Response data
respLen,the	length of Response data

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.3.4.43 int device_setBurstMode (IN BYTE *mode*)

Send Burst Mode - NEO

Sets the burst mode for the device.

Parameters

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

Returns

success or error code. Values can be parsed with device_getIDGStatusCodeString

See Also

ErrorCode

12.3.4.44 int device_setCurrentDevice (int *deviceType*)

Sets the current device to talk to

The connect status can be checked by device_isConnected().

Parameters

deviceType	Device to connect to
	
	`enum DEVICE_TYPE`
	`{`
	`IDT_DEVICE_UNKNOWN=0,`
	`IDT_DEVICE_AUGUSTA_HID,`
	`IDT_DEVICE_AUGUSTA_KB,`
	`IDT_DEVICE_AUGUSTA_S_HID,`
	`IDT_DEVICE_AUGUSTA_S_KB,`
	`IDT_DEVICE_AUGUSTA_S_TTK_HID,`
	`IDT_DEVICE_SPECTRUM_PRO,`
	`IDT_DEVICE_MINISMART_II,`
	`IDT_DEVICE_L100,`
	`IDT_DEVICE_UNIPAY,`
	`IDT_DEVICE_UNIPAY_I_V,`
	`IDT_DEVICE_VP3300_AJ,`
	`IDT_DEVICE_KIOSK_III,`
	`IDT_DEVICE_KIOSK_III_S,`
	`IDT_DEVICE_PIP_READER,`
	`IDT_DEVICE_VENDI,`
	`IDT_DEVICE_VP3300_USB,`
	`IDT_DEVICE_UNIPAY_I_V_TTK,`
	`IDT_DEVICE_VP3300_BT,`
	`IDT_DEVICE_VP8800,`
	`IDT_DEVICE_SREDKEY2_HID,`
	`IDT_DEVICE_SREDKEY2_KB,`
	`IDT_DEVICE_NEO2,`
	`IDT_DEVICE_MINISMART_II_COM = IDT_DEVICE_NEO2+5,`
	`IDT_DEVICE_SPECTRUM_PRO_COM,`
	`IDT_DEVICE_KIOSK_III_COM,`
	`IDT_DEVICE_KIOSK_III_S_COM,`
	`IDT_DEVICE_VP3300_COM,`
	`IDT_DEVICE_NEO2_COM,`
	`IDT_DEVICE_MAX_DEVICES = IDT_DEVICE_NEO2_COM+5`
	`};`

Returns

RETURN_CODE: 1: success, 0: failed

12.3.4.45 int device_setMerchantRecord (int *index*, int *enabled*, char * *merchantID*, char * *merchantURL*)

Set Merchant Record - NEO 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 device_getIDGStatusCodeString()

12.3.4.46 int device_setPollMode (IN BYTE *mode*)

Set Poll Mode - NEO

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 device_getIDGStatusCodeString()

12.3.4.47 void device_setSDKWaitTime (int *waitTime*)

Set SDK Wait Time

Set the SDK wait time for transactions

Parameters

waitTime	The SDK wait time for transaction in seconds

12.3.4.48 void device_setThreadStackSize (int *threadSize*)

Set Thread Stack Size

Set the stack size setting for newly created threads

12.3.4.49 void emv_registerCallBk (pEMV_callBack *pEMVf*)

To register the emv callback function to get the EMV processing response. (Pass NULL to disable the callback.)

12.3.4.50 void parseMSRData (IN BYTE * *resData,* IN int *resLen,* IN_OUT IDTMSRData * *cardData*)

Parser the MSR data from the buffer into IDTMSTData structure

Parameters

resData	MSR card data buffer
resLen	the length of resData
cardData	the parser result with IDTMSTData structure

12.3.4.51 void pin_registerCallBk (pPIN_callBack *pPINf*)

To register the pin callback function to get the PINPad data. (Pass NULL to disable the callback.)

12.3.4.52 void registerHotplugCallBk (pMessageHotplug *pMsgHotplug*)

To register the USB HID hot-plug callback function which implemented in the application to monitor the hotplug message from the SDK.

12.3.4.53 void registerLogCallBk (pSendDataLog *pFSend,* pReadDataLog *pFRead*)

To register the log callback function which implemented in the application to monitor sending/reading data between application and reader.

12.3.4.54 int rs232_device_init (int *deviceType,* int *port_number,* int *brate*)

Initial the device by RS232

It will try to connect to the device with provided deviceType, port_number, and brate.

Parameters

deviceType	Device to connect to
port_number	Port number of the device

Port nr. | **Linux** | **Windows**

| 0 | ttyS0 | COM1 | | 1 | ttyS1 | COM2 | | 2 | ttyS2 | COM3 | | 3 | ttyS3 | COM4 | | 4 | ttyS4 | COM5 | | 5 | ttyS5 | COM6 | | 6 | ttyS6 | COM7 | | 7 | ttyS7 | COM8 | | 8 | ttyS8 | COM9 | | 9 | ttyS9 | COM10 | | 10 | ttyS10 | COM11 | | 11 | ttyS11 | COM12 | | 12 | ttyS12 | COM13 | | 13 | ttyS13 | COM14 | | 14 | ttyS14 | COM15 | | 15 | ttyS15 | COM16 | | 16 | ttyUSB0 | n.a. | | 17 | ttyUSB1 | n.a. | | 18 | ttyUSB2 | n.a. | | 19 | ttyUSB3 | n.a. | | 20 | ttyUSB4 | n.a. | | 21 | ttyUSB5 | n.a. | | 22 | ttyAMA0 | n.a. | | 23 | ttyAMA1 | n.a. | | 24 | ttyACM0 | n.a. | | 25 | ttyACM1 | n.a. | | 26 | rfcomm0 | n.a. | | 27 | rfcomm1 | n.a. | | 28 | ircomm0 | n.a. | | 29 | ircomm1 | n.a. | | 30 | cuau0 | n.a. | | 31 | cuau1 | n.a. | | 32 | cuau2 | n.a. | | 33 | cuau3 | n.a. | | 34 | cuaU0 | n.a. | | 35 | cuaU1 | n.a. | | 36 | cuaU2 | n.a. | | 37 | cuaU3 | n.a. |

Parameters

brate	Bitrate of the device

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.3.4.55 char* SDK_Version ( )

To Get SDK version

Returns

return the SDK version string

12.3.4.56 int setAbsoluteLibraryPath (const char * *absoluteLibraryPath*)

Set the path to use when searching for ID TECH's libraries. If this is not set, the libraries will be searched for with the system's default procedures.

Parameters

absoluteLibrary-Path	The absolute path to ID TECH's libraries.

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.4 Source_C/libIDT_L100.h File Reference

L100 API.

```
#include "IDTDef.h"
```

Macros

- #define IN
- #define OUT
- #define IN_OUT

Typedefs

- typedef void(∗ pMessageHotplug)(int, int)
- typedef void(∗ pSendDataLog)(unsigned char ∗, int)
- typedef void(∗ pReadDataLog)(unsigned char ∗, int)
- typedef void(∗ pEMV_callBack)(int, int, unsigned char ∗, int, IDTTransactionData ∗, EMV_Callback ∗, int)
- typedef void(∗ pMSR_callBack)(int, IDTMSRData)
- typedef void(∗ pMSR_callBackp)(int, IDTMSRData ∗)
- typedef void(∗ pPIN_callBack)(int, IDTPINData ∗)
- typedef void(∗ pCMR_callBack)(int, IDTCMRData ∗)
- typedef void(∗ pCSFS_callBack)(BYTE status)
- typedef void(∗ pFW_callBack)(int, int, int, int, int)
- typedef void(∗ ftpComm_callBack)(int, int, int)
- typedef void(∗ httpComm_callBack)(BYTE ∗, int)
- typedef void(∗ v4Comm_callBack)(BYTE, BYTE, BYTE ∗, int)

Functions

- void registerHotplugCallBk (pMessageHotplug pMsgHotplug)
- void registerLogCallBk (pSendDataLog pFSend, pReadDataLog pFRead)
- void emv_registerCallBk (pEMV_callBack pEMVf)
- void msr_registerCallBk (pMSR_callBack pMSRf)
- void msr_registerCallBkp (pMSR_callBackp pMSRf)
- void pin_registerCallBk (pPIN_callBack pPINf)
- void device_registerCameraCallBk (pCMR_callBack pCMRf)
- void device_registerCardStatusFrontSwitchCallBk (pCSFS_callBack pCSFSf)
- void device_registerFWCallBk (pFW_callBack pFWf)
- char ∗ SDK_Version ()
- int setAbsoluteLibraryPath (const char ∗absoluteLibraryPath)
- int device_init ()
- int device_setCurrentDevice (int deviceType)
- int device_close ()
- void device_getResponseCodeString (IN int returnCode, OUT char ∗despcrition)
- int device_isConnected ()
- int device_isAttached (int deviceType)
- int device_getFirmwareVersion (OUT char ∗firmwareVersion)
- int device_getFirmwareVersion_Len (OUT char ∗firmwareVersion, IN_OUT int ∗firmwareVersionLen)
- int device_getDateTime (OUT BYTE ∗dateTime)
- int device_getDateTime_Len (OUT BYTE ∗dateTime, IN_OUT int ∗dateTimeLen)
- int device_getCurrentDeviceType ()
- int device_SendDataCommand (IN BYTE ∗cmd, IN int cmdLen, IN BYTE ∗data, IN int dataLen, OUT BYTE ∗response, IN_OUT int ∗respLen)
- int device_updateFirmware (IN BYTE ∗firmwareData, IN int firmwareDataLen, IN char ∗firmwareName, IN int encryptionType, IN BYTE ∗keyBlob, IN int keyBlobLen)
- int device_rebootDevice ()
- int device_getKeyStatus (int ∗newFormat, BYTE ∗status, int ∗statusLen)
- int device_enterStopMode ()
- int device_setSleepModeTime (int time)

- int config_getModelNumber (OUT char *sNumber)
- int config_getModelNumber_Len (OUT char *sNumber, IN_OUT int *sNumberLen)
- int config_getSerialNumber (OUT char *sNumber)
- int config_getSerialNumber_Len (OUT char *sNumber, IN_OUT int *sNumberLen)
- int pin_getEncryptedPIN (int keyType, char *PAN, int PANLen, char *message, int messageLen, int timeout)
- int pin_promptForKeyInput (int messageID, int languageID, int maskInput, int minLen, int maxLen, int timeout)
- int pin_promptForAmountInput (int messageID, int languageID, int minLen, int maxLen, int timeout)
- int pin_getFunctionKey (int timeout)
- int pin_sendBeep (int frequency, int duration)
- int pin_setKeyValues (int mode)
- int lcd_savePrompt (int promptNumber, char *prompt, int promptLen)
- int lcd_displayPrompt (int promptNumber, int lineNumber)
- int lcd_displayMessage (int lineNumber, char *message, int messageLen)
- int lcd_enableBacklight (int enable)
- int lcd_getBacklightStatus (int *enabled)

12.4.1 Detailed Description

L100 API. L100 Global API methods.

12.4.2 Macro Definition Documentation

12.4.2.1 #define IN

INPUT parameter.

12.4.2.2 #define IN_OUT

INPUT / OUTPUT PARAMETER.

12.4.2.3 #define OUT

OUTPUT parameter.

12.4.3 Typedef Documentation

12.4.3.1 typedef void(* ftpComm_callBack)(int, int, int)

Define the comm callback function to get FTP file transfer status

It should be passed as a parameter in a FTP request, Signature (int, int, int) = response code, current block, total blocks RESPONSE CODES: 100 = FILE DOWNLOAD STARTED 101 = FILE BLOCK XX OF XX RECEIVED 102 = FILE DOWNLOAD COMPLETED 103 = FILE DOWNLOAD TERMINATED PREMATURELY

12.4.3.2 typedef void(* httpComm_callBack)(BYTE *, int)

Define the comm callback function to get the async url data

It should be registered using the comm_registerHTTPCallback

12.4.3.3 typedef void(* pCMR_callBack)(int, IDTCMRData *)

Define the camera callback function to get the image data

It should be registered using the device_registerCameraCallBk,

12.4.3.4 typedef void(* pCSFS_callBack)(BYTE status)

Define the card status and front switch callback function to get card and front switch status

It should be registered using the device_registerCardStatusFrontSwitchCallBk,

12.4.3.5 typedef void(* pEMV_callBack)(int, int, unsigned char *, int, IDTTransactionData *, EMV_Callback *, int)

Define the EMV callback function to get the transaction message/data/result.

It should be registered using the emv_registerCallBk,

12.4.3.6 typedef void(* pFW_callBack)(int, int, int, int, int)

Define the firmware update callback function to get the status of firmware update

It should be registered using the device_registerFWCallBk,

12.4.3.7 typedef void(* pMessageHotplug)(int, int)

Define the USB hot-plug callback function to monitor the info when plug in/out the reader.

It should be registered using the registerHotplugCallBk, The first integer parameter is device type, and the second integer parameter is either 0: Device Plugged Out or 1: Device Plugged In

12.4.3.8 typedef void(* pMSR_callBack)(int, IDTMSRData)

Define the MSR callback function to get the MSR card data

It should be registered using the msr_registerCallBk, this callback function is for backward compatibility

12.4.3.9 typedef void(* pMSR_callBackp)(int, IDTMSRData *)

Define the MSR callback function to get pointer to the MSR card data

It should be registered using the msr_registerCallBk, this callback function is recommended instead of pMSR_callBack

12.4.3.10 typedef void(* pPIN_callBack)(int, IDTPINData *)

Define the PINPad callback function to get the input PIN Pad data

It should be registered using the pin_registerCallBk,

12.4.3.11 typedef void(* pReadDataLog)(unsigned char *, int)

Define the read response callback function to monitor the reading response from the reader.

It should be registered using the registerLogCallBk,

12.4.3.12 typedef void(∗ pSendDataLog)(unsigned char ∗, int)

Define the send command callback function to monitor the sending command into the reader.

It should be registered using the registerLogCallBk,

12.4.3.13 typedef void(∗ v4Comm_callBack)(BYTE, BYTE, BYTE ∗, int)

Define the comm callback function to receive the V4 Protocol packets received by the device from an external source (IP/USB/RS-232) It should be registered using the comm_registerV4Callback, Data callback will contain command, sub-command, and data from V4 packet

12.4.4 Function Documentation

12.4.4.1 int config_getModelNumber (OUT char ∗ *sNumber*)

DEPRECATED : please use config_getModelNumber_Len(OUT char∗ sNumber, IN_OUT int ∗sNumberLen)

Polls device for Model Number

Parameters

sNumber	Returns Model Number; needs to have at least 64 bytes of memory

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.4.4.2 int config_getModelNumber_Len (OUT char ∗ *sNumber,* IN_OUT int ∗ *sNumberLen*)

Polls device for Model Number

Parameters

sNumber	Returns Model Number
sNumber	length of Model Number

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.4.4.3 int config_getSerialNumber (OUT char ∗ *sNumber*)

DEPRECATED : please use config_getSerialNumber_Len(OUT char∗ sNumber, IN_OUT int ∗sNumberLen)

Polls device for Serial Number

Parameters

sNumber	Returns Serial Number; needs to have at least 64 bytes of memory

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.4.4.4 int config_getSerialNumber_Len (OUT char ∗ *sNumber,* IN_OUT int ∗ *sNumberLen*)

Polls device for Serial Number

Parameters

sNumber	Returns Serial Number
sNumberLen	Length of Serial Number

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.4.4.5 int device_close ( )

Close the device

Returns

RETURN_CODE: 0: success, 0x0A: failed

12.4.4.6 int 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

12.4.4.7 int device_getCurrentDeviceType ( )

Get current active device type

Returns

: return the device type defined as DEVICE_TYPE in the IDTDef.h

12.4.4.8 int device_getDateTime (OUT BYTE * *dateTime*)

Polls device for Date and Time

Parameters

dateTime	Response returned of Date and Time; needs to have at least 6 bytes of memory

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.4.4.9 int device_getDateTime_Len (OUT BYTE * *dateTime,* IN_OUT int * *dateTimeLen*)

Polls device for Date and Time

Parameters

dateTime	Response returned of Date and Time
dateTime	Length of Date and Time

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.4.4.10 int device_getFirmwareVersion (OUT char * *firmwareVersion*)

DEPRECATED : please use device_getFirmwareVersion_Len(OUT char* firmwareVersion, IN_OUT int *firmwareVersionLen)

Polls device for Firmware Version

Parameters

firmwareVersion	Response returned of Firmware Version; needs to have at least 128 bytes of memory

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.4.4.11 int device_getFirmwareVersion_Len (OUT char * *firmwareVersion,* IN_OUT int * *firmwareVersionLen*)

Polls device for Firmware Version

Parameters

firmwareVersion	Response returned of Firmware Version
firmwareVersion-Len	Length of Firmware Version

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.4.4.12 int device_getKeyStatus (int * *newFormat,* BYTE * *status,* int * *statusLen*)

Get Key Status

Gets the status of loaded keys

Parameters

status	newFormat for Augusta and miniSmartII only 1: new format of key status 0: reserved format for support previous device
status	For L100, Augusta and miniSmartII: When the newFormat is 0, data format as follows. For Augusta and miniSmartII: byte 0: PIN DUKPT Key, Does not support, always 0 byte 1: PIN Master Key, Does not support, always 0 byte 2: PIN Session Key, Does not support, always 0 byte 3: Account/MSR DUKPT Key, 1 Exists, 0 None, 0xFF STOP byte 4: Account/ICC DUKPT Key, 1 Exists, 0 None, 0xFF STOP byte 5: Admin DUKPT Key, 1 Exists, 0 None, 0xFF STOP For L100: byte 0: PIN DUKPT Key byte 1: PIN Master Key byte 2: Standard PIN Session Key byte 3: Desjardins PIN Session Key byte 4: Account/MSR DUKPT Key, 1 Exists, 0 None, 0xFF STOP, Does not support, always 0 byte 5: Account/ICC DUKPT Key, 1 Exists, 0 None, 0xFF STOP, Does not support, always 0 byte 6: Admin DUKPT Key, 1 Exists, 0 None, 0xFF STOP byte 7: Data DUKPT Key, 1 Exists, 0 None, 0xFF STOP byte 8: MAC DUKPT Key, 1 Exists, 0 None, 0xFF STOP

when the newFormat is 1, data format as follows. [Block Length] [KeyStatusBlock1] [KeyStatusBlock2]...[KeyStatusBlockN] Where: [Block Length] is 2 bytes, format is Len_L Len_H, is KeyStatusBlock Number [KeyStatusBlockX> is 4 bytes, format is [Key Index and Key Name] [key slot] [key status]: [Key Index and Key Name] is 1 byte. Please refer to following table 0x14 LCL-KEK to Encrypt Other Keys 0x02 Data encryption Key to Encrypt ICC/MSR 0x05 MAC DUKPT Key for Host-Device - MAC Verification 0x05 MTK DUKPT Key for TTK Self-Test 0x0C RKI-KEK for Remote Key Injection [key slot] is 2 bytes. Range is 0 - 9999 the MTK DUKPT Key slot is 16, the others are all 0 [key status] is 1 byte. 0 - Not Exist 1 - Exist 0xFF - (Stop. Only Valid for DUKPT Key) For NEO2 and SREDKey2: Each unit of three bytes represents one key's parameters (index and slot). Key Name Index (1 byte): 0x14 - LCL-KEK 0x01 - Pin encryption Key (NEO2 only) 0x02 - Data encryption Key 0x05 - MAC DUKPT Key 0x0A - PCI Pairing Key (NEO2 only) Key Slot (2 bytes): Indicate different slots of a certain Key Name Example: slot =5 (0x00 0x05), slot=300 (0x01 0x2C) For BTPay380, slot is always 0 For example, 0x14 0x00 0x00 0x02 0x00 0x00 0x0A 0x00 0x00 will represent [KeyNameIndex=0x14,KeySlot=0x0000], [KeyNameIndex=0x02,KeySlot=0x0000] and [KeyNameIndex=0x0A,KeySlot=0x0000]

Parameters

statusLen	the length of status

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.4.4.13 void device_getResponseCodeString (IN int *returnCode,* OUT char * *despcrition*)

Review the return code description.

Parameters

returnCode	the response result.
description	

Return values

the	string for description of response result

- 0: "no error, beginning task";
- 1: "no response from reader";
- 2: "invalid response data";
- 3: "time out for task or CMD";
- 4: "wrong parameter";
- 5: "SDK is doing MSR or ICC task";
- 6: "SDK is doing PINPad task";
- 7: "SDK is doing CTLS task";
- 8: "SDK is doing EMV task";
- 9: "SDK is doing Other task";
- 10: "err response or data";
- 11: "no reader attached";
- 12: "mono audio is enabled";
- 13: "did connection";
- 14: "audio volume is too low";

- 15: "task or CMD be canceled";
- 16: "UF wrong string format";
- 17: "UF file not found";
- 18: "UF wrong file format";
- 19: "Attempt to contact online host failed";
- 20: "Attempt to perform RKI failed";
- 22: "Buffer size is not enough";
- 0X300: "Key Type(TDES) of Session Key is not same as the related Master Key.";
- 0X400: "Related Key was not loaded.";
- 0X500: "Key Same.";
- 0X501: "Key is all zero";
- 0X502: "TR-31 format error";
- 0X702: "PAN is Error Key.";
- 0X705: "No Internal MSR PAN (or Internal MSR PAN is erased timeout)";
- 0X0C01: "Incorrect Frame Tag";
- 0X0C02: "Incorrect Frame Type";
- 0X0C03: "Unknown Frame Type";
- 0X0C04: "Unknown Command";
- 0X0C05: "Unknown Sub-Command";
- 0X0C06: "CRC Error";
- 0X0C07: "Failed";
- 0X0C08: "Timeout";
- 0X0C0A: "Incorrect Parameter";
- 0X0C0B: "Command Not Supported";
- 0X0C0C: "Sub-Command Not Supported";
- 0X0C0D: "Parameter Not Supported / Status Abort Command";
- 0X0C0F: "Sub-Command Not Allowed";
- 0X0D01: "Incorrect Header Tag";
- 0X0D02: "Unknown Command";
- 0X0D03: "Unknown Sub-Command";
- 0X0D04: "CRC Error in Frame";
- 0X0D05: "Incorrect Parameter";
- 0X0D06: "Parameter Not Supported";
- 0X0D07: "Mal-formatted Data";
- 0X0D08: "Timeout";
- 0X0D0A: "Failed / NACK";

- 0X0D0B: "Command not Allowed";
- 0X0D0C: "Sub-Command not Allowed";
- 0X0D0D: "Buffer Overflow (Data Length too large for reader buffer)";
- 0X0D0E: "User Interface Event";
- 0X0D11: "Communication type not supported, VT-1, burst, etc.";
- 0X0D12: "Secure interface is not functional or is in an intermediate state.";
- 0X0D13: "Data field is not mod 8";
- 0X0D14: "Pad - 0X80 not found where expected";
- 0X0D15: "Specified key type is invalid";
- 0X0D1: "Could not retrieve key from the SAM(InitSecureComm)";
- 0X0D17: "Hash code problem";
- 0X0D18: "Could not store the key into the SAM(InstallKey)";
- 0X0D19: "Frame is too large";
- 0X0D1A: "Unit powered up in authentication state but POS must resend the InitSecureComm command";
- 0X0D1B: "The EEPROM may not be initialized because SecCommInterface does not make sense";
- 0X0D1C: "Problem encoding APDU";
- 0X0D20: "Unsupported Index(ILM) SAM Transceiver error - problem communicating with the SAM(Key Mgr)";
- 0X0D2: "Unexpected Sequence Counter in multiple frames for single bitmap(ILM) Length error in data returned from the SAM(Key Mgr)";
- 0X0D22: "Improper bit map(ILM)";
- 0X0D23: "Request Online Authorization";
- 0X0D24: "ViVOCard3 raw data read successful";
- 0X0D25: "Message index not available(ILM) ViVOcomm activate transaction card type(ViVOcomm)";
- 0X0D26: "Version Information Mismatch(ILM)";
- 0X0D27: "Not sending commands in correct index message index(ILM)";
- 0X0D28: "Time out or next expected message not received(ILM)";
- 0X0D29: "ILM languages not available for viewing(ILM)";
- 0X0D2A: "Other language not supported(ILM)";
- 0X0D41: "Unknown Error from SAM";
- 0X0D42: "Invalid data detected by SAM";
- 0X0D43: "Incomplete data detected by SAM";
- 0X0D44: "Reserved";
- 0X0D45: "Invalid key hash algorithm";
- 0X0D46: "Invalid key encryption algorithm";
- 0X0D47: "Invalid modulus length";
- 0X0D48: "Invalid exponent";

- 0X0D49: "Key already exists";
- 0X0D4A: "No space for new RID";
- 0X0D4B: "Key not found";
- 0X0D4C: "Crypto not responding";
- 0X0D4D: "Crypto communication error";
- 0X0D4E: "Module-specific error for Key Manager";
- 0X0D4F: "All key slots are full (maximum number of keys has been installed)";
- 0X0D50: "Auto-Switch OK";
- 0X0D51: "Auto-Switch failed";
- 0X0D90: "Account DUKPT Key not exist";
- 0X0D91: "Account DUKPT Key KSN exausted";
- 0X0D00: "This Key had been loaded.";
- 0X0E00: "Base Time was loaded.";
- 0X0F00: "Encryption Or Decryption Failed.";
- 0X1000: "Battery Low Warning (It is High Priority Response while Battery is Low.)";
- 0X1800: "Send 'Cancel Command' after send 'Get Encrypted PIN' & 'Get Numeric' & 'Get Amount'; - 0X1900: "Press 'Cancel' key after send 'Get Encrypted PIN' & 'Get Numeric' & 'Get Amount';
- 0X30FF: "Security Chip is not connect";
- 0X3000: "Security Chip is deactivation & Device is In Removal Legally State.";
- 0X3101: "Security Chip is activation & Device is In Removal Legally State.";
- 0X5500: "No Admin DUKPT Key.";
- 0X5501: "Admin DUKPT Key STOP.";
- 0X5502: "Admin DUKPT Key KSN is Error.";
- 0X5503: "Get Authentication Code1 Failed.";
- 0X5504: "Validate Authentication Code Error.";
- 0X5505: "Encrypt or Decrypt data failed.";
- 0X5506: "Not Support the New Key Type.";
- 0X5507: "New Key Index is Error.";
- 0X5508: "Step Error.";
- 0X5509: "KSN Error";
- 0X550A: "MAC Error.";
- 0X550B: "Key Usage Error.";
- 0X550C: "Mode Of Use Error.";
- 0X550F: "Other Error.";
- 0X6000: "Save or Config Failed / Or Read Config Error.";
- 0X6200: "No Serial Number.";

- 0X6900: "Invalid Command - Protocol is right, but task ID is invalid.";
- 0X6A01: "Unsupported Command - Protocol and task ID are right, but command is invalid - In this State";
- 0X6A00: "Unsupported Command - Protocol and task ID are right, but command is invalid.";
- 0X6B00: "Unknown parameter in command - Protocol task ID and command are right, but parameter is invalid.";
- 0X6C00: "Unknown parameter in command - Protocol task ID and command are right, but length is out of the requirement.";
- 0X7200: "Device is suspend (MKSK suspend or press password suspend).";
- 0X7300: "PIN DUKPT is STOP (21 bit 1).";
- 0X7400: "Device is Busy.";
- 0XE100: "Can not enter sleep mode";
- 0XE200: "File has existed";
- 0XE300: "File has not existed";
- 0XE313: "IO line low -- Card error after session start";
- 0XE400: "Open File Error";
- 0XE500: "SmartCard Error";
- 0XE600: "Get MSR Card data is error";
- 0XE700: "Command time out";
- 0XE800: "File read or write is error";
- 0XE900: "Active 1850 error!";
- 0XEA00: "Load bootloader error";
- 0XEF00: "Protocol Error- STX or ETX or check error.";
- 0XEB00: "Picture is not exist";
- 0X2C02: "No Microprocessor ICC seated";
- 0X2C06: "no card seated to request ATR";
- 0X2D01: "Card Not Supported,";
- 0X2D03: "Card Not Supported, wants CRC";
- 0X690D: "Command not supported on reader without ICC support";
- 0X8100: "ICC error time out on power-up";
- 0X8200: "invalid TS character received - Wrong operation step";
- 0X8300: "Decode MSR Error";
- 0X8400: "TriMagII no Response";
- 0X8500: "No Swipe MSR Card";
- 0X8510: "No Financial Card";
- 0X8600: "Unsupported F, D, or combination of F and D";
- 0X8700: "protocol not supported EMV TD1 out of range";
- 0X8800: "power not at proper level";

- 0X8900: "ATR length too long";
- 0X8B01: "EMV invalid TA1 byte value";
- 0X8B02: "EMV TB1 required";
- 0X8B03: "EMV Unsupported TB1 only 00 allowed";
- 0X8B04: "EMV Card Error, invalid BWI or CWI";
- 0X8B06: "EMV TB2 not allowed in ATR";
- 0X8B07: "EMV TC2 out of range";
- 0X8B08: "EMV TC2 out of range";
- 0X8B09: "per EMV96 TA3 must be $>$ - 0XF";
- 0X8B10: "ICC error on power-up";
- 0X8B11: "EMV T=1 then TB3 required";
- 0X8B12: "Card Error, invalid BWI or CWI";
- 0X8B13: "Card Error, invalid BWI or CWI";
- 0X8B17: "EMV TC1/TB3 conflict-";
- 0X8B20: "EMV TD2 out of range must be T=1";
- 0X8C00: "TCK error";
- 0XA304: "connector has no voltage setting";
- 0XA305: "ICC error on power-up invalid (SBLK(IFSD) exchange";
- 0XE301: "ICC error after session start";
- 0XFF00: "Request to go online";
- 0XFF01: "EMV: Accept the offline transaction";
- 0XFF02: "EMV: Decline the offline transaction";
- 0XFF03: "EMV: Accept the online transaction";
- 0XFF04: "EMV: Decline the online transaction";
- 0XFF05: "EMV: Application may fallback to magstripe technology";
- 0XFF06: "EMV: ICC detected tah the conditions of use are not satisfied";
- 0XFF07: "EMV: ICC didn't accept transaction";
- 0XFF08: "EMV: Transaction was cancelled";
- 0XFF09: "EMV: Application was not selected by kernel or ICC format error or ICC missing data error";
- 0XFF0A: "EMV: Transaction is terminated";
- 0XFF0B: "EMV: Other EMV Error";
- 0XFFFF: "NO RESPONSE";
- 0XF002: "ICC communication timeout";
- 0XF003: "ICC communication Error";
- 0XF00F: "ICC Card Seated and Highest Priority, disable MSR work request";
- 0XF200: "AID List / Application Data is not exist";

- 0XF201: "Terminal Data is not exist";
- 0XF202: "TLV format is error";
- 0XF203: "AID List is full";
- 0XF204: "Any CA Key is not exist";
- 0XF205: "CA Key RID is not exist";
- 0XF206: "CA Key Index it not exist";
- 0XF207: "CA Key is full";
- 0XF208: "CA Key Hash Value is Error";
- 0XF209: "Transaction format error";
- 0XF20A: "The command will not be processing";
- 0XF20B: "CRL is not exist";
- 0XF20C: "CRL number exceed max number";
- 0XF20D: "Amount,Other Amount,Trasaction Type are missing";
- 0XF20E: "The Identification of algorithm is mistake";
- 0XF20F: "No Financial Card";
- 0XF210: "In Encrypt Result state, TLV total Length is greater than Max Length";
- 0X1001: "INVALID ARG";
- 0X1002: "FILE_OPEN_FAILED";
- 0X1003: "FILE OPERATION_FAILED";
- 0X2001: "MEMORY_NOT_ENOUGH";
- 0X3002: "SMARTCARD_FAIL";
- 0X3003: "SMARTCARD_INIT_FAILED";
- 0X3004: "FALLBACK_SITUATION";
- 0X3005: "SMARTCARD_ABSENT";
- 0X3006: "SMARTCARD_TIMEOUT";
- 0X3012: "EMV_RESULT_CODE_MSR_CARD_ERROR_FALLBACK";
- 0X5001: "EMV_PARSING_TAGS_FAILED";
- 0X5002: "EMV_DUPLICATE_CARD_DATA_ELEMENT";
- 0X5003: "EMV_DATA_FORMAT_INCORRECT";
- 0X5004: "EMV_NO_TERM_APP";
- 0X5005: "EMV_NO_MATCHING_APP";
- 0X5006: "EMV_MISSING_MANDATORY_OBJECT";
- 0X5007: "EMV_APP_SELECTION_RETRY";
- 0X5008: "EMV_GET_AMOUNT_ERROR";
- 0X5009: "EMV_CARD_REJECTED";
- 0X5010: "EMV_AIP_NOT_RECEIVED";

- 0X5011: "EMV_AFL_NOT_RECEIVED";
- 0X5012: "EMV_AFL_LEN_OUT_OF_RANGE";
- 0X5013: "EMV_SFI_OUT_OF_RANGE";
- 0X5014: "EMV_AFL_INCORRECT";
- 0X5015: "EMV_EXP_DATE_INCORRECT";
- 0X5016: "EMV_EFF_DATE_INCORRECT";
- 0X5017: "EMV_ISS_COD_TBL_OUT_OF_RANGE";
- 0X5018: "EMV_CRYPTOGRAM_TYPE_INCORRECT";
- 0X5019: "EMV_PSE_NOT_SUPPORTED_BY_CARD";
- 0X5020: "EMV_USER_SELECTED_LANGUAGE";
- 0X5021: "EMV_SERVICE_NOT_ALLOWED";
- 0X5022: "EMV_NO_TAG_FOUND";
- 0X5023: "EMV_CARD_BLOCKED";
- 0X5024: "EMV_LEN_INCORRECT";
- 0X5025: "CARD_COM_ERROR";
- 0X5026: "EMV_TSC_NOT_INCREASED";
- 0X5027: "EMV_HASH_INCORRECT";
- 0X5028: "EMV_NO_ARC";
- 0X5029: "EMV_INVALID_ARC";
- 0X5030: "EMV_NO_ONLINE_COMM";
- 0X5031: "TRAN_TYPE_INCORRECT";
- 0X5032: "EMV_APP_NO_SUPPORT";
- 0X5033: "EMV_APP_NOT_SELECT";
- 0X5034: "EMV_LANG_NOT_SELECT";
- 0X5035: "EMV_NO_TERM_DATA";
- 0X5039: "EMV_PIN_ENTRY_TIMEOUT";
- 0X6001: "CVM_TYPE_UNKNOWN";
- 0X6002: "CVM_AIP_NOT_SUPPORTED";
- 0X6003: "CVM_TAG_8E_MISSING";
- 0X6004: "CVM_TAG_8E_FORMAT_ERROR";
- 0X6005: "CVM_CODE_IS_NOT_SUPPORTED";
- 0X6006: "CVM_COND_CODE_IS_NOT_SUPPORTED";
- 0X6007: "NO_MORE_CVM";
- 0X6008: "PIN_BYPASSED_BEFORE";
- 0X7001: "PK_BUFFER_SIZE_TOO_BIG";
- 0X7002: "PK_FILE_WRITE_ERROR";

- 0X7003: "PK_HASH_ERROR";
- 0X8001: "NO_CARD_HOLDER_CONFIRMATION";
- 0X8002: "GET_ONLINE_PIN";
- 0XD000: "Data not exist";
- 0XD001: "Data access error";
- 0XD100: "RID not exist";
- 0XD101: "RID existed";
- 0XD102: "Index not exist";
- 0XD200: "Maximum exceeded";
- 0XD201: "Hash error";
- 0XD205: "System Busy";
- 0X0E01: "Unable to go online";
- 0X0E02: "Technical Issue";
- 0X0E03: "Declined";
- 0X0E04: "Issuer Referral transaction";
- 0X0F01: "Decline the online transaction";
- 0X0F02: "Request to go online";
- 0X0F03: "Transaction is terminated";
- 0X0F05: "Application was not selected by kernel or ICC format error or ICC missing data error";
- 0X0F07: "ICC didn't accept transaction";
- 0X0F0A: "Application may fallback to magstripe technology";
- 0X0F0C: "Transaction was cancelled";
- 0X0F0D: "Timeout";
- 0X0F0F: "Other EMV Error";
- 0X0F10: "Accept the offline transaction";
- 0X0F11: "Decline the offline transaction";
- 0X0F21: "ICC detected tah the conditions of use are not satisfied";
- 0X0F22: "No app were found on card matching terminal configuration";
- 0X0F23: "Terminal file does not exist";
- 0X0F24: "CAPK file does not exist";
- 0X0F25: "CRL Entry does not exist";
- 0X0FFE: "code when blocking is disabled";
- 0X0FFF: "code when command is not applicable on the selected device";
- 0XF005: "ICC Encrypted C-APDU Data Structure Length Error Or Format Error.";
- 0XBBE0: "CM100 Success";
- 0XBBE1: "CM100 Parameter Error";

- 0XBBE2: "CM100 Low Output Buffer";
- 0XBBE3: "CM100 Card Not Found";
- 0XBBE4: "CM100 Collision Card Exists";
- 0XBBE5: "CM100 Too Many Cards Exist";
- 0XBBE6: "CM100 Saved Data Does Not Exist";
- 0XBBE8: "CM100 No Data Available";
- 0XBBE9: "CM100 Invalid CID Returned";
- 0XBBEA: "CM100 Invalid Card Exists";
- 0XBBEC: "CM100 Command Unsupported";
- 0XBBED: "CM100 Error In Command Process";
- 0XBBEE: "CM100 Invalid Command";
- 0X9031: "Unknown command";
- 0X9032: "Wrong parameter (such as the length of the command is incorrect)";
- 0X9038: "Wait (the command couldnt be finished in BWT)";
- 0X9039: "Busy (a previously command has not been finished)";
- 0X903A: "Number of retries over limit";
- 0X9040: "Invalid Manufacturing system data";
- 0X9041: "Not authenticated";
- 0X9042: "Invalid Master DUKPT Key";
- 0X9043: "Invalid MAC Key";
- 0X9044: "Reserved for future use";
- 0X9045: "Reserved for future use";
- 0X9046: "Invalid DATA DUKPT Key";
- 0X9047: "Invalid PIN Pairing DUKPT Key";
- 0X9048: "Invalid DATA Pairing DUKPT Key";
- 0X9049: "No nonce generated";
- 0X9949: "No GUID available. Perform getVersion first.";
- 0X9950: "MAC Calculation unsuccessful. Check BDK value.";
- 0X904A: "Not ready";
- 0X904B: "Not MAC data";
- 0X9050: "Invalid Certificate";
- 0X9051: "Duplicate key detected";
- 0X9052: "AT checks failed";
- 0X9053: "TR34 checks failed";
- 0X9054: "TR31 checks failed";
- 0X9055: "MAC checks failed";

- 0X9056: "Firmware download failed";
- 0X9060: "Log is full";
- 0X9061: "Removal sensor unengaged";
- 0X9062: "Any hardware problems";
- 0X9070: "ICC communication timeout";
- 0X9071: "ICC data error (such check sum error)";
- 0X9072: "Smart Card not powered up";

12.4.4.14 int device_init ( )

Initial the device by USB

It will detect the device and trying connect.

The connect status can be checked by device_isConnected().

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.4.4.15 int device_isAttached (int *deviceType*)

Check if the device is attached to the USB port The function device_init() must be called before this function.

Parameters

deviceType,the	device type of the USB device

Returns

1 if the device is attached, or 0 if the device is not attached

12.4.4.16 int device_isConnected ( )

Check the device conntected status

Returns

DEVICE_DISCONNECT=0, or DEVICE_CONNECTED = 1

12.4.4.17 int 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()

12.4.4.18 void device_registerCameraCallBk (pCMR_callBack *pCMRf*)

To register the camera callback function to get the image data. (Pass NULL to disable the callback.)

12.4.4.19 void device_registerCardStatusFrontSwitchCallBk (pCSFS_callBack *pCSFSf*)

To register the card status and front switch callback function to get status. (Pass NULL to disable the callback.)

12.4.4.20 void device_registerFWCallBk (pFW_callBack *pFWf*)

To register the firmware update callback function to get the status of firmware update. (Pass NULL to disable the callback.)

12.4.4.21 int device_SendDataCommand (IN BYTE * *cmd,* IN int *cmdLen,* IN BYTE * *data,* IN int *dataLen,* OUT BYTE * *response,* IN_OUT int * *respLen*)

Send a Command to device

Sends a command to the device .

Parameters

cmd	buffer of command to execute.
cmdLen,the	length of the buffer cmd.
data	buffer of IDG command data.
dataLen,the	length of the buffer data.
response	Response data
respLen,the	length of Response data

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.4.4.22 int device_setCurrentDevice (int *deviceType*)

Sets the current device to talk to

The connect status can be checked by device_isConnected().

Parameters

deviceType	Device to connect to
	
	`enum DEVICE_TYPE`
	`{`
	`IDT_DEVICE_UNKNOWN=0,`
	`IDT_DEVICE_AUGUSTA_HID,`
	`IDT_DEVICE_AUGUSTA_KB,`
	`IDT_DEVICE_AUGUSTA_S_HID,`
	`IDT_DEVICE_AUGUSTA_S_KB,`
	`IDT_DEVICE_AUGUSTA_S_TTK_HID,`
	`IDT_DEVICE_SPECTRUM_PRO,`
	`IDT_DEVICE_MINISMART_II,`
	`IDT_DEVICE_L100,`
	`IDT_DEVICE_UNIPAY,`
	`IDT_DEVICE_UNIPAY_I_V,`
	`IDT_DEVICE_VP3300_AJ,`
	`IDT_DEVICE_KIOSK_III,`
	`IDT_DEVICE_KIOSK_III_S,`
	`IDT_DEVICE_PIP_READER,`
	`IDT_DEVICE_VENDI,`
	`IDT_DEVICE_VP3300_USB,`
	`IDT_DEVICE_UNIPAY_I_V_TTK,`
	`IDT_DEVICE_VP3300_BT,`
	`IDT_DEVICE_VP8800,`
	`IDT_DEVICE_SREDKEY2_HID,`
	`IDT_DEVICE_SREDKEY2_KB,`
	`IDT_DEVICE_NEO2,`
	`IDT_DEVICE_MINISMART_II_COM = IDT_DEVICE_NEO2+5,`
	`IDT_DEVICE_SPECTRUM_PRO_COM,`
	`IDT_DEVICE_KIOSK_III_COM,`
	`IDT_DEVICE_KIOSK_III_S_COM,`
	`IDT_DEVICE_VP3300_COM,`
	`IDT_DEVICE_NEO2_COM,`
	`IDT_DEVICE_MAX_DEVICES = IDT_DEVICE_NEO2_COM+5`
	`};`

Returns

RETURN_CODE: 1: success, 0: failed

12.4.4.23 int 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

12.4.4.24 int device_updateFirmware (IN BYTE * *firmwareData,* IN int *firmwareDataLen,* IN char * *firmwareName,* IN int *encryptionType,* IN BYTE * *keyBlob,* IN int *keyBlobLen*)

Update Firmware Updates the firmware of Augusta.

Parameters

firmwareData	Signed binary data of a firmware file provided by IDTech
firmwareData-Len	Length of firmwareData
firmwareName	Firmware name. • For example "Augusta_S_TTK_V1.00.002.fm"
encryptionType	Encryption type • 0 : Plaintext • 1 : TDES ECB, PKCS#5 padding • 2 : TDES CBC, PKCS#5, IV is all 0
keyBlob	Encrypted firmware session key blob, TR-31 Rev B, wrapped by FW Key (Optional, none if firmware is plaintext)
keyBlobLen	Length of keyBlob

Returns

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

Firmware update status is returned in the callback with the following values: sender = AUGUSTA state = DeviceState.FirmwareUpdate data = File Progress. Two bytes, with byte[0] = current block, and byte[1] = total blocks. 0x0310 = block 3 of 16 transactionResultCode:

- RETURN_CODE_DO_SUCCESS = Firmware Update Completed Successfully
- RETURN_CODE_BLOCK_TRANSFER_SUCCESS = Current block transferred successfully
- Any other return code represents an error condition

12.4.4.25 void emv_registerCallBk (pEMV_callBack *pEMVf*)

To register the emv callback function to get the EMV processing response. (Pass NULL to disable the callback.)

12.4.4.26 int lcd_displayMessage (int *lineNumber,* char * *message,* int *messageLen*)

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
messageLen	length of message

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.4.4.27 int 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

12.4.4.28 int lcd_enableBacklight (int *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

12.4.4.29 int lcd_getBacklightStatus (int * *enabled*)

Get Backlight Status

Returns the status of the LCD back lighting.

Parameters

enabled	1 = Backlight is ON, 0 = Backlight is OFF

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.4.4.30 int lcd_savePrompt (int *promptNumber,* char * *prompt,* int *promptLen*)

Save Prompt

Saves a message prompt to L100 memory.

Parameters

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

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.4.4.31 void msr_registerCallBk (pMSR_callBack *pMSRf*)

To register the msr callback function to get the MSR card data. (Pass NULL to disable the callback.)

12.4.4.32 void msr_registerCallBkp (pMSR_callBackp *pMSRf*)

To register the msr callback function to get the MSR card data pointer. (Pass NULL to disable the callback.)

12.4.4.33 int pin_getEncryptedPIN (int *keyType,* char * *PAN,* int *PANLen,* char * *message,* int *messageLen,* 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
PANLen	length of PAN
message	Message to display
messageLen	length of message
timeout	PIN entry timeout

Returns

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

12.4.4.34 int pin_getFunctionKey (int *timeout*)

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

Parameters

timeout	Timeout, in seconds

Returns

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

12.4.4.35 int pin_promptForAmountInput (int *messageID,* int *languageID,* int *minLen,* int *maxLen,* int *timeout*)

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
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	N筋ERO PO	NUMERO PO	No COMMANDE
8	DRIVER ID	LICEN斤墩	LICENCIA	ID CONDUCTEUR
9	ODOMETER	ODOMETER	ODOMETRO	ODOMETRE
10	ID NUMBER	N筋ERO 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 N筋ERO	NUMERO EMP	No TRAVAIL
14	WORK ORDER	TRABALHO ORDEM	ORDEN TRABAJO	FICHE TRAVAIL
15	VEHICLE ID	ID VE斤ULO	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	N筋ERO PATH	RUTA NUMERO	No ROUTE
23	ENTER USER ID	ENTER USER ID	INGRESE ID USUARIO	ID UTILISATEUR
24	FLEET NUMBER	N筋ERO DE FROTA	FLOTA NUMERO	No PARC AUTO

25	ENTER PRODUCT	ADICIONAR PRODUTO	INGRESE PRODUCTO	ENTREZ PRODUIT
26	DRIVER NUMBER	N筋ERO DRIVER	CONDUCTOR NUMERO	No CONDUCTEUR
27	ENTER LICENSE	ENTER LICEN斤墩	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 VE斤ULO	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	N筋ERO DO VE斤ULO	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 USU筋粋IO	INGRESE INFO USUARIO	INFO UTILISATEUR
39	ENTER CUST CODE	ENTER CODE. CLIENTE	INGRESE COD. CLIENTE	ENTR CODE CLIENT
40	ENTER EMPLOYEE	ENTER FUNCION筋粋IO	INGRESE EMPLEADO	ENTREZ EMPLOYE
41	ENTER ID NUMBER	ENTER N筋ERO 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	N筋ERO 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 VE斤ULO	REINGRESE VEHICULO	RE-ENTR VEHICULE
48	ENTER VEHICLE ID	ENTER VE斤ULO 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 REFER斤憍CIA	INGRESE REFERENCIA	ENTREZ REFERENCE
53	ENTER AUTH NUMBR	ENTER N筋ERO 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	QUILOMETRAGE-M	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 #	VE斤ULO #	VEHICULO #	# VEHICULE
68	ENTER VEHICLE #	ENTER VE斤ULO #	INGRESE VEHICULO #	ENT # VEHICULE
69	JOB #	TRABALHO #	TRABAJO #	# TRAVAIL
70	ENTER JOB #	ENTER JOB #	INGRESE TRABAJO #	ENTREZ # TRAVAIL
71	DEPT NUMBER	N筋ERO DEPT	NUMERO DEPTO	No DEPARTEMENT
72	DEPARTMENT #	DEPARTAMENTO #	DEPARTAMENTO #	DEPARTEMENT
73	ENTER DEPT #	ENTER DEPT #	INGRESE DEPTO #	ENT# DEPARTEMENT
74	LICENSE NUMBER	N筋ERO DE LICEN斤墩	NUMERO LICENCIA	No PERMIS
75	LICENSE #	LICEN斤墩 #	LICENCIA #	# PERMIS
76	ENTER LICENSE #	ENTER LICEN斤墩 #	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 N筋ERO DE CART斤	INGRESE NUM TARJETA	ENTREZ NO CARTE
86	EXP DATE(YYMM)	VALIDADE VAL (AAMM)	FECHA EXP (AAMM)	DATE EXPIR(AAMM)
87	PHONE NUMBER	TELEFONE	NUMERO TELEFONO	NO TEL
88	CVV START DATE	CVV DATA DE IN斤IO	CVV FECHA INICIO	CVV DATE DE DEBUT
89	ISSUE NUMBER	N筋ERO DE EMISS斤	NUMERO DE EMISION	NO DEMISSION
90	START DATE (MMYY)	DATA DE IN斤IO (AAMM)	FECHA INICIO (AAMM)	DATE DE DEBUT-AAMM

```
@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
@param timeout Timout value, in seconds


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

12.4.4.36 int pin_promptForKeyInput (int *messageID,* int *languageID,* int *maskInput,* int *minLen,* int *maxLen,* int *timeout*)

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	N筋ERO PO	NUMERO PO	No COMMANDE
8	DRIVER ID	LICEN斤墩	LICENCIA	ID CONDUCTEUR
9	ODOMETER	ODOMETER	ODOMETRO	ODOMETRE
10	ID NUMBER	N筋ERO 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 N筋ERO	NUMERO EMP	No TRAVAIL
14	WORK ORDER	TRABALHO ORDEM	ORDEN TRABAJO	FICHE TRAVAIL

15	VEHICLE ID	ID VE斤ULO	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	N筋ERO PATH	RUTA NUMERO	No ROUTE
23	ENTER USER ID	ENTER USER ID	INGRESE ID USUARIO	ID UTILISATEUR
24	FLEET NUMBER	N筋ERO DE FROTA	FLOTA NUMERO	No PARC AUTO
25	ENTER PRODUCT	ADICIONAR PRODUTO	INGRESE PRODUCTO	ENTREZ PRODUIT
26	DRIVER NUMBER	N筋ERO DRIVER	CONDUCTOR NUMERO	No CONDUCTEUR
27	ENTER LICENSE	ENTER LICEN斤墩	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 VE斤ULO	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	N筋ERO DO VE斤ULO	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 USU筋粯IO	INGRESE INFO USUARIO	INFO UTILISATEUR
39	ENTER CUST CODE	ENTER CODE. CLIENTE	INGRESE COD. CLIENTE	ENTR CODE CLIENT
40	ENTER EMPLOYEE	ENTER FUNCION筋粯IO	INGRESE EMPLEADO	ENTREZ EMPLOYE
41	ENTER ID NUMBER	ENTER N筋ERO 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	N筋ERO 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 VE斤ULO	REINGRESE VEHICULO	RE-ENTR VEHICULE
48	ENTER VEHICLE ID	ENTER VE斤ULO 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 REFER斤憢CIA	INGRESE REFERENCIA	ENTREZ REFERENCE
53	ENTER AUTH NUMBR	ENTER N筋ERO 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	QUILOMETRAGE-M	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 #	VE斤ULO #	VEHICULO #	# VEHICULE
68	ENTER VEHICLE #	ENTER VE斤ULO #	INGRESE VEHICULO #	ENT # VEHICULE

69	JOB #	TRABALHO #	TRABAJO #	# TRAVAIL
70	ENTER JOB #	ENTER JOB #	INGRESE TRABAJO #	ENTREZ # TRAVAIL
71	DEPT NUMBER	N筋ERO DEPT	NUMERO DEPTO	No DEPARTEMENT
72	DEPARTMENT #	DEPARTAMENTO #	DEPARTAMENTO #	DEPARTEMENT
73	ENTER DEPT #	ENTER DEPT #	INGRESE DEPTO #	ENT# DEPARTEMENT
74	LICENSE NUMBER	N筋ERO DE LICEN斤墩	NUMERO LICENCIA	No PERMIS
75	LICENSE #	LICEN斤墩 #	LICENCIA #	# PERMIS
76	ENTER LICENSE #	ENTER LICEN斤墩 #	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 N筋ERO DE CART斤	INGRESE NUM TARJETA	ENTREZ NO CARTE
86	EXP DATE(YYMM)	VALIDADE VAL (AAMM)	FECHA EXP (AAMM)	DATE EXPIR(AAMM)
87	PHONE NUMBER	TELEFONE	NUMERO TELEFONO	NO TEL
88	CVV START DATE	CVV DATA DE IN斤IO	CVV FECHA INICIO	CVV DATE DE DEBUT
89	ISSUE NUMBER	N筋ERO DE EMISS斤	NUMERO DE EMISION	NO DEMISSION
90	START DATE (MMYY)	DATA DE IN斤IO (AAMM)	FECHA INICIO (AAMM)	DATE DE DEBUT-AAMM

```
@param messageID  Message (1-90)
@param languageID 0=English Prompt, 1=Portuguese Prompt, 2=Spanish Prompt, 3=French Prompt
@param maskInput  1 = entry is masked with '*', 0 = 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
@param timeout Timout value, in seconds


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

12.4.4.37 void pin_registerCallBk (pPIN_callBack *pPINf*)

To register the pin callback function to get the PINPad data. (Pass NULL to disable the callback.)

12.4.4.38 int pin_sendBeep (int *frequency,* int *duration*)

Send Beep

Executes a beep request.

Parameters

frequency	Frequency, range 200-20000Hz Not used for NEO 2 devices
duration	Duration, range 16-65535ms Not used for NEO 2 devices

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.4.4.39 int pin_setKeyValues (int *mode*)

Set Key Values

Set return key values on or off

Parameters

mode	On: 1, Off: 0

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.4.4.40 void registerHotplugCallBk (pMessageHotplug *pMsgHotplug*)

To register the USB HID hot-plug callback function which implemented in the application to monitor the hotplug message from the SDK.

12.4.4.41 void registerLogCallBk (pSendDataLog *pFSend,* pReadDataLog *pFRead*)

To register the log callback function which implemented in the application to monitor sending/reading data between application and reader.

12.4.4.42 char* SDK_Version ( )

To Get SDK version

Returns

return the SDK version string

12.4.4.43 int setAbsoluteLibraryPath (const char * *absoluteLibraryPath*)

Set the path to use when searching for ID TECH's libraries. If this is not set, the libraries will be searched for with the system's default procedures.

Parameters

absoluteLibrary-Path	The absolute path to ID TECH's libraries.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.5 Source_C/libIDT_MiniSmartII.h File Reference

MiniSmartII API.

```
#include "IDTDef.h"
```

Macros

- #define IN
- #define OUT
- #define IN_OUT

Typedefs

- typedef void(∗ pMessageHotplug)(int, int)
- typedef void(∗ pSendDataLog)(unsigned char ∗, int)
- typedef void(∗ pReadDataLog)(unsigned char ∗, int)
- typedef void(∗ pEMV_callBack)(int, int, unsigned char ∗, int, IDTTransactionData ∗, EMV_Callback ∗, int)
- typedef void(∗ pMSR_callBack)(int, IDTMSRData)
- typedef void(∗ pMSR_callBackp)(int, IDTMSRData ∗)
- typedef void(∗ pPIN_callBack)(int, IDTPINData ∗)
- typedef void(∗ pCMR_callBack)(int, IDTCMRData ∗)
- typedef void(∗ pCSFS_callBack)(BYTE status)
- typedef void(∗ ftpComm_callBack)(int, int, int)
- typedef void(∗ httpComm_callBack)(BYTE ∗, int)
- typedef void(∗ v4Comm_callBack)(BYTE, BYTE, BYTE ∗, int)

Functions

- void registerHotplugCallBk (pMessageHotplug pMsgHotplug)
- void registerLogCallBk (pSendDataLog pFSend, pReadDataLog pFRead)
- void emv_registerCallBk (pEMV_callBack pEMVf)
- void msr_registerCallBk (pMSR_callBack pMSRf)
- void msr_registerCallBkp (pMSR_callBackp pMSRf)
- void pin_registerCallBk (pPIN_callBack pPINf)
- void device_registerCameraCallBk (pCMR_callBack pCMRf)
- void device_registerCardStatusFrontSwitchCallBk (pCSFS_callBack pCSFSf)
- void comm_registerHTTPCallback (httpComm_callBack cBack)
- void comm_registerV4Callback (v4Comm_callBack cBack)
- char ∗ SDK_Version ()
- int setAbsoluteLibraryPath (const char ∗absoluteLibraryPath)
- int device_init ()
- int rs232_device_init (int deviceType, int port_number, int brate)
- int device_setCurrentDevice (int deviceType)
- int device_close ()
- void device_getResponseCodeString (IN int returnCode, OUT char ∗despcrition)
- int device_isConnected ()
- int device_isAttached (int deviceType)
- int device_getFirmwareVersion (OUT char ∗firmwareVersion)
- int device_getFirmwareVersion_Len (OUT char ∗firmwareVersion, IN_OUT int ∗firmwareVersionLen)
- int device_getCurrentDeviceType ()
- int device_SendDataCommand (IN BYTE ∗cmd, IN int cmdLen, IN BYTE ∗data, IN int dataLen, OUT BYTE ∗response, IN_OUT int ∗respLen)

- int device_updateFirmware (IN BYTE *firmwareData, IN int firmwareDataLen, IN char *firmwareName, IN int encryptionType, IN BYTE *keyBlob, IN int keyBlobLen)
- int device_rebootDevice ()
- int device_controlLED (byte indexLED, byte control, int intervalOn, int intervalOff)
- int device_controlLED_ICC (int controlMode, int interval)
- int device_controlLED_MSR (byte control, int intervalOn, int intervalOff)
- int device_controlBeep (int index, int frequency, int duration)
- int device_getKeyStatus (int *newFormat, BYTE *status, int *statusLen)
- int device_getSDKWaitTime ()
- void device_setSDKWaitTime (int waitTime)
- int device_getThreadStackSize ()
- void device_setThreadStackSize (int threadSize)
- int config_getModelNumber (OUT char *sNumber)
- int config_getModelNumber_Len (OUT char *sNumber, IN_OUT int *sNumberLen)
- int config_getSerialNumber (OUT char *sNumber)
- int config_getSerialNumber_Len (OUT char *sNumber, IN_OUT int *sNumberLen)
- int config_setLEDController (int firmwareControlMSRLED, int firmwareControlICCLED)
- int config_getLEDController (int *firmwareControlMSRLED, int *firmwareControlICCLED)
- int config_setBeeperController (int firmwareControlBeeper)
- int config_getBeeperController (int *firmwareControlBeeper)
- int config_setEncryptionControl (int msr, int icc)
- int config_getEncryptionControl (int *msr, int *icc)
- int icc_enable (IN int withNotification)
- int icc_disable ()
- int icc_powerOnICC (OUT BYTE *ATR, IN_OUT int *inLen)
- int icc_powerOffICC ()
- int icc_exchangeAPDU (IN BYTE *c_APDU, IN int cLen, OUT BYTE *reData, IN_OUT int *reLen)
- int icc_exchangeEncryptedAPDU (IN BYTE *c_APDU, IN int cLen, OUT BYTE *reData, IN_OUT int *reLen)
- int icc_getAPDU_KSN (OUT BYTE *KSN, IN_OUT int *inLen)
- int icc_getFunctionStatus (OUT int *enabled, OUT int *withNotification)
- int icc_getICCReaderStatus (OUT BYTE *status)
- int icc_getKeyFormatForICCDUKPT (OUT BYTE *format)
- int icc_getKeyTypeForICCDUKPT (OUT BYTE *type)
- int emv_getEMVKernelVersion (OUT char *version)
- int emv_getEMVKernelVersion_Len (OUT char *version, IN_OUT int *versionLen)
- int emv_getEMVKernelCheckValue (OUT BYTE *checkValue, IN_OUT int *checkValueLen)
- int emv_getEMVConfigurationCheckValue (OUT BYTE *checkValue, IN_OUT int *checkValueLen)
- void emv_setAutoAuthenticateTransaction (IN int authenticate)
- void emv_setAutoCompleteTransaction (IN int complete)
- int emv_getAutoAuthenticateTransaction ()
- int emv_getAutoCompleteTransaction ()
- void emv_allowFallback (IN int allow)
- int emv_startTransaction (IN double amount, IN double amtOther, IN int exponent, IN int type, IN int timeout, IN BYTE *tags, IN int tagsLen, IN int forceOnline)
- int emv_activateTransaction (IN int timeout, IN BYTE *tags, IN int tagsLen, IN int forceOnline)
- int emv_authenticateTransaction (IN BYTE *updatedTLV, IN int updatedTLVLen)
- int emv_authenticateTransactionWithTimeout (IN int timeout, IN BYTE *updatedTLV, IN int updatedTLVLen)
- int emv_completeTransaction (IN int commError, IN BYTE *authCode, IN int authCodeLen, IN BYTE *iad, IN int iadLen, IN BYTE *tlvScripts, IN int tlvScriptsLen, IN BYTE *tlv, IN int tlvLen)
- int emv_cancelTransaction ()
- int emv_retrieveTransactionResult (IN BYTE *tags, IN int tagsLen, IDTTransactionData *cardData)
- int emv_callbackResponseLCD (IN int type, byte selection)
- int emv_callbackResponseMSR (IN BYTE *MSR, IN_OUT int MSRLen)
- int emv_retrieveApplicationData (IN BYTE *AID, IN int AIDLen, OUT BYTE *tlv, IN_OUT int *tlvLen)
- int emv_setApplicationData (IN BYTE *name, IN int nameLen, IN BYTE *tlv, IN int tlvLen)

- int emv_removeApplicationData (IN BYTE *AID, IN int AIDLen)
- int emv_removeAllApplicationData ()
- int emv_retrieveAIDList (OUT BYTE *AIDList, IN_OUT int *AIDListLen)
- int emv_retrieveTerminalData (OUT BYTE *tlv, IN_OUT int *tlvLen)
- int emv_setTerminalData (IN BYTE *tlv, IN int tlvLen)
- int emv_removeTerminalData ()
- int emv_retrieveCAPK (IN BYTE *capk, IN int capkLen, OUT BYTE *key, IN_OUT int *keyLen)
- int emv_setCAPK (IN BYTE *capk, IN int capkLen)
- int emv_removeCAPK (IN BYTE *capk, IN int capkLen)
- int emv_removeAllCAPK ()
- int emv_retrieveCAPKList (OUT BYTE *keys, IN_OUT int *keysLen)
- int emv_retrieveTerminalID (OUT char *terminalID)
- int emv_retrieveTerminalID_Len (OUT char *terminalID, IN_OUT int *terminalIDLen)
- int emv_setTerminalID (IN char *terminalID)
- int emv_retrieveCRL (OUT BYTE *list, IN_OUT int *lssLen)
- int emv_setCRL (IN BYTE *list, IN int lsLen)
- int emv_removeCRL (IN BYTE *list, IN int lsLen)
- int emv_removeAllCRL ()

12.5.1 Detailed Description

MiniSmartII API. MiniSmartII Global API methods.

12.5.2 Macro Definition Documentation

12.5.2.1 #define IN

INPUT parameter.

12.5.2.2 #define IN_OUT

INPUT / OUTPUT PARAMETER.

12.5.2.3 #define OUT

OUTPUT parameter.

12.5.3 Typedef Documentation

12.5.3.1 typedef void(* ftpComm_callBack)(int, int, int)

Define the comm callback function to get FTP file transfer status

It should be passed as a parameter in a FTP request, Signature (int, int, int) = response code, current block, total blocks RESPONSE CODES: 100 = FILE DOWNLOAD STARTED 101 = FILE BLOCK XX OF XX RECEIVED 102 = FILE DOWNLOAD COMPLETED 103 = FILE DOWNLOAD TERMINATED PREMATURELY

12.5.3.2 typedef void(* httpComm_callBack)(BYTE *, int)

Define the comm callback function to get the async url data

It should be registered using the comm_registerHTTPCallback

12.5.3.3 typedef void(* pCMR_callBack)(int, IDTCMRData *)

Define the camera callback function to get the image data

It should be registered using the device_registerCameraCallBk,

12.5.3.4 typedef void(* pCSFS_callBack)(BYTE status)

Define the card status and front switch callback function to get card and front switch status

It should be registered using the device_registerCardStatusFrontSwitchCallBk,

12.5.3.5 typedef void(* pEMV_callBack)(int, int, unsigned char *, int, IDTTransactionData *, EMV_Callback *, int)

Define the EMV callback function to get the transaction message/data/result.

It should be registered using the emv_registerCallBk,

12.5.3.6 typedef void(* pMessageHotplug)(int, int)

Define the USB hot-plug callback function to monitor the info when plug in/out the reader.

It should be registered using the registerHotplugCallBk, The first integer parameter is device type, and the second integer parameter is either 0: Device Plugged Out or 1: Device Plugged In

12.5.3.7 typedef void(* pMSR_callBack)(int, IDTMSRData)

Define the MSR callback function to get the MSR card data

It should be registered using the msr_registerCallBk, this callback function is for backward compatibility

12.5.3.8 typedef void(* pMSR_callBackp)(int, IDTMSRData *)

Define the MSR callback function to get pointer to the MSR card data

It should be registered using the msr_registerCallBk, this callback function is recommended instead of pMSR_callBack

12.5.3.9 typedef void(* pPIN_callBack)(int, IDTPINData *)

Define the PINPad callback function to get the input PIN Pad data

It should be registered using the pin_registerCallBk,

12.5.3.10 typedef void(* pReadDataLog)(unsigned char *, int)

Define the read response callback function to monitor the reading response from the reader.

It should be registered using the registerLogCallBk,

12.5.3.11 typedef void(* pSendDataLog)(unsigned char *, int)

Define the send command callback function to monitor the sending command into the reader.

It should be registered using the registerLogCallBk,

12.5.3.12 typedef void(∗ v4Comm_callBack)(BYTE, BYTE, BYTE ∗, int)

Define the comm callback function to receive the V4 Protocol packets received by the device from an external source (IP/USB/RS-232) It should be registered using the comm_registerV4Callback, Data callback will contain command, sub-command, and data from V4 packet

12.5.4 Function Documentation

12.5.4.1 void comm_registerHTTPCallback (httpComm_callBack *cBack*)

Register Comm HTTP Async Callback

Parameters

cBack	- HTTP Comm callback

12.5.4.2 void comm_registerV4Callback (v4Comm_callBack *cBack*)

Register External V4 Protocol commands Callback

Parameters

cBack	- V4 Protocol Comm callback

12.5.4.3 int config_getBeeperController (int ∗ *firmwareControlBeeper*)

Get the Beeper Controller Status Set the Beeper controlled Status by software or firmware

Parameters

firmwareControl-Beeper	1 means firmware control the beeper, 0 means software control beeper.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.5.4.4 int config_getEncryptionControl (int ∗ *msr,* int ∗ *icc*)

Get Encryption Control

Get Encryption Control to switch status between MSR and ICC/EMV function. Following Encryption status supported:

- MSR ON, ICC/EMV ON,
- MSR ON, ICC/EMV OFF,
- MSR OFF, ICC/EMV OFF,

Parameters

msr	• 1: enabled MSR with Encryption, • 0: disabled MSR with Encryption,
icc	• 1: enabled ICC with Encryption, • 0: disabled ICC with Encryption,

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.5.4.5 int config_getLEDController (int * *firmwareControlMSRLED,* int * *firmwareControlICCLED*)

Get the LED Controller Status Get the MSR / ICC LED controlled status by software or firmware NOTE: The ICC LED always controlled by software.

Parameters

firmwareControl-MSRLED	• 1: firmware control the MSR LED • 0: software control the MSR LED
firmwareControl-ICCLED	• 1: firmware control the ICC LED • 0: software control the ICC LED

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.5.4.6 int config_getModelNumber (OUT char * *sNumber*)

DEPRECATED : please use config_getModelNumber_Len(OUT char* sNumber, IN_OUT int *sNumberLen)

Polls device for Model Number

Parameters

sNumber	Returns Model Number; needs to have at least 64 bytes of memory

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.5.4.7 int config_getModelNumber_Len (OUT char * *sNumber,* IN_OUT int * *sNumberLen*)

Polls device for Model Number

Parameters

sNumber	Returns Model Number
sNumber	length of Model Number

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.5.4.8 int config_getSerialNumber (OUT char * *sNumber*)

DEPRECATED : please use config_getSerialNumber_Len(OUT char* sNumber, IN_OUT int *sNumberLen)

Polls device for Serial Number

Parameters

sNumber	Returns Serial Number; needs to have at least 64 bytes of memory

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.5.4.9 int config_getSerialNumber_Len (OUT char * *sNumber,* IN_OUT int * *sNumberLen*)

Polls device for Serial Number

Parameters

sNumber	Returns Serial Number
sNumberLen	Length of Serial Number

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.5.4.10 int config_setBeeperController (int *firmwareControlBeeper*)

Set the Beeper Controller Set the Beeper controlled by software or firmware

Parameters

firmwareControl-Beeper	1 means firmware control the beeper, 0 means software control beeper.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.5.4.11 int config_setEncryptionControl (int *msr,* int *icc*)

Set Encryption Control

Set Encryption Control to switch status between MSR and ICC/EMV function. Following Encryption status supported:

- MSR ON, ICC/EMV ON,

- MSR ON, ICC/EMV OFF,
- MSR OFF, ICC/EMV OFF,

Parameters

msr	• 1: enable MSR with Encryption, • 0: disable MSR with Encryption,
icc	• 1: enable ICC with Encryption, • 0: disable ICC with Encryption,

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.5.4.12 int config_setLEDController (int *firmwareControlMSRLED,* int *firmwareControlICCLED*)

Set the LED Controller Set the MSR / ICC LED controlled by software or firmware NOTE: The ICC LED always controlled by software.

Parameters

firmwareControl-MSRLED	• 1: firmware control the MSR LED • 0: software control the MSR LED
firmwareControl-ICCLED	• 1: firmware control the ICC LED • 0: software control the ICC LED

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.5.4.13 int device_close ( )

Close the device

Returns

RETURN_CODE: 0: success, 0x0A: failed

12.5.4.14 int device_controlBeep (int *index,* int *frequency,* int *duration*)

Control Beep

Controls the Beeper

Parameters

index	For Augusta, must be set to 1 (only one beeper)
frequency	Frequency, range 1000-20000 (suggest minimum 3000)
duration	Duration, in milliseconds (range 1 - 65525)

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.5.4.15 int device_controlLED (byte *indexLED,* byte *control,* int *intervalOn,* int *intervalOff*)

Control MSR LED

Controls the LED for the MSR

Parameters

indexLED	For Augusta, must be set to 1 (MSR LED)
control	LED Status: • 00: OFF • 01: RED Solid • 02: RED Blink • 11: GREEN Solid • 12: GREEN Blink • 21: BLUE Solid • 22: BLUE Blink
intervalOn	Blink interval ON, in ms (Range 200 - 2000)
intervalOff	Blink interval OFF, in ms (Range 200 - 2000)

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.5.4.16 int device_controlLED_ICC (int *controlMode,* int *interval*)

Control ICC LED

Controls the LED for the ICC card slot

Parameters

controlMode	0 = off, 1 = solid, 2 = blink
interval	Blink interval, in ms (500 = 500 ms)

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.5.4.17 int device_controlLED_MSR (byte *control,* int *intervalOn,* int *intervalOff*)

Control the MSR LED

Controls the MSR / ICC LED This API not recommended to control ICC LED

Parameters

control	• 0x00 = off, • 0x01 = RED Solid, • 0x02 = RED Blink, • 0x11 = GREEN Solid, • 0x12 = GREEN Blink, • 0x21 = BLUE Solid, • 0x22 = BLUE Blink,
intervalOn	Blink interval on time last, in ms (500 = 500 ms, valid from 200 to 2000)
intervalOff	Blink interval off time last, in ms (500 = 500 ms, valid from 200 to 2000)

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.5.4.18 int device_getCurrentDeviceType ( )

Get current active device type

Returns

: return the device type defined as DEVICE_TYPE in the IDTDef.h

12.5.4.19 int device_getFirmwareVersion (OUT char * *firmwareVersion*)

DEPRECATED : please use device_getFirmwareVersion_Len(OUT char* firmwareVersion, IN_OUT int *firmware-VersionLen)

Polls device for Firmware Version

Parameters

firmwareVersion	Response returned of Firmware Version; needs to have at least 128 bytes of memory

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.5.4.20 int device_getFirmwareVersion_Len (OUT char * *firmwareVersion,* IN_OUT int * *firmwareVersionLen*)

Polls device for Firmware Version

Parameters

firmwareVersion	Response returned of Firmware Version

firmwareVersion-Len	Length of Firmware Version

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.5.4.21 int device_getKeyStatus (int * *newFormat,* BYTE * *status,* int * *statusLen*)

Get Key Status

Gets the status of loaded keys

Parameters

status	newFormat for Augusta and miniSmartII only 1: new format of key status 0: reserved format for support previous device
status	For L100, Augusta and miniSmartII: When the newFormat is 0, data format as follows. For Augusta and miniSmartII: byte 0: PIN DUKPT Key, Does not support, always 0 byte 1: PIN Master Key, Does not support, always 0 byte 2: PIN Session Key, Does not support, always 0 byte 3: Account/MSR DUKPT Key, 1 Exists, 0 None, 0xFF STOP byte 4: Account/ICC DUKPT Key, 1 Exists, 0 None, 0xFF STOP byte 5: Admin DUKPT Key, 1 Exists, 0 None, 0xFF STOP For L100: byte 0: PIN DUKPT Key byte 1: PIN Master Key byte 2: Standard PIN Session Key byte 3: Desjardins PIN Session Key byte 4: Account/MSR DUKPT Key, 1 Exists, 0 None, 0xFF STOP, Does not support, always 0 byte 5: Account/ICC DUKPT Key, 1 Exists, 0 None, 0xFF STOP, Does not support, always 0 byte 6: Admin DUKPT Key, 1 Exists, 0 None, 0xFF STOP byte 7: Data DUKPT Key, 1 Exists, 0 None, 0xFF STOP byte 8: MAC DUKPT Key, 1 Exists, 0 None, 0xFF STOP

when the newFormat is 1, data format as follows. [Block Length] [KeyStatusBlock1] [KeyStatusBlock2]...[KeyStatusBlockN] Where: [Block Length] is 2 bytes, format is Len_L Len_H, is KeyStatusBlock Number [KeyStatusBlockX> is 4 bytes, format is [Key Index and Key Name] [key slot] [key status]: [Key Index and Key Name] is 1 byte. Please refer to following table 0x14 LCL-KEK to Encrypt Other Keys 0x02 Data encryption Key to Encrypt ICC/MSR 0x05 MAC DUKPT Key for Host-Device - MAC Verification 0x05 MTK DUKPT Key for TTK Self-Test 0x0C RKI-KEK for Remote Key Injection [key slot] is 2 bytes. Range is 0 - 9999 the MTK DUKPT Key slot is 16, the others are all 0 [key status] is 1 byte. 0 - Not Exist 1 - Exist 0xFF - (Stop. Only Valid for DUKPT Key) For NEO2 and SREDKey2: Each unit of three bytes represents one key's parameters (index and slot). Key Name Index (1 byte): 0x14 - LCL-KEK 0x01 - Pin encryption Key (NEO2 only) 0x02 - Data encryption Key 0x05 - MAC DUKPT Key 0x0A - PCI Pairing Key (NEO2 only) Key Slot (2 bytes): Indicate different slots of a certain Key Name Example: slot =5 (0x00 0x05), slot=300 (0x01 0x2C) For BTPay380, slot is always 0 For example, 0x14 0x00 0x00 0x02 0x00 0x00 0x0A 0x00 0x00 will represent [KeyNameIndex=0x14,KeySlot=0x0000], [KeyNameIndex=0x02,KeySlot=0x0000] and [KeyNameIndex=0x0A,KeySlot=0x0000]

Parameters

statusLen	the length of status

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.5.4.22 void device_getResponseCodeString (IN int *returnCode,* OUT char * *despcrition*)

Review the return code description.

Parameters

returnCode	the response result.
description	

Return values

the	string for description of response result

- 0: "no error, beginning task";
- 1: "no response from reader";
- 2: "invalid response data";
- 3: "time out for task or CMD";
- 4: "wrong parameter";
- 5: "SDK is doing MSR or ICC task";
- 6: "SDK is doing PINPad task";
- 7: "SDK is doing CTLS task";
- 8: "SDK is doing EMV task";
- 9: "SDK is doing Other task";
- 10: "err response or data";
- 11: "no reader attached";
- 12: "mono audio is enabled";
- 13: "did connection";
- 14: "audio volume is too low";
- 15: "task or CMD be canceled";
- 16: "UF wrong string format";
- 17: "UF file not found";
- 18: "UF wrong file format";
- 19: "Attempt to contact online host failed";
- 20: "Attempt to perform RKI failed";
- 22: "Buffer size is not enough";
- 0X300: "Key Type(TDES) of Session Key is not same as the related Master Key.";
- 0X400: "Related Key was not loaded.";
- 0X500: "Key Same.";
- 0X501: "Key is all zero";
- 0X502: "TR-31 format error";
- 0X702: "PAN is Error Key.";
- 0X705: "No Internal MSR PAN (or Internal MSR PAN is erased timeout)";
- 0X0C01: "Incorrect Frame Tag";

- 0X0C02: "Incorrect Frame Type";
- 0X0C03: "Unknown Frame Type";
- 0X0C04: "Unknown Command";
- 0X0C05: "Unknown Sub-Command";
- 0X0C06: "CRC Error";
- 0X0C07: "Failed";
- 0X0C08: "Timeout";
- 0X0C0A: "Incorrect Parameter";
- 0X0C0B: "Command Not Supported";
- 0X0C0C: "Sub-Command Not Supported";
- 0X0C0D: "Parameter Not Supported / Status Abort Command";
- 0X0C0F: "Sub-Command Not Allowed";
- 0X0D01: "Incorrect Header Tag";
- 0X0D02: "Unknown Command";
- 0X0D03: "Unknown Sub-Command";
- 0X0D04: "CRC Error in Frame";
- 0X0D05: "Incorrect Parameter";
- 0X0D06: "Parameter Not Supported";
- 0X0D07: "Mal-formatted Data";
- 0X0D08: "Timeout";
- 0X0D0A: "Failed / NACK";
- 0X0D0B: "Command not Allowed";
- 0X0D0C: "Sub-Command not Allowed";
- 0X0D0D: "Buffer Overflow (Data Length too large for reader buffer)";
- 0X0D0E: "User Interface Event";
- 0X0D11: "Communication type not supported, VT-1, burst, etc.";
- 0X0D12: "Secure interface is not functional or is in an intermediate state.";
- 0X0D13: "Data field is not mod 8";
- 0X0D14: "Pad - 0X80 not found where expected";
- 0X0D15: "Specified key type is invalid";
- 0X0D1: "Could not retrieve key from the SAM(InitSecureComm)";
- 0X0D17: "Hash code problem";
- 0X0D18: "Could not store the key into the SAM(InstallKey)";
- 0X0D19: "Frame is too large";
- 0X0D1A: "Unit powered up in authentication state but POS must resend the InitSecureComm command";
- 0X0D1B: "The EEPROM may not be initialized because SecCommInterface does not make sense";

- 0X0D1C: "Problem encoding APDU";
- 0X0D20: "Unsupported Index(ILM) SAM Transceiver error - problem communicating with the SAM(Key Mgr)";
- 0X0D2: "Unexpected Sequence Counter in multiple frames for single bitmap(ILM) Length error in data returned from the SAM(Key Mgr)";
- 0X0D22: "Improper bit map(ILM)";
- 0X0D23: "Request Online Authorization";
- 0X0D24: "ViVOCard3 raw data read successful";
- 0X0D25: "Message index not available(ILM) ViVOcomm activate transaction card type(ViVOcomm)";
- 0X0D26: "Version Information Mismatch(ILM)";
- 0X0D27: "Not sending commands in correct index message index(ILM)";
- 0X0D28: "Time out or next expected message not received(ILM)";
- 0X0D29: "ILM languages not available for viewing(ILM)";
- 0X0D2A: "Other language not supported(ILM)";
- 0X0D41: "Unknown Error from SAM";
- 0X0D42: "Invalid data detected by SAM";
- 0X0D43: "Incomplete data detected by SAM";
- 0X0D44: "Reserved";
- 0X0D45: "Invalid key hash algorithm";
- 0X0D46: "Invalid key encryption algorithm";
- 0X0D47: "Invalid modulus length";
- 0X0D48: "Invalid exponent";
- 0X0D49: "Key already exists";
- 0X0D4A: "No space for new RID";
- 0X0D4B: "Key not found";
- 0X0D4C: "Crypto not responding";
- 0X0D4D: "Crypto communication error";
- 0X0D4E: "Module-specific error for Key Manager";
- 0X0D4F: "All key slots are full (maximum number of keys has been installed)";
- 0X0D50: "Auto-Switch OK";
- 0X0D51: "Auto-Switch failed";
- 0X0D90: "Account DUKPT Key not exist";
- 0X0D91: "Account DUKPT Key KSN exausted";
- 0X0D00: "This Key had been loaded.";
- 0X0E00: "Base Time was loaded.";
- 0X0F00: "Encryption Or Decryption Failed.";
- 0X1000: "Battery Low Warning (It is High Priority Response while Battery is Low.)";

- 0X1800: "Send 'Cancel Command' after send 'Get Encrypted PIN' & 'Get Numeric' & 'Get Amount'; - 0X1900: "Press 'Cancel' key after send 'Get Encrypted PIN' & 'Get Numeric' & 'Get Amount';
- 0X30FF: "Security Chip is not connect";
- 0X3000: "Security Chip is deactivation & Device is In Removal Legally State.";
- 0X3101: "Security Chip is activation & Device is In Removal Legally State.";
- 0X5500: "No Admin DUKPT Key.";
- 0X5501: "Admin DUKPT Key STOP.";
- 0X5502: "Admin DUKPT Key KSN is Error.";
- 0X5503: "Get Authentication Code1 Failed.";
- 0X5504: "Validate Authentication Code Error.";
- 0X5505: "Encrypt or Decrypt data failed.";
- 0X5506: "Not Support the New Key Type.";
- 0X5507: "New Key Index is Error.";
- 0X5508: "Step Error.";
- 0X5509: "KSN Error";
- 0X550A: "MAC Error.";
- 0X550B: "Key Usage Error.";
- 0X550C: "Mode Of Use Error.";
- 0X550F: "Other Error.";
- 0X6000: "Save or Config Failed / Or Read Config Error.";
- 0X6200: "No Serial Number.";
- 0X6900: "Invalid Command - Protocol is right, but task ID is invalid.";
- 0X6A01: "Unsupported Command - Protocol and task ID are right, but command is invalid - In this State";
- 0X6A00: "Unsupported Command - Protocol and task ID are right, but command is invalid.";
- 0X6B00: "Unknown parameter in command - Protocol task ID and command are right, but parameter is invalid.";
- 0X6C00: "Unknown parameter in command - Protocol task ID and command are right, but length is out of the requirement.";
- 0X7200: "Device is suspend (MKSK suspend or press password suspend).";
- 0X7300: "PIN DUKPT is STOP (21 bit 1).";
- 0X7400: "Device is Busy.";
- 0XE100: "Can not enter sleep mode";
- 0XE200: "File has existed";
- 0XE300: "File has not existed";
- 0XE313: "IO line low -- Card error after session start";
- 0XE400: "Open File Error";
- 0XE500: "SmartCard Error";

- 0XE600: "Get MSR Card data is error";
- 0XE700: "Command time out";
- 0XE800: "File read or write is error";
- 0XE900: "Active 1850 error!";
- 0XEA00: "Load bootloader error";
- 0XEF00: "Protocol Error- STX or ETX or check error.";
- 0XEB00: "Picture is not exist";
- 0X2C02: "No Microprocessor ICC seated";
- 0X2C06: "no card seated to request ATR";
- 0X2D01: "Card Not Supported,";
- 0X2D03: "Card Not Supported, wants CRC";
- 0X690D: "Command not supported on reader without ICC support";
- 0X8100: "ICC error time out on power-up";
- 0X8200: "invalid TS character received - Wrong operation step";
- 0X8300: "Decode MSR Error";
- 0X8400: "TriMagII no Response";
- 0X8500: "No Swipe MSR Card";
- 0X8510: "No Financial Card";
- 0X8600: "Unsupported F, D, or combination of F and D";
- 0X8700: "protocol not supported EMV TD1 out of range";
- 0X8800: "power not at proper level";
- 0X8900: "ATR length too long";
- 0X8B01: "EMV invalid TA1 byte value";
- 0X8B02: "EMV TB1 required";
- 0X8B03: "EMV Unsupported TB1 only 00 allowed";
- 0X8B04: "EMV Card Error, invalid BWI or CWI";
- 0X8B06: "EMV TB2 not allowed in ATR";
- 0X8B07: "EMV TC2 out of range";
- 0X8B08: "EMV TC2 out of range";
- 0X8B09: "per EMV96 TA3 must be $>$ - 0XF";
- 0X8B10: "ICC error on power-up";
- 0X8B11: "EMV T=1 then TB3 required";
- 0X8B12: "Card Error, invalid BWI or CWI";
- 0X8B13: "Card Error, invalid BWI or CWI";
- 0X8B17: "EMV TC1/TB3 conflict-";
- 0X8B20: "EMV TD2 out of range must be T=1";

- 0X8C00: "TCK error";
- 0XA304: "connector has no voltage setting";
- 0XA305: "ICC error on power-up invalid (SBLK(IFSD) exchange";
- 0XE301: "ICC error after session start";
- 0XFF00: "Request to go online";
- 0XFF01: "EMV: Accept the offline transaction";
- 0XFF02: "EMV: Decline the offline transaction";
- 0XFF03: "EMV: Accept the online transaction";
- 0XFF04: "EMV: Decline the online transaction";
- 0XFF05: "EMV: Application may fallback to magstripe technology";
- 0XFF06: "EMV: ICC detected tah the conditions of use are not satisfied";
- 0XFF07: "EMV: ICC didn't accept transaction";
- 0XFF08: "EMV: Transaction was cancelled";
- 0XFF09: "EMV: Application was not selected by kernel or ICC format error or ICC missing data error";
- 0XFF0A: "EMV: Transaction is terminated";
- 0XFF0B: "EMV: Other EMV Error";
- 0XFFFF: "NO RESPONSE";
- 0XF002: "ICC communication timeout";
- 0XF003: "ICC communication Error";
- 0XF00F: "ICC Card Seated and Highest Priority, disable MSR work request";
- 0XF200: "AID List / Application Data is not exist";
- 0XF201: "Terminal Data is not exist";
- 0XF202: "TLV format is error";
- 0XF203: "AID List is full";
- 0XF204: "Any CA Key is not exist";
- 0XF205: "CA Key RID is not exist";
- 0XF206: "CA Key Index it not exist";
- 0XF207: "CA Key is full";
- 0XF208: "CA Key Hash Value is Error";
- 0XF209: "Transaction format error";
- 0XF20A: "The command will not be processing";
- 0XF20B: "CRL is not exist";
- 0XF20C: "CRL number exceed max number";
- 0XF20D: "Amount,Other Amount,Trasaction Type are missing";
- 0XF20E: "The Identification of algorithm is mistake";
- 0XF20F: "No Financial Card";

- 0XF210: "In Encrypt Result state, TLV total Length is greater than Max Length";
- 0X1001: "INVALID ARG";
- 0X1002: "FILE_OPEN_FAILED";
- 0X1003: "FILE OPERATION_FAILED";
- 0X2001: "MEMORY_NOT_ENOUGH";
- 0X3002: "SMARTCARD_FAIL";
- 0X3003: "SMARTCARD_INIT_FAILED";
- 0X3004: "FALLBACK_SITUATION";
- 0X3005: "SMARTCARD_ABSENT";
- 0X3006: "SMARTCARD_TIMEOUT";
- 0X3012: "EMV_RESULT_CODE_MSR_CARD_ERROR_FALLBACK";
- 0X5001: "EMV_PARSING_TAGS_FAILED";
- 0X5002: "EMV_DUPLICATE_CARD_DATA_ELEMENT";
- 0X5003: "EMV_DATA_FORMAT_INCORRECT";
- 0X5004: "EMV_NO_TERM_APP";
- 0X5005: "EMV_NO_MATCHING_APP";
- 0X5006: "EMV_MISSING_MANDATORY_OBJECT";
- 0X5007: "EMV_APP_SELECTION_RETRY";
- 0X5008: "EMV_GET_AMOUNT_ERROR";
- 0X5009: "EMV_CARD_REJECTED";
- 0X5010: "EMV_AIP_NOT_RECEIVED";
- 0X5011: "EMV_AFL_NOT_RECEIVED";
- 0X5012: "EMV_AFL_LEN_OUT_OF_RANGE";
- 0X5013: "EMV_SFI_OUT_OF_RANGE";
- 0X5014: "EMV_AFL_INCORRECT";
- 0X5015: "EMV_EXP_DATE_INCORRECT";
- 0X5016: "EMV_EFF_DATE_INCORRECT";
- 0X5017: "EMV_ISS_COD_TBL_OUT_OF_RANGE";
- 0X5018: "EMV_CRYPTOGRAM_TYPE_INCORRECT";
- 0X5019: "EMV_PSE_NOT_SUPPORTED_BY_CARD";
- 0X5020: "EMV_USER_SELECTED_LANGUAGE";
- 0X5021: "EMV_SERVICE_NOT_ALLOWED";
- 0X5022: "EMV_NO_TAG_FOUND";
- 0X5023: "EMV_CARD_BLOCKED";
- 0X5024: "EMV_LEN_INCORRECT";
- 0X5025: "CARD_COM_ERROR";

- 0X5026: "EMV_TSC_NOT_INCREASED";
- 0X5027: "EMV_HASH_INCORRECT";
- 0X5028: "EMV_NO_ARC";
- 0X5029: "EMV_INVALID_ARC";
- 0X5030: "EMV_NO_ONLINE_COMM";
- 0X5031: "TRAN_TYPE_INCORRECT";
- 0X5032: "EMV_APP_NO_SUPPORT";
- 0X5033: "EMV_APP_NOT_SELECT";
- 0X5034: "EMV_LANG_NOT_SELECT";
- 0X5035: "EMV_NO_TERM_DATA";
- 0X5039: "EMV_PIN_ENTRY_TIMEOUT";
- 0X6001: "CVM_TYPE_UNKNOWN";
- 0X6002: "CVM_AIP_NOT_SUPPORTED";
- 0X6003: "CVM_TAG_8E_MISSING";
- 0X6004: "CVM_TAG_8E_FORMAT_ERROR";
- 0X6005: "CVM_CODE_IS_NOT_SUPPORTED";
- 0X6006: "CVM_COND_CODE_IS_NOT_SUPPORTED";
- 0X6007: "NO_MORE_CVM";
- 0X6008: "PIN_BYPASSED_BEFORE";
- 0X7001: "PK_BUFFER_SIZE_TOO_BIG";
- 0X7002: "PK_FILE_WRITE_ERROR";
- 0X7003: "PK_HASH_ERROR";
- 0X8001: "NO_CARD_HOLDER_CONFIRMATION";
- 0X8002: "GET_ONLINE_PIN";
- 0XD000: "Data not exist";
- 0XD001: "Data access error";
- 0XD100: "RID not exist";
- 0XD101: "RID existed";
- 0XD102: "Index not exist";
- 0XD200: "Maximum exceeded";
- 0XD201: "Hash error";
- 0XD205: "System Busy";
- 0X0E01: "Unable to go online";
- 0X0E02: "Technical Issue";
- 0X0E03: "Declined";
- 0X0E04: "Issuer Referral transaction";

- 0X0F01: "Decline the online transaction";
- 0X0F02: "Request to go online";
- 0X0F03: "Transaction is terminated";
- 0X0F05: "Application was not selected by kernel or ICC format error or ICC missing data error";
- 0X0F07: "ICC didn't accept transaction";
- 0X0F0A: "Application may fallback to magstripe technology";
- 0X0F0C: "Transaction was cancelled";
- 0X0F0D: "Timeout";
- 0X0F0F: "Other EMV Error";
- 0X0F10: "Accept the offline transaction";
- 0X0F11: "Decline the offline transaction";
- 0X0F21: "ICC detected tah the conditions of use are not satisfied";
- 0X0F22: "No app were found on card matching terminal configuration";
- 0X0F23: "Terminal file does not exist";
- 0X0F24: "CAPK file does not exist";
- 0X0F25: "CRL Entry does not exist";
- 0X0FFE: "code when blocking is disabled";
- 0X0FFF: "code when command is not applicable on the selected device";
- 0XF005: "ICC Encrypted C-APDU Data Structure Length Error Or Format Error.";
- 0XBBE0: "CM100 Success";
- 0XBBE1: "CM100 Parameter Error";
- 0XBBE2: "CM100 Low Output Buffer";
- 0XBBE3: "CM100 Card Not Found";
- 0XBBE4: "CM100 Collision Card Exists";
- 0XBBE5: "CM100 Too Many Cards Exist";
- 0XBBE6: "CM100 Saved Data Does Not Exist";
- 0XBBE8: "CM100 No Data Available";
- 0XBBE9: "CM100 Invalid CID Returned";
- 0XBBEA: "CM100 Invalid Card Exists";
- 0XBBEC: "CM100 Command Unsupported";
- 0XBBED: "CM100 Error In Command Process";
- 0XBBEE: "CM100 Invalid Command";
- 0X9031: "Unknown command";
- 0X9032: "Wrong parameter (such as the length of the command is incorrect)";
- 0X9038: "Wait (the command couldnt be finished in BWT)";
- 0X9039: "Busy (a previously command has not been finished)";

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

12.5.4.23 int device_getSDKWaitTime ( )

Get SDK Wait Time

Get the SDK wait time for transactions

Returns

SDK wait time in seconds

12.5.4.24 int device_getThreadStackSize ( )

Get Thread Stack Size

Get the stack size setting for newly created threads

Returns

Thread Stack Size

12.5.4.25 int device_init ( )

Initial the device by USB

It will detect the device and trying connect.

The connect status can be checked by device_isConnected().

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.5.4.26 int device_isAttached (int *deviceType*)

Check if the device is attached to the USB port The function device_init() must be called before this function.

Parameters

deviceType,the	device type of the USB device

Returns

1 if the device is attached, or 0 if the device is not attached

12.5.4.27 int device_isConnected ( )

Check the device conntected status

Returns

DEVICE_DISCONNECT=0, or DEVICE_CONNECTED = 1

12.5.4.28 int 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()

12.5.4.29 void device_registerCameraCallBk (pCMR_callBack *pCMRf*)

To register the camera callback function to get the image data. (Pass NULL to disable the callback.)

12.5.4.30 void device_registerCardStatusFrontSwitchCallBk (pCSFS_callBack *pCSFSf*)

To register the card status and front switch callback function to get status. (Pass NULL to disable the callback.)

12.5.4.31 int device_SendDataCommand (IN BYTE * *cmd*, IN int *cmdLen*, IN BYTE * *data*, IN int *dataLen*, OUT BYTE * *response*, IN_OUT int * *respLen*)

Send a Command to device

Sends a command to the device .

Parameters

cmd	buffer of command to execute.
cmdLen,the	length of the buffer cmd.
data	buffer of IDG command data.
dataLen,the	length of the buffer data.
response	Response data
respLen,the	length of Response data

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.5.4.32 int device_setCurrentDevice (int *deviceType*)

Sets the current device to talk to

The connect status can be checked by device_isConnected().

Parameters

deviceType	Device to connect to
	
	`enum DEVICE_TYPE`
	`{`
	`IDT_DEVICE_UNKNOWN=0,`
	`IDT_DEVICE_AUGUSTA_HID,`
	`IDT_DEVICE_AUGUSTA_KB,`
	`IDT_DEVICE_AUGUSTA_S_HID,`
	`IDT_DEVICE_AUGUSTA_S_KB,`
	`IDT_DEVICE_AUGUSTA_S_TTK_HID,`
	`IDT_DEVICE_SPECTRUM_PRO,`
	`IDT_DEVICE_MINISMART_II,`
	`IDT_DEVICE_L100,`
	`IDT_DEVICE_UNIPAY,`
	`IDT_DEVICE_UNIPAY_I_V,`
	`IDT_DEVICE_VP3300_AJ,`
	`IDT_DEVICE_KIOSK_III,`
	`IDT_DEVICE_KIOSK_III_S,`
	`IDT_DEVICE_PIP_READER,`
	`IDT_DEVICE_VENDI,`
	`IDT_DEVICE_VP3300_USB,`
	`IDT_DEVICE_UNIPAY_I_V_TTK,`
	`IDT_DEVICE_VP3300_BT,`
	`IDT_DEVICE_VP8800,`
	`IDT_DEVICE_SREDKEY2_HID,`
	`IDT_DEVICE_SREDKEY2_KB,`
	`IDT_DEVICE_NEO2,`
	`IDT_DEVICE_MINISMART_II_COM = IDT_DEVICE_NEO2+5,`
	`IDT_DEVICE_SPECTRUM_PRO_COM,`
	`IDT_DEVICE_KIOSK_III_COM,`
	`IDT_DEVICE_KIOSK_III_S_COM,`
	`IDT_DEVICE_VP3300_COM,`
	`IDT_DEVICE_NEO2_COM,`
	`IDT_DEVICE_MAX_DEVICES = IDT_DEVICE_NEO2_COM+5`
	`};`

Returns

RETURN_CODE: 1: success, 0: failed

12.5.4.33 void device_setSDKWaitTime (int *waitTime*)

Set SDK Wait Time

Set the SDK wait time for transactions

Parameters

waitTime	The SDK wait time for transaction in seconds

12.5.4.34 void device_setThreadStackSize (int *threadSize*)

Set Thread Stack Size

Set the stack size setting for newly created threads

12.5.4.35 int device_updateFirmware (IN BYTE * *firmwareData,* IN int *firmwareDataLen,* IN char * *firmwareName,* IN int *encryptionType,* IN BYTE * *keyBlob,* IN int *keyBlobLen*)

Update Firmware Updates the firmware of Augusta.

Parameters

firmwareData	Signed binary data of a firmware file provided by IDTech
firmwareData-Len	Length of firmwareData
firmwareName	Firmware name. • For example "Augusta_S_TTK_V1.00.002.fm"
encryptionType	Encryption type • 0 : Plaintext • 1 : TDES ECB, PKCS#5 padding • 2 : TDES CBC, PKCS#5, IV is all 0
keyBlob	Encrypted firmware session key blob, TR-31 Rev B, wrapped by FW Key (Optional, none if firmware is plaintext)
keyBlobLen	Length of keyBlob

Returns

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

Firmware update status is returned in the callback with the following values: sender = AUGUSTA state = DeviceState.FirmwareUpdate data = File Progress. Two bytes, with byte[0] = current block, and byte[1] = total blocks. 0x0310 = block 3 of 16 transactionResultCode:

- RETURN_CODE_DO_SUCCESS = Firmware Update Completed Successfully
- RETURN_CODE_BLOCK_TRANSFER_SUCCESS = Current block transferred successfully
- Any other return code represents an error condition

12.5.4.36 int emv_activateTransaction (IN int *timeout,* IN BYTE * *tags,* IN int *tagsLen,* IN int *forceOnline*)

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	Tags to be included in the request. Passed as a TLV stream. Example, tag 9F0C with amount 0x000000000100 would be 0x9F0C06000000000100
tagsLen	Length of tags
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 = DFEE1A079F029-F369f9F37

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString >>>>>IMPORTANT: parameters for amount and amtOther MUST BE PASSED AS A DOUBLE VALUE WITH DECIMAL POINT. Example, do not pass 1, but instead pass 1.0 or 1.00. Otherwise, results will be unpredictable

12.5.4.37 void emv_allowFallback (IN int *allow*)

Allow fallback for EMV transactions. Default is TRUE

Parameters

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

12.5.4.38 int emv_authenticateTransaction (IN BYTE * *updatedTLV,* IN int *updatedTLVLen*)

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.

Parameters

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, 9F37 to be included in response = DFEE1-A079F029F36959F37
updatedTLVLen	

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.5.4.39 int emv_authenticateTransactionWithTimeout (IN int *timeout,* IN BYTE * *updatedTLV,* IN int *updatedTLVLen*)

Authenticate EMV Transaction Request with Timeout

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.

Parameters

timeout	Timeout value in seconds.
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, 9F37 to be included in response = DFEE1-A079F029F36959F37

updatedTLVLen	

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.5.4.40 int emv_callbackResponseLCD (IN int *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_DISPLAY_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 device_getResponseCodeString()

12.5.4.41 int emv_callbackResponseMSR (IN BYTE * *MSR*, IN_OUT int *MSRLen*)

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
MSRLen	the length of Swiped track data

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.5.4.42 int emv_cancelTransaction ( )

Cancel EMV Transaction

Cancels the currently executing EMV transaction.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.5.4.43 int emv_completeTransaction (IN int *commError,* IN BYTE * *authCode,* IN int *authCodeLen,* IN BYTE * *iad,* IN int *iadLen,* IN BYTE * *tlvScripts,* IN int *tlvScriptsLen,* IN BYTE * *tlv,* IN int *tlvLen*)

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(1) if host was unreachable, or FALSE(0) 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
authCodeLen	the length of authCode
iad	Issuer Authentication Data, if any. Example 0x11223344556677883030 (tag value 91).
iadLen	the length of iadLen
tlvScripts	71/72 scripts, if any
tlvScriptsLen	the length of tlvScriptsLen
tlv	Additional TLV data to return with transaction results (if any)
tlvLen	the length of tlv

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.5.4.44 int emv_getAutoAuthenticateTransaction ( )

Gets auto authenticate value for EMV transactions.

Returns

RETURN_CODE: TRUE = auto authenticate, FALSE = manually authenticate

12.5.4.45 int emv_getAutoCompleteTransaction ( )

Gets auto complete value for EMV transactions.

Returns

RETURN_CODE: TRUE = auto complete, FALSE = manually complete

12.5.4.46 int emv_getEMVConfigurationCheckValue (OUT BYTE * *checkValue,* IN_OUT int * *checkValueLen*)

Get EMV Kernel configuration check value info

Parameters

checkValue	Response returned of Kernel configuration check value info
checkValueLen	the length of checkValue

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.5.4.47 int emv_getEMVKernelCheckValue (OUT BYTE * *checkValue,* IN_OUT int * *checkValueLen*)

Get EMV Kernel check value info

Parameters

checkValue	Response returned of Kernel check value info
checkValueLen	the length of checkValue

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.5.4.48 int emv_getEMVKernelVersion (OUT char * *version*)

DEPRECATED : please use emv_getEMVKernelVersion_Len(OUT char* version, IN_OUT int *versionLen)

Polls device for EMV Kernel Version

Parameters

version	Response returned of Kernel Version; needs to have at least 128 bytes of memory.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.5.4.49 int emv_getEMVKernelVersion_Len (OUT char * *version*, IN_OUT int * *versionLen*)

Polls device for EMV Kernel Version

Parameters

version	Response returned of Kernel Version
versionLen	Length of version

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.5.4.50 void emv_registerCallBk (pEMV_callBack *pEMVf*)

To register the emv callback function to get the EMV processing response. (Pass NULL to disable the callback.)

12.5.4.51 int emv_removeAllApplicationData ( )

Remove All Application Data

Removes all the Application Data

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.5.4.52 int emv_removeAllCAPK ( )

Remove All Certificate Authority Public Key

Removes all the CAPK

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.5.4.53 int emv_removeAllCRL ( )

Remove All Certificate Revocation List Entries

Removes all CRLEntry entries

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.5.4.54 int emv_removeApplicationData (IN BYTE * *AID*, IN int *AIDLen*)

Remove Application Data by AID

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

Parameters

AID	Name of ApplicationID Must be between 5 and 16 bytes
AIDLen	the length of AID data buffer

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.5.4.55 int emv_removeCAPK (IN BYTE * *capk*, IN int *capkLen*)

Remove Certificate Authority Public Key

Removes the CAPK as specified by the RID/Index

Parameters

capk	6 byte CAPK = 5 bytes RID + 1 byte INDEX
capkLen	the length of capk data buffer

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.5.4.56 int emv_removeCRL (IN BYTE * *list*, IN int *lsLen*)

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
lssLen	the length of list data buffer

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.5.4.57 int emv_removeTerminalData ( )

Remove Terminal Data

Removes the Terminal Data

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.5.4.58 int emv_retrieveAIDList (OUT BYTE * *AIDList*, IN_OUT int * *AIDListLen*)

Retrieve AID list

Returns all the AID names installed on the terminal.

Parameters

AIDList	array of AID name byte arrays
AIDListLen	the length of AIDList array buffer

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.5.4.59 int emv_retrieveApplicationData (IN BYTE * *AID*, IN int *AIDLen*, OUT BYTE * *tlv*, IN_OUT int * *tlvLen*)

Retrieve Application Data by AID

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

Parameters

AID	Name of ApplicationID. Must be between 5 and 16 bytes
AIDLen	the length of AID data buffer.
tlv	The TLV elements of the requested AID
tlvLen	the length of tlv data buffer.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.5.4.60 int emv_retrieveCAPK (IN BYTE * *capk*, IN int *capkLen*, OUT BYTE * *key*, IN_OUT int * *keyLen*)

Retrieve Certificate Authority Public Key

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

Parameters

capk	6 bytes CAPK = 5 bytes RID + 1 byte Index
capkLen	the length of capk data buffer
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.
keyLen	the length of key data buffer •

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.5.4.61 int emv_retrieveCAPKList (OUT BYTE * *keys*, IN_OUT int * *keysLen*)

Retrieve the Certificate Authority Public Key list

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

Parameters

keys	[key1][key2]...[keyn], each key 6 bytes where key = 5 bytes RID + 1 byte index
keysLen	the length of keys data buffer

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.5.4.62 int emv_retrieveCRL (OUT BYTE * *list*, IN_OUT int * *lssLen*)

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
lssLen	the length of list data buffer

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.5.4.63 int emv_retrieveTerminalData (OUT BYTE * *tlv*, IN_OUT int * *tlvLen*)

Retrieve Terminal Data

Retrieves the Terminal Data.

Parameters

tlv	Response returned as a TLV
tlvLen	the length of tlv data buffer

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.5.4.64 int emv_retrieveTerminalID (OUT char * *terminalID*)

DEPRECATED : please use emv_retrieveTerminalID_Len(OUT char* terminalID, IN_OUT int *terminalIDLen)

Gets the terminal ID as printable characters .

Parameters

terminalID	Terminal ID string; needs to have at least 30 bytes of memory

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.5.4.65 int emv_retrieveTerminalID_Len (OUT char * *terminalID*, IN_OUT int * *terminalIDLen*)

Gets the terminal ID as printable characters .

Parameters

terminalID	Terminal ID string
terminalIDLen	Length of terminalID

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.5.4.66 int emv_retrieveTransactionResult (IN BYTE * *tags*, IN int *tagsLen*, IDTTransactionData * *cardData*)

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
tagsLen	Length of tag list
cardData	All requested tags returned as unencrypted, encrypted and masked TLV data in IDTTransactionData object

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.5.4.67 int emv_setApplicationData (IN BYTE * *name*, IN int *nameLen*, IN BYTE * *tlv*, IN int *tlvLen*)

Set Application Data by AID

Sets the Application Data as specified by the application name and TLV data

Parameters

name	Application name, 10-32 ASCII hex characters representing 5-16 bytes Example "a0000000031010"
nameLen	the length of name data buffer of Application name,
tlv	Application data in TLV format
tlvLen	the length of tlv data buffer

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.5.4.68 void emv_setAutoAuthenticateTransaction (IN int *authenticate*)

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

Parameters

authenticate	TRUE = auto authenticate, FALSE = manually authenticate

12.5.4.69 void emv_setAutoCompleteTransaction (IN int *complete*)

Enables complete for EMV transactions. If a emv_authenticateTranaction results in code 0x0004 (go online), then emv_completeTransaction can automatically execute if parameter is set to TRUE

Parameters

complete	TRUE = auto complete, FALSE = manually complete

12.5.4.70 int emv_setCAPK (IN BYTE * *capk*, IN int *capkLen*)

Set Certificate Authority Public Key

Sets the CAPK as specified by the CAKey structure

Parameters

capk	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.
capkLen	the length of capk data buffer

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.5.4.71 int emv_setCRL (IN BYTE * *list*, IN int *lsLen*)

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]
lsLen	the length of list data buffer

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.5.4.72 int emv_setTerminalData (IN BYTE * *tlv*, IN int *tlvLen*)

Set Terminal Data

Sets the Terminal Data as specified by the TerminalData structure passed as a parameter

Parameters

tlv	TerminalData configuration file
tlvLen	the length of tlv data buffer

Return values

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

12.5.4.73 int emv_setTerminalID (IN char * *terminalID*)

Sets the terminal ID as printable characters .

Parameters

terminalID	Terminal ID to set

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.5.4.74 int emv_startTransaction (IN double *amount,* IN double *amtOther,* IN int *exponent,* IN int *type,* IN int *timeout,* IN BYTE * *tags,* IN int *tagsLen,* IN int *forceOnline*)

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) - SEE IMPORTANT NOTE BELOW
amtOther	Other amount value, if any (tag value 9F03) - SEE IMPORTANT NOTE BELOW
exponent	Number of characters after decimal point
type	Transaction type (tag value 9C).
timeout	Timeout value in seconds.
tags	Any other tags to be included in the request. Passed as a TLV stream. Example, tag 9F0C with amount 0x000000000100 would be 0x9F0C06000000000100 If tags 9F02 (amount),9-F03 (other amount), or 9C (transaction type) are included, they will take priority over these values supplied as individual parameters to this method.
tagsLen	Length of tags
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 = DFEE1A079F029-F36959F37

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString >>>>>IMPORTANT: parameters for amount and amtOther MUST BE PASSED AS A DOUBLE VALUE WITH DECIMAL POINT. Example, do not pass 1, but instead pass 1.0 or 1.00. Otherwise, results will be unpredictable

12.5.4.75 int icc_disable ( )

ICC Function enable/disable Disable ICC function

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.5.4.76 int icc_enable (IN int *withNotification*)

ICC Function enable/disable Enable ICC function with or without seated notification

Parameters

withNotification	• 1: with notification when ICC seated status changed, • 0: without notification.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.5.4.77 int icc_exchangeAPDU (IN BYTE * *c_APDU,* IN int *cLen,* OUT BYTE * *reData,* IN_OUT int * *reLen*)

Exchange APDU with plain text For Non-SRED Augusta Only

Sends an APDU packet to the ICC. If successful, response is the APDU data in response parameter.

Parameters

c_APDU	APDU data packet
cLen	APDU data packet length
reData	Unencrypted APDU response
reLen	Unencrypted APDU response data length

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.5.4.78 int icc_exchangeEncryptedAPDU (IN BYTE * *c_APDU,* IN int *cLen,* OUT BYTE * *reData,* IN_OUT int * *reLen*)

Exchange APDU with encrypted data For SRED Augusta Only

Sends an APDU packet to the ICC. If successful, response is the APDU data in response parameter.

Parameters

c_APDU	KSN + encytpted APDU data packet, or no KSN (use last known KSN) + encrypted APDU data packet With KSN: [0A][KSN][Encrypted C-APDU] Without KSN: [00][Encrypted C-APDU]

The format of Raw C-APDU Data Structure of [m-bytes Encrypted C-APDU] is below:

- m = 2 bytes Valid C-APDU Length + x bytes Valid C-APDU + y bytes Padding (0x00) Note: For TDES mode: 2+x should be multiple of 8. If it was not multiple of 8, unit should padded y bytes 0x00 automatically (2+x+y should be multiple of 8). For AES mode: 2+x should be multiple of 16. If it was not multiple of 16, unit should padded y bytes 0x00 automatically (2+x+y should be multiple of 16).

  Parameters

cLen	data packet length
reData	response encrypted APDU response. Can be three options:

[00] + [Plaintext R-APDU]

- [01] + [0A] + [KSN] + [n bytes Encrypted R-APDU without Status Bytes] + [2 bytes Status Bytes]
- [01] + [00] + [n bytes Encrypted R-APDU without Status Bytes] + [2 bytes Status Bytes]

  The KSN, when provided, will be 10 bytes. The KSN will only be provided when it has changed since the last provided KSN. Each card Power-On generates a new KSN. During a sequence of commands where the KSN

is identical, the first response will have a KSN length set to [0x0A] followed by the KSN, while subsequent commands with the same KSN value will have a KSN length of [0x00] followed by the Encrypted R-APDU without Status Bytes.

Parameters

reLen	encrypted APDU response data length

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.5.4.79 int icc_getAPDU_KSN (OUT BYTE * *KSN,* IN_OUT int * *inLen*)

Get APDU KSN

Retrieves the KSN used in ICC Encypted APDU usage

Parameters

KSN	Returns the encrypted APDU packet KSN
inLen	KSN data length

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.5.4.80 int icc_getFunctionStatus (OUT int * *enabled,* OUT int * *withNotification*)

Get ICC Function status Get ICC Function status about enable/disable and with or without seated notification

Parameters

enabled	• 1: ICC Function enabled, • 0: means disabled.
withNotification	1 means with notification when ICC seated status changed. 0 means without notification.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.5.4.81 int icc_getICCReaderStatus (OUT 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

12.5.4.82 int icc_getKeyFormatForICCDUKPT (OUT BYTE * *format*)

Get Key Format For DUKPT

Specifies how data will be encrypted with Data Key or PIN key (if DUKPT key loaded). This applies to both MSR and ICC

Parameters

format	Response returned from method: • 'TDES': Encrypted card data with TDES if DUKPT Key had been loaded.(default) • 'AES': Encrypted card data with AES if DUKPT Key had been loaded. • 'NONE': No Encryption.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.5.4.83 int icc_getKeyTypeForICCDUKPT (OUT BYTE * *type*)

Get Key Type for DUKPT

Specifies the key type used for DUKPT encryption This applies to both MSR and ICC

Parameters

type	Response returned from method: • 'DATA': Encrypted card data with Data Key DUKPT Key had been loaded.(default) • 'PIN': Encrypted card data with PIN Key if DUKPT Key had been loaded.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.5.4.84 int icc_powerOffICC ( )

Power Off ICC

Powers down the ICC

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

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

12.5.4.85 int icc_powerOnICC (OUT BYTE * *ATR,* IN_OUT int * *inLen*)

Power On ICC

Power up the currently selected microprocessor card in the ICC reader

Parameters

ATR,the	ATR data response when succeeded power on ICC,
inLen,the	length of ATR data

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.5.4.86 void msr_registerCallBk (pMSR_callBack *pMSRf*)

To register the msr callback function to get the MSR card data. (Pass NULL to disable the callback.)

12.5.4.87 void msr_registerCallBkp (pMSR_callBackp *pMSRf*)

To register the msr callback function to get the MSR card data pointer. (Pass NULL to disable the callback.)

12.5.4.88 void pin_registerCallBk (pPIN_callBack *pPINf*)

To register the pin callback function to get the PINPad data. (Pass NULL to disable the callback.)

12.5.4.89 void registerHotplugCallBk (pMessageHotplug *pMsgHotplug*)

To register the USB HID hot-plug callback function which implemented in the application to monitor the hotplug message from the SDK.

12.5.4.90 void registerLogCallBk (pSendDataLog *pFSend,* pReadDataLog *pFRead*)

To register the log callback function which implemented in the application to monitor sending/reading data between application and reader.

12.5.4.91 int rs232_device_init (int *deviceType,* int *port_number,* int *brate*)

Initial the device by RS232

It will try to connect to the device with provided deviceType, port_number, and brate.

Parameters

deviceType	Device to connect to
port_number	Port number of the device

Port nr. | **Linux** | **Windows**

| 0 | ttyS0 | COM1 | | 1 | ttyS1 | COM2 | | 2 | ttyS2 | COM3 | | 3 | ttyS3 | COM4 | | 4 | ttyS4 | COM5 | | 5 | ttyS5 | COM6 | | 6 | ttyS6 | COM7 | | 7 | ttyS7 | COM8 | | 8 | ttyS8 | COM9 | | 9 | ttyS9 | COM10 | | 10 | ttyS10 | COM11 | | 11 | ttyS11 | COM12 | | 12 | ttyS12 | COM13 | | 13 | ttyS13 | COM14 | | 14 | ttyS14 | COM15 | | 15 | ttyS15 | COM16 | | 16 | ttyUSB0 | n.a. | | 17 | ttyUSB1 | n.a. | | 18 | ttyUSB2 | n.a. | | 19 | ttyUSB3 | n.a. | | 20 | ttyUSB4 | n.a. | | 21 | ttyUSB5 | n.a. | | 22 | ttyAMA0 | n.a. | | 23 | ttyAMA1 | n.a. | | 24 | ttyACM0 | n.a. | | 25 | ttyACM1 | n.a. | | 26 | rfcomm0 | n.a. | | 27 | rfcomm1 | n.a. | | 28 | ircomm0 | n.a. | | 29 | ircomm1 | n.a. | | 30 | cuau0 | n.a. | | 31 | cuau1 | n.a. | | 32 | cuau2 | n.a. | | 33 | cuau3 | n.a. | | 34 | cuaU0 | n.a. | | 35 | cuaU1 | n.a. | | 36 | cuaU2 | n.a. | | 37 | cuaU3 | n.a. |

Parameters

brate	Bitrate of the device

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.5.4.92 char* SDK_Version ( )

To Get SDK version

Returns

return the SDK version string

12.5.4.93 int setAbsoluteLibraryPath (const char * *absoluteLibraryPath*)

Set the path to use when searching for ID TECH's libraries. If this is not set, the libraries will be searched for with the system's default procedures.

Parameters

absoluteLibrary-Path	The absolute path to ID TECH's libraries.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.6 Source_C/libIDT_NEO2.h File Reference

NEO2 API.

```
#include "IDTDef.h"
```

Macros

- #define IN
- #define OUT
- #define IN_OUT

Typedefs

- typedef void(* pMessageHotplug)(int, int)
- typedef void(* pSendDataLog)(unsigned char *, int)
- typedef void(* pReadDataLog)(unsigned char *, int)
- typedef void(* pEMV_callBack)(int, int, unsigned char *, int, IDTTransactionData *, EMV_Callback *, int)
- typedef void(* pFW_callBack)(int, int, int, int, int)
- typedef void(* pMSR_callBack)(int, IDTMSRData)
- typedef void(* pMSR_callBackp)(int, IDTMSRData *)
- typedef void(* pPIN_callBack)(int, IDTPINData *)
- typedef void(* pCMR_callBack)(int, IDTCMRData *)
- typedef void(* pCSFS_callBack)(BYTE status)
- typedef void(* pLCD_callBack)(int, IDTLCDItem *)
- typedef void(* ftpComm_callBack)(int, int, int)
- typedef void(* httpComm_callBack)(BYTE *, int)
- typedef void(* v4Comm_callBack)(BYTE, BYTE, BYTE *, int)

Functions

- void registerHotplugCallBk (pMessageHotplug pMsgHotplug)
- void registerLogCallBk (pSendDataLog pFSend, pReadDataLog pFRead)
- void device_registerFWCallBk (pFW_callBack pFWf)
- void device_registerCameraCallBk (pCMR_callBack pCMRf)
- void device_registerCardStatusFrontSwitchCallBk (pCSFS_callBack pCSFSf)
- void emv_registerCallBk (pEMV_callBack pEMVf)
- void loyalty_registerCallBk (pEMV_callBack pEMVf)
- void msr_registerCallBk (pMSR_callBack pMSRf)
- void msr_registerCallBkp (pMSR_callBackp pMSRf)
- void ctls_registerCallBk (pMSR_callBack pCTLSf)
- void ctls_registerCallBkp (pMSR_callBackp pCTLSf)
- void pin_registerCallBk (pPIN_callBack pPINf)
- void lcd_registerCallBk (pLCD_callBack pLCDf)
- void comm_registerHTTPCallback (httpComm_callBack cBack)
- void comm_registerV4Callback (v4Comm_callBack cBack)
- char * SDK_Version ()
- int setAbsoluteLibraryPath (const char *absoluteLibraryPath)
- int device_setConfigPath (const char *path)
- int device_setNEO2DevicesConfigs (IN const char *configs, IN int len)
- int device_init ()
- int rs232_device_init (int deviceType, int port_number, int brate)
- int device_setCurrentDevice (int deviceType)
- int device_isAttached (int deviceType)
- int device_close ()
- void device_getIDGStatusCodeString (IN int returnCode, OUT char *despcrition)
- int device_isConnected ()
- int device_getFirmwareVersion (OUT char *firmwareVersion)
- int device_getFirmwareVersion_Len (OUT char *firmwareVersion, IN_OUT int *firmwareVersionLen)
- int device_pingDevice ()
- int device_controlUserInterface (IN BYTE *values)
- int device_getCurrentDeviceType ()
- int device_SendDataCommandNEO (IN int cmd, IN int subCmd, IN BYTE *data, IN int dataLen, OUT BYTE *response, IN_OUT int *respLen)
- int device_getSDKWaitTime ()
- void device_setSDKWaitTime (int waitTime)
- int device_getThreadStackSize ()
- void device_setThreadStackSize (int threadSize)
- void device_toSDCard (int forSDCard)
- int ctls_displayOnlineAuthResult (IN int statusCode, IN BYTE *TLV, IN int TLVLen)
- int device_enablePassThrough (int enablePassThrough)
- int device_enableL100PassThrough (int enableL100PassThrough)
- int device_getL100PassThroughMode ()
- int device_setBurstMode (IN BYTE mode)
- int device_setPollMode (IN BYTE mode)
- int device_pollForToken (IN int timeout, OUT BYTE *respData, IN_OUT int *respDataLen)
- int device_setMerchantRecord (int index, int enabled, char *merchantID, char *merchantURL)
- int device_getMerchantRecord (IN int index, OUT BYTE *record)
- int device_getMerchantRecord_Len (IN int index, OUT BYTE *record, IN_OUT int *recordLen)
- int device_getTransactionResults (IDTMSRData *cardData)
- int device_startTransaction (IN double amount, IN double amtOther, IN int type, IN const int _timeout, IN BYTE *tags, IN int tagsLen)
- int loyalty_startTransaction (IN double amount, IN double amtOther, IN int type, IN const int _timeout, IN BYTE *tags, IN int tagsLen, IN const int cardType, IN const int iccReadType)

- void device_setTransactionExponent (int exponent)
- int device_activateTransaction (IN const int _timeout, IN BYTE *tags, IN int tagsLen)
- int device_cancelTransaction ()
- int loyalty_cancelTransaction ()
- int device_setCancelTransactionMode (int mode)
- int device_cancelTransactionSilent (int enable)
- int loyalty_cancelTransactionSilent (int enable)
- int device_configureButtons (IN BYTE done, IN BYTE swipe, IN BYTE delay)
- int device_getButtonConfiguration (OUT BYTE *done, OUT BYTE *swipe, OUT BYTE *delay)
- int device_disableBlueLED ()
- int device_enableBlueLED (IN BYTE *data, IN int dataLen)
- int device_lcdDisplayClear ()
- int device_enableExternalLCDMessages (IN int enableExtLCDMsg)
- int device_enableRFAntenna (IN int enableAntenna)
- int device_turnOffYellowLED ()
- int device_turnOnYellowLED ()
- int device_buzzerOnOff ()
- int device_lcdDisplayLine1Message (IN BYTE *message, IN int messageLen)
- int device_lcdDisplayLine2Message (IN BYTE *message, IN int messageLen)
- int device_getKeyStatus (int *newFormat, BYTE *status, int *statusLen)
- int device_updateFirmware (IN BYTE *firmwareData, IN int firmwareDataLen, IN char *firmwareName, IN int encryptionType, IN BYTE *keyBlob, IN int keyBlobLen)
- int device_transferFile (IN char *fileName, IN int fileNameLen, IN BYTE *file, IN int fileLen)
- int device_deleteFile (IN char *fileName, IN int fileNameLen)
- int device_queryFile (IN char *directoryName, IN int directoryNameLen, IN char *fileName, IN int fileNameLen, OUT int *isExist, OUT BYTE *timeStamp, IN_OUT int *timeStampLen, OUT char *fileSize, IN_OUT int *fileSizeLen)
- int device_startListenNotifications ()
- int device_stopListenNotifications ()
- int device_startQRCodeScan (IN int _timeout)
- int device_stopQRCodeScan ()
- int device_startTakingPhoto (IN int _timeout)
- int device_stopTakingPhoto ()
- void device_getResponseCodeString (IN int returnCode, OUT char *despcrition)
- int device_listDirectory (IN char *directoryName, IN int directoryNameLen, IN int recursive, IN int onSD, OUT char *directory, IN_OUT int *directoryLen)
- int device_deleteDirectory (IN char *dirName, IN int dirNameLen)
- int device_getDeviceMemoryUsageInfo (OUT int *freeHeapSize, OUT int *notFreedBlockCnt, OUT int *minEverFreeHeapSize)
- int felica_authentication (IN BYTE *key, IN int keyLen)
- int felica_readWithMac (IN int blockCnt, IN BYTE *blockList, IN int blockListLen, OUT BYTE *blockData, OUT int *blockDataLen)
- int felica_writeWithMac (IN BYTE blockNum, IN BYTE *blockData, IN int blockDataLen)
- int felica_read (IN BYTE *serviceCodeList, IN int serviceCodeListLen, IN int blockCnt, IN BYTE *blockList, IN int blockListLen, OUT BYTE *blockData, OUT int *blockDataLen)
- int felica_write (IN BYTE *serviceCodeList, IN int serviceCodeListLen, IN int blockCnt, IN BYTE *blockList, IN int blockListLen, IN BYTE *blockData, IN int blockDataLen, OUT BYTE *statusFlag, OUT int *statusFlagLen)
- int felica_poll (IN BYTE *systemCode, IN int systemCodeLen, OUT BYTE *respData, OUT int *respDataLen)
- int felica_SendCommand (IN BYTE *command, IN int commandLen, OUT BYTE *respData, OUT int *respDataLen)
- int felica_requestService (IN BYTE *nodeCode, IN int nodeCodeLen, OUT BYTE *respData, OUT int *respDataLen)
- int config_getSerialNumber (OUT char *sNumber)
- int config_getSerialNumber_Len (OUT char *sNumber, IN_OUT int *sNumberLen)
- int config_getModelNumber (OUT char *sNumber)

- int config_getModelNumber_Len (OUT char *sNumber, IN_OUT int *sNumberLen)
- int config_setCmdTimeOutDuration (IN int millisecond)
- int ctls_startTransaction (IN double amount, IN double amtOther, IN int type, IN const int _timeout, IN BYTE *tags, IN int tagsLen)
- int ctls_activateTransaction (IN const int _timeout, IN BYTE *tags, IN int tagsLen)
- int ctls_cancelTransaction ()
- int ctls_retrieveApplicationData (IN BYTE *AID, IN int AIDLen, OUT BYTE *tlv, IN_OUT int *tlvLen)
- int ctls_setApplicationData (IN BYTE *tlv, IN int tlvLen)
- int ctls_removeApplicationData (IN BYTE *AID, IN int AIDLen)
- int ctls_removeAllApplicationData ()
- int ctls_retrieveAIDList (OUT BYTE *AIDList, IN_OUT int *AIDListLen)
- int ctls_retrieveTerminalData (OUT BYTE *tlv, IN_OUT int *tlvLen)
- int ctls_setTerminalData (IN BYTE *tlv, IN int tlvLen)
- int ctls_retrieveCAPK (IN BYTE *capk, IN int capkLen, OUT BYTE *key, IN_OUT int *keyLen)
- int ctls_setCAPK (IN BYTE *capk, IN int capkLen)
- int ctls_removeCAPK (IN BYTE *capk, IN int capkLen)
- int ctls_removeAllCAPK ()
- int ctls_retrieveCAPKList (OUT BYTE *keys, IN_OUT int *keysLen)
- int ctls_setConfigurationGroup (IN BYTE *tlv, IN int tlvLen)
- int ctls_getConfigurationGroup (IN int group, OUT BYTE *tlv, OUT int *tlvLen)
- int ctls_getAllConfigurationGroups (OUT BYTE *tlv, IN_OUT int *tlvLen)
- int ctls_removeConfigurationGroup (int group)
- int emv_getEMVKernelVersion (OUT char *version)
- int emv_getEMVKernelVersion_Len (OUT char *version, IN_OUT int *versionLen)
- int emv_getEMVKernelCheckValue (OUT BYTE *checkValue, IN_OUT int *checkValueLen)
- int emv_getEMVConfigurationCheckValue (OUT BYTE *checkValue, IN_OUT int *checkValueLen)
- void emv_allowFallback (IN int allow)
- void emv_setAutoAuthenticateTransaction (IN int authenticate)
- void emv_setAutoCompleteTransaction (IN int complete)
- int emv_getAutoAuthenticateTransaction ()
- int emv_getAutoCompleteTransaction ()
- void emv_setTransactionParameters (IN double amount, IN double amtOther, IN int type, IN int timeout, IN BYTE *tags, IN int tagsLen)
- int emv_startTransaction (IN double amount, IN double amtOther, IN int exponent, IN int type, IN int timeout, IN BYTE *tags, IN int tagsLen, IN int forceOnline)
- int emv_activateTransaction (IN int timeout, IN BYTE *tags, IN int tagsLen, IN int forceOnline)
- int emv_authenticateTransaction (IN BYTE *updatedTLV, IN int updatedTLVLen)
- int emv_authenticateTransactionWithTimeout (IN int timeout, IN BYTE *updatedTLV, IN int updatedTLVLen)
- int emv_completeTransaction (IN int commError, IN BYTE *authCode, IN int authCodeLen, IN BYTE *iad, IN int iadLen, IN BYTE *tlvScripts, IN int tlvScriptsLen, IN BYTE *tlv, IN int tlvLen)
- int emv_cancelTransaction ()
- int emv_retrieveTransactionResult (IN BYTE *tags, IN int tagsLen, OUT IDTTransactionData *cardData)
- int emv_retrieveApplicationData (IN BYTE *AID, IN int AIDLen, OUT BYTE *tlv, IN_OUT int *tlvLen)
- int emv_setApplicationData (IN BYTE *name, IN int nameLen, IN BYTE *tlv, IN int tlvLen)
- int emv_setApplicationDataTLV (IN BYTE *tlv, IN int tlvLen)
- int emv_removeApplicationData (IN BYTE *AID, IN int AIDLen)
- int emv_removeAllApplicationData ()
- int emv_retrieveAIDList (OUT BYTE *AIDList, IN_OUT int *AIDListLen)
- int emv_retrieveTerminalData (OUT BYTE *tlv, IN_OUT int *tlvLen)
- int emv_setTerminalData (IN BYTE *tlv, IN int tlvLen)
- int emv_setTerminalMajorConfiguration (IN int configuration)
- int emv_retrieveCAPK (IN BYTE *capk, IN int capkLen, OUT BYTE *key, IN_OUT int *keyLen)
- int emv_setCAPK (IN BYTE *capk, IN int capkLen)
- int emv_removeCAPK (IN BYTE *capk, IN int capkLen)
- int emv_removeAllCAPK ()

- int emv_retrieveCAPKList (OUT BYTE *keys, IN_OUT int *keysLen)
- int emv_retrieveCRL (OUT BYTE *list, IN_OUT int *lssLen)
- int emv_setCRL (IN BYTE *list, IN int lsLen)
- int emv_removeCRL (IN BYTE *list, IN int lsLen)
- int emv_removeAllCRL ()
- int icc_getICCReaderStatus (OUT BYTE *status)
- int icc_powerOnICC (OUT BYTE *ATR, IN_OUT int *inLen)
- int icc_powerOffICC ()
- int icc_exchangeAPDU (IN BYTE *c_APDU, IN int cLen, OUT BYTE *reData, IN_OUT int *reLen)
- int lcd_createScreen (IN char *screenName, IN int screenNameLen, OUT int *ScreenID)
- int lcd_destroyScreen (IN char *screenName, IN int screenNameLen)
- int lcd_getActiveScreen (OUT char *screenName, IN_OUT int *screenNameLen)
- int lcd_showScreen (IN char *screenName, IN int screenNameLen)
- int lcd_getButtonEvent (OUT int *screenID, OUT int *objectID, OUT char *screenName, IN_OUT int *screenNameLen, OUT char *objectName, IN_OUT int *objectNameLen, OUT int *isLongPress)
- int lcd_addButton (IN char *screenName, IN int screenNameLen, IN char *buttonName, IN int buttonNameLen, IN BYTE type, IN BYTE alignment, IN int xCord, IN int yCord, IN char *label, IN int labelLen, OUT IDTLCDItem *returnItem)
- int lcd_addEthernet (IN char *screenName, IN int screenNameLen, IN char *objectName, IN int objectNameLen, IN BYTE alignment, IN int xCord, IN int yCord, OUT IDTLCDItem *returnItem)
- int lcd_addLED (IN char *screenName, IN int screenNameLen, IN char *objectName, IN int objectNameLen, IN BYTE alignment, IN int xCord, IN int yCord, OUT IDTLCDItem *returnItem, IN BYTE *LED, IN int LEDLen)
- int lcd_addText (IN char *screenName, IN int screenNameLen, IN char *objectName, IN int objectNameLen, IN BYTE alignment, IN int xCord, IN int yCord, IN int width, IN int height, IN BYTE fontID, IN BYTE *color, IN int colorLen, IN char *label, IN int labelLen, OUT IDTLCDItem *returnItem)
- int lcd_addImage (IN char *screenName, IN int screenNameLen, IN char *objectName, IN int objectNameLen, IN BYTE alignment, IN int xCord, IN int yCord, IN char *filename, IN int filenameLen, OUT IDTLCDItem *returnItem)
- int lcd_addVideo (IN char *screenName, IN int screenNameLen, IN char *objectName, IN int objectNameLen, IN BYTE alignment, IN int xCord, IN int yCord, IN char *filename, IN int filenameLen, OUT IDTLCDItem *returnItem)
- int lcd_cloneScreen (IN char *screenName, IN int screenNameLen, IN char *cloneName, IN int cloneNameLen, OUT int *cloneID)
- int lcd_updateLabel (IN char *screenName, IN int screenNameLen, IN char *objectName, IN int objectNameLen, IN char *label, IN int labelLen)
- int lcd_updateColor (IN char *screenName, IN int screenNameLen, IN char *objectName, IN int objectNameLen, IN BYTE *color, IN int colorLen)
- int lcd_updatePosition (IN char *screenName, IN int screenNameLen, IN char *objectName, IN int objectNameLen, IN BYTE alignment, IN int new_xCord, IN int new_yCord)
- int lcd_removeItem (IN char *screenName, IN int screenNameLen, IN char *objectName, IN int objectNameLen)
- int lcd_storeScreenInfo ()
- int lcd_loadScreenInfo ()
- int lcd_clearScreenInfo ()
- int lcd_getAllScreens (IN_OUT int *screenNumbers, OUT IDTScreenInfo *screenInfo)
- int lcd_getAllObjects (IN char *screenName, IN int screenNameLen, IN_OUT int *objectNumbers, OUT IDTObjectInfo *objectInfo)
- int lcd_queryScreenbyName (IN char *screenName, IN int screenNameLen, OUT int *result)
- int lcd_queryObjectbyName (IN char *objectName, IN int objectNameLen, IN_OUT int *objectNumbers, OUT IDTScreenInfo *screenInfo)
- int lcd_queryScreenbyID (IN int screenID, OUT int *result, OUT int *screenName, IN_OUT int *screenNameLen)
- int lcd_queryObjectbyID (IN int objectID, OUT int *objectNumbers, OUT IDTScreenInfo *screenInfo)
- int lcd_setBacklight (IN BYTE backlightVal)
- int msr_cancelMSRSwipe ()
- int msr_startMSRSwipe (IN int _timeout)

- void parseMSRData (IN BYTE ∗resData, IN int resLen, IN_OUT IDTMSRData ∗cardData)
- int pin_capturePin (IN int timeout, IN int type, IN char ∗PAN, IN int PANLen, IN int minPIN, IN int maxPIN, IN char ∗message, IN int messageLen)
- int pin_capturePinExt (IN int type, IN char ∗PAN, IN int PANLen, IN int minPIN, IN int maxPIN, IN char ∗message, IN int messageLen, IN char ∗verify, IN int verifyLen)
- int pin_promptForNumericKeyWithSwipe (IN int timeout, IN BYTE function, IN int minLen, IN int maxLen, IN char ∗line1, IN int line1Len, IN char ∗line2, IN int line2Len, BYTE ∗signature, IN int signatureLen)
- int pin_promptForNumericKey (IN int timeout, IN int maskInput, IN int minLen, IN int maxLen, IN char ∗message, IN int messageLen, BYTE ∗signature, IN int signatureLen)
- int pin_inputFromPrompt (BYTE mask, BYTE preClearText, BYTE postClearText, int minLen, int maxLen, char ∗lang, BYTE promptID, char ∗defaultResponse, int defaultResponseLen, int timeout)
- int pin_getPanEntry (IN int csc, IN int expDate, IN int ADR, IN int ZIP, IN int mod10CK, IN int timeout, IN int encPANOnly)
- int pin_cancelPINEntry ()
- int pin_setKeyValues (int mode)

12.6.1 Detailed Description

NEO2 API. NEO2 Global API methods.

12.6.2 Macro Definition Documentation

12.6.2.1 #define IN

INPUT parameter.

12.6.2.2 #define IN_OUT

INPUT / OUTPUT PARAMETER.

12.6.2.3 #define OUT

OUTPUT parameter.

12.6.3 Typedef Documentation

12.6.3.1 typedef void(∗ ftpComm_callBack)(int, int, int)

Define the comm callback function to get FTP file transfer status

It should be passed as a parameter in a FTP request, Signature (int, int, int) = response code, current block, total blocks RESPONSE CODES: 100 = FILE DOWNLOAD STARTED 101 = FILE BLOCK XX OF XX RECEIVED 102 = FILE DOWNLOAD COMPLETED 103 = FILE DOWNLOAD TERMINATED PREMATURELY

12.6.3.2 typedef void(∗ httpComm_callBack)(BYTE ∗, int)

Define the comm callback function to get the async url data

It should be registered using the comm_registerHTTPCallback

12.6.3.3 typedef void(* pCMR_callBack)(int, IDTCMRData *)

Define the camera callback function to get the image data

It should be registered using the device_registerCameraCallBk,

12.6.3.4 typedef void(* pCSFS_callBack)(BYTE status)

Define the card status and front switch callback function to get card and front switch status

It should be registered using the device_registerCardStatusFrontSwitchCallBk,

12.6.3.5 typedef void(* pEMV_callBack)(int, int, unsigned char *, int, IDTTransactionData *, EMV_Callback *, int)

Define the EMV callback function to get the transaction message/data/result.

It should be registered using the emv_registerCallBk

12.6.3.6 typedef void(* pFW_callBack)(int, int, int, int, int)

Define the firmware update callback function to get the firmware update status

It should be registered using the device_registerFWCallBk

12.6.3.7 typedef void(* pLCD_callBack)(int, IDTLCDItem *)

Define the LCD callback function to get the input LCDItem

It should be registered using the lcd_registerCallBk,

12.6.3.8 typedef void(* pMessageHotplug)(int, int)

Define the USB hot-plug callback function to monitor the info when plug in/out the reader.

It should be registered using the registerHotplugCallBk, The first integer parameter is device type, and the second integer parameter is either 0: Device Plugged Out or 1: Device Plugged In

12.6.3.9 typedef void(* pMSR_callBack)(int, IDTMSRData)

Define the MSR callback function to get the MSR card data

It should be registered using the msr_registerCallBk, this callback function is for backward compatibility

12.6.3.10 typedef void(* pMSR_callBackp)(int, IDTMSRData *)

Define the MSR callback function to get pointer to the MSR card data

It should be registered using the msr_registerCallBk, this callback function is recommended instead of pMSR_callBack

12.6.3.11 typedef void(* pPIN_callBack)(int, IDTPINData *)

Define the PINPad callback function to get the input PIN Pad data

It should be registered using the pin_registerCallBk,

12.6.3.12 typedef void(∗ pReadDataLog)(unsigned char ∗, int)

Define the read response callback function to monitor the reading response from the reader.

It should be registered using the registerLogCallBk

12.6.3.13 typedef void(∗ pSendDataLog)(unsigned char ∗, int)

Define the send command callback function to monitor the sending command into the reader.

It should be registered using the registerLogCallBk

12.6.3.14 typedef void(∗ v4Comm_callBack)(BYTE, BYTE, BYTE ∗, int)

Define the comm callback function to receive the V4 Protocol packets received by the device from an external source (IP/USB/RS-232) It should be registered using the comm_registerV4Callback, Data callback will contain command, sub-command, and data from V4 packet

12.6.4 Function Documentation

12.6.4.1 void comm_registerHTTPCallback (httpComm_callBack *cBack*)

Register Comm HTTP Async Callback

Parameters

cBack	- HTTP Comm callback

12.6.4.2 void comm_registerV4Callback (v4Comm_callBack *cBack*)

Register External V4 Protocol commands Callback

Parameters

cBack	- V4 Protocol Comm callback

12.6.4.3 int config_getModelNumber (OUT char ∗ *sNumber*)

DEPRECATED : please use config_getModelNumber_Len(OUT char∗ sNumber, IN_OUT int ∗sNumberLen)

Polls device for Model Number

Parameters

sNumber	Returns Model Number; needs to have at least 64 bytes of memory

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString

12.6.4.4 int config_getModelNumber_Len (OUT char ∗ *sNumber,* IN_OUT int ∗ *sNumberLen*)

Polls device for Model Number

Parameters

sNumber	Returns Model Number
sNumber	length of Model Number

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString

12.6.4.5 int config_getSerialNumber (OUT char * *sNumber*)

DEPRECATED : please use config_getSerialNumber_Len(OUT char* sNumber, IN_OUT int *sNumberLen)

Polls device for Serial Number

Parameters

sNumber	Returns Serial Number; needs to have at least 64 bytes of memory

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString

12.6.4.6 int config_getSerialNumber_Len (OUT char * *sNumber,* IN_OUT int * *sNumberLen*)

Polls device for Serial Number

Parameters

sNumber	Returns Serial Number
sNumberLen	Length of Serial Number

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString

12.6.4.7 int config_setCmdTimeOutDuration (IN int *millisecond*)

Set the timeout duration for regular commands The new timeout value will affect all the functions actually send (sync) commands that doesn't need to wait for a callback function, such as device_getFirmwareVersion(), device_pingDevice(), device_SendDataCommandNEO(), device_enablePassThrough(), device_setBurstMode(), device_setPollMode(), device_updateFirmware() . . . etc.

Parameters

millisecond	timeout value in milliseconds

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.6.4.8 int ctls_activateTransaction (IN const int *_timeout,* IN BYTE * *tags,* IN int *tagsLen*)

Start CTLS Transaction Request

Authorizes the CTLS 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 stream. Example, tag 9F02 with amount 0x000000000100 would be 0x9F0206000000000100
tagsLen	The length of tags data buffer.

>>>>>IMPORTANT: parameters for amount and amtOther MUST BE PASSED AS A DOUBLE VALUE WITH DECIMAL POINT. Example, do not pass 1, but instead pass 1.0 or 1.00. Otherwise, results will be unpredictable

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString() Note: if auto poll is on, it will returm the error IDG_P2_STATUS_CODE_COMMAND_NOT_ALLOWED

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 DF01. 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

12.6.4.9 int ctls_cancelTransaction ( )

Cancel EMV Transaction

Cancels the currently executing EMV transaction.

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.6.4.10 int ctls_displayOnlineAuthResult (IN int *statusCode*, IN BYTE * *TLV*, IN int *TLVLen*)

Start Remote Key Injection

Starts a remote key injection request with IDTech RKI servers. This function is reserved and not implemented.

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

Display Online Authorization Result Use this command to display the status of an online authorization request on the reader's display (OK or NOT OK). Use this command after the reader sends an online request to the issuer.

Parameters

statusCode	1 = OK, 0 = NOT OK, 2 = ARC response 89 for Interac
TLV	Optional TLV for AOSA
TLVLen	TLV Length

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.6.4.11 int ctls_getAllConfigurationGroups (OUT BYTE * *tlv*, IN_OUT int * *tlvLen*)

Retrieve All Configuration Groups

Returns all the Configuration Groups installed on the terminal for CTLS

Parameters

tlv	The TLV elements data
tlvLen	the length of tlv data buffer.

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.6.4.12 int ctls_getConfigurationGroup (IN int *group*, OUT BYTE * *tlv*, OUT int * *tlvLen*)

Get Configuration Group

Retrieves the Configuration for the specified Group.

Parameters

group	Configuration Group
tlv	return data
tlvLen	the length of tlv data buffer

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.6.4.13 void ctls_registerCallBk (pMSR_callBack *pCTLSf*)

To register the ctls callback function to get the MSR card data. (Pass NULL to disable the callback.)

12.6.4.14 void ctls_registerCallBkp (pMSR_callBackp *pCTLSf*)

To register the ctls callback function to get the MSR card data pointer. (Pass NULL to disable the callback.)

12.6.4.15 int ctls_removeAllApplicationData ( )

Remove All Application Data

Removes all the Application Data

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.6.4.16 int ctls_removeAllCAPK ( )

Remove All Certificate Authority Public Key

Removes all the CAPK

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.6.4.17 int ctls_removeApplicationData (IN BYTE * *AID,* IN int *AIDLen*)

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
AIDLen	the length of AID data buffer

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.6.4.18 int ctls_removeCAPK (IN BYTE * *capk,* IN int *capkLen*)

Remove Certificate Authority Public Key

Removes the CAPK as specified by the RID/Index

Parameters

capk	6 byte CAPK = 5 bytes RID + 1 byte INDEX
capkLen	the length of capk data buffer

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.6.4.19 int 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	Values can be parsed with device_getIDGStatusCodeString()

12.6.4.20 int ctls_retrieveAIDList (OUT BYTE * *AIDList*, IN_OUT int * *AIDListLen*)

Retrieve AID list

Returns all the AID names installed on the terminal for CTLS. .

Parameters

AIDList	array of AID name byte arrays
AIDListLen	the length of AIDList array buffer

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.6.4.21 int ctls_retrieveApplicationData (IN BYTE * *AID*, IN int *AIDLen*, OUT BYTE * *tlv*, IN_OUT int * *tlvLen*)

Retrieve Application Data by AID

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

Parameters

AID	Name of ApplicationID. Must be between 5 and 16 bytes
AIDLen	the length of AID data buffer.
tlv	The TLV elements of the requested AID
tlvLen	the length of tlv data buffer.

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.6.4.22 int ctls_retrieveCAPK (IN BYTE * *capk*, IN int *capkLen*, OUT BYTE * *key*, IN_OUT int * *keyLen*)

Retrieve Certificate Authority Public Key

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

Parameters

capk	6 bytes CAPK = 5 bytes RID + 1 byte Index
capkLen	the length of capk data buffer
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.
keyLen	the length of key data buffer •

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.6.4.23 int ctls_retrieveCAPKList (OUT BYTE * *keys*, IN_OUT int * *keysLen*)

Retrieve the Certificate Authority Public Key list

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

Parameters

keys	[key1][key2]...[keyn], each key 6 bytes where key = 5 bytes RID + 1 byte index
keysLen	the length of keys data buffer

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.6.4.24 int ctls_retrieveTerminalData (OUT BYTE * *tlv*, IN_OUT int * *tlvLen*)

Retrieve Terminal Data

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

Parameters

tlv	Response returned as a TLV
tlvLen	the length of tlv data buffer

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.6.4.25 int ctls_setApplicationData (IN BYTE * *tlv*, IN int *tlvLen*)

Set Application Data by AID

Sets the Application Data for CTLS as specified by TLV data

Parameters

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

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

Parameters

tlvLen	the length of tlv data buffer

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.6.4.26 int ctls_setCAPK (IN BYTE * *capk*, IN int *capkLen*)

Set Certificate Authority Public Key

Sets the CAPK as specified by the CAKey structure

Parameters

capk	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.
capkLen	the length of capk data buffer

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.6.4.27 int ctls_setConfigurationGroup (IN BYTE * *tlv*, IN int *tlvLen*)

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
tlvLen	the length of tlv data buffer

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.6.4.28 int ctls_setTerminalData (IN BYTE * *tlv*, IN int *tlvLen*)

Set Terminal Data

Sets the Terminal Data for CTLS as specified by the TLV. The first TLV must be Configuration Group Number (Tag FFE4). The terminal global data is group 0, so the first TLV would be FFE40100. Other groups can be defined using this method (1 or greater), and those can be retrieved with ctls_getConfigurationGroup(int group), and deleted with ctls_removeConfigurationGroup(int group). You cannot delete group 0.

Parameters

tlv	TerminalData configuration file
tlvLen	the length of tlv data buffer

Return values

RETURN_CODE	Values can be parsed with device_getIDGStatusCodeString()

12.6.4.29 int ctls_startTransaction (IN double *amount*, IN double *amtOther*, IN int *type*, IN const int *_timeout*, IN BYTE * *tags*, IN int *tagsLen*)

Start CTLS Transaction Request

Authorizes the CTLS transaction for an ICC card

The tags will be returned in the callback routine.

Parameters

amount	Transaction amount value (tag value 9F02) • SEE IMPORTANT NOTE BELOW

amtOther	Other amount value, if any (tag value 9F03) • SEE IMPORTANT NOTE BELOW
type	Transaction type (tag value 9C).
timeout	Timeout value in seconds.
tags	Any other tags to be included in the request. Passed as TLV stream. 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.
tagsLen	The length of tags data buffer.

>>>>>IMPORTANT: parameters for amount and amtOther MUST BE PASSED AS A DOUBLE VALUE WITH DECIMAL POINT. Example, do not pass 1, but instead pass 1.0 or 1.00. Otherwise, results will be unpredictable

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString() Note: if auto poll is on, it will returm the error IDG_P2_STATUS_CODE_COMMAND_NOT_ALLOWED

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 DF01. 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

12.6.4.30 int device_activateTransaction (IN const int *_timeout,* IN BYTE * *tags,* IN int *tagsLen*)

Activate Transaction Request

Authorizes the transaction for an MSR/CTLS/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 a TLV. Example, tag 9F02 with amount 0x000000000100 would be 0x9F0206000000000100 Be sure to include 9F02 (amount)and 9C (transaction type).
tagsLen	The length of tags data buffer.

>>>>>IMPORTANT: parameters for amount and amtOther MUST BE PASSED AS A DOUBLE VALUE WITH DECIMAL POINT. Example, do not pass 1, but instead pass 1.0 or 1.00. Otherwise, results will be unpredictable

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString() Note: if auto poll is on, it will returm the error IDG_P2_STATUS_CODE_COMMAND_NOT_ALLOWED

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 device_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

12.6.4.31 int device_buzzerOnOff ( )

Use this function to make the buzzer beep once

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.6.4.32 int device_cancelTransaction ( )

Cancel Transaction request.

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.6.4.33 int device_cancelTransactionSilent (int *enable*)

Cancel Transaction Silent

Cancel transaction with or without showing the LCD message

Parameters

enable	0: With LCD message 1: Without LCD message

Returns

success or error code. Values can be parsed with device_getIDGStatusCodeString

12.6.4.34 int device_close ( )

Close the device

Returns

RETURN_CODE: 0: success, 0x0A: failed

12.6.4.35 int device_configureButtons (IN BYTE *done,* IN BYTE *swipe,* IN BYTE *delay*)

Configures the buttons on the ViVOpay Vendi reader

Parameters

done	0x01: the Done switch is enabled 0x00: the Done switch is disabled
swipe	0x01: the Swipe Card switch is enabled 0x00: the Swipe Card switch is disabled
delay	an unsigned delay value (<= 30) in seconds

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.6.4.36 int device_controlUserInterface (IN 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 device_getIDGStatusCodeString()

12.6.4.37 int device_deleteDirectory (IN char * *dirName,* IN int *dirNameLen*)

Delete Directory This command deletes an empty directory. For NEO 2 devices, it will delete the directory even the directory is not empty.

Parameters

dirName	Complete path of the directory you want to delete. You do not need to specify the root directory. Indicate subdirectories with a forward slash (/). For NEO 2 devices, to delete the root directory, simply pass "" with 0 for dirNameLen.
dirNameLen	Directory Name Length.

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.6.4.38 int device_deleteFile (IN char * *fileName,* IN int *fileNameLen*)

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 (/).
filenameLen	File Name Length.

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.6.4.39 int device_disableBlueLED ( )

Stops the blue LEDs on the ViVOpay Vendi reader from flashing in left to right sequence and turns the LEDs off, and contactless function is disabled at the same time

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.6.4.40 int device_enableBlueLED (IN BYTE * *data,* IN int *dataLen*)

Use this function to control the blue LED behavior on the Vendi reader

Parameters

data	Sequence data Byte 0 (Cycle): 0 = Cycle once, 1 = Repeat Byte 1 (LEDs): LED State Bitmap Byte 2-3 (Duration): Given in multiples of 10 millisecond Byte 4 (LED): LED State Bitmap Byte 5-6 (Duration): Given in multiples of 10 millisecond Byte 7-24 (Additional LED/Durations): Define up to 8 LED and duration pars

LED State Bitmap: Bit 8: Left blue LED, 0 = off, 1 = on Bit 7: Center Blue LED, 0 = off, 1 = on Bit 6: Right Blue LED0 = off, 1 = on Bit 5: Yellow LED, 0 = off, 1 = on Bit 4: Reserved for future use Bit 3: Reserved for future use Bit 2: Reserved for future use Bit 1: Reserved for future use

Parameters

dataLen	Length of the sequence data: 0 or 4 to 25 bytes

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.6.4.41 int device_enableExternalLCDMessages (IN int *enableExtLCDMsg*)

Enable or disable the external LCD message It will turn off the external LCD messages including EMV transactions. (For the users who only need MSR and/or CTLS transactions.) The function only works for VP5300

Parameters

enableExtLCD-Msg	1 = ON, 0 = OFF

Returns

success or error code. Values can be parsed with device_getIDGStatusCodeString

See Also

ErrorCode

12.6.4.42 int device_enableL100PassThrough (int *enableL100PassThrough*)

Enable L100 Pass Through

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

Parameters

enableL100-PassThrough	1 = pass through ON, 0 = pass through OFF

Returns

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

12.6.4.43 int device_enablePassThrough (int *enablePassThrough*)

Enable Pass Through

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

Parameters

enablePass-Through	1 = pass through ON, 0 = pass through OFF

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.6.4.44 int device_enableRFAntenna (IN int *enableAntenna*)

Enable or disable the RF Antenna

Parameters

enableAntenna	1 = ON, 0 = OFF

Returns

success or error code. Values can be parsed with device_getIDGStatusCodeString

See Also

ErrorCode

12.6.4.45 int device_getButtonConfiguration (OUT BYTE * *done*, OUT BYTE * *swipe*, OUT BYTE * *delay*)

Reads the button configuration from the ViVOpay Vendi reader

Parameters

done	0x01: the Done switch is enabled 0x00: the Done switch is disabled
swipe	0x01: the Swipe Card switch is enabled 0x00: the Swipe Card switch is disabled
delay	an unsigned delay value in seconds

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.6.4.46 int device_getCurrentDeviceType ( )

Get current active device type

Returns

: return the device type defined as DEVICE_TYPE in the IDTDef.h

12.6.4.47 int device_getDeviceMemoryUsageInfo (OUT int * *freeHeapSize,* OUT int * *notFreedBlockCnt,* OUT int * *minEverFreeHeapSize*)

Get Device Memory Usage Information

Parameters

freeHeapSize	Free Heap Size: Available heap size
notFreedBlock-Cnt	Memory Not Freed Block Count: Memory in use block count
minEverFree-HeapSize	Minimum Ever Free Heap Size: The lowest ever available heap size

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.6.4.48 int device_getFirmwareVersion (OUT char * *firmwareVersion*)

DEPRECATED : please use device_getFirmwareVersion_Len(OUT char* firmwareVersion, IN_OUT int *firmware-VersionLen)

Polls device for Firmware Version

Parameters

firmwareVersion	Response returned of Firmware Version; needs to have at least 128 bytes of memory

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.6.4.49 int device_getFirmwareVersion_Len (OUT char * *firmwareVersion,* IN_OUT int * *firmwareVersionLen*)

Polls device for Firmware Version

Parameters

firmwareVersion	Response returned of Firmware Version
firmwareVersion-Len	Length of Firmware Version

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.6.4.50 void device_getIDGStatusCodeString (IN int *returnCode,* OUT char * *despcrition*)

Review the return code description.

Parameters

returnCode	the response result.
description	

Return values

the	string for description of response result

- 0: "no error, beginning task";
- 1: "no response from reader";
- 2: "invalid response data";
  - 01: " Incorrect Header Tag";
  - 02: " Unknown Command";
  - 03: " Unknown Sub-Command";
  - 04: " CRC Error in Frame";
  - 05: " Incorrect Parameter";
  - 06: " Parameter Not Supported";
  - 07: " Mal-formatted Data";
  - 08: " Timeout";
  - 0A: " Failed / NACK";
  - 0B: " Command not Allowed";
  - 0C: " Sub-Command not Allowed";
  - 0D: " Buffer Overflow (Data Length too large for reader buffer)";
  - 0E: " User Interface Event";
  - 10: " Need clear firmware(apply in boot loader only)";
  - 11: " Communication type not supported, VT-1, burst, etc. Need encrypted firmware (apply in boot loader only)";
  - 12: " Secure interface is not functional or is in an intermediate state.";
  - 13: " Data field is not mod 8";
  - 14: " Pad 0x80 not found where expected";
  - 15: " Specified key type is invalid";
  - 16: " Could not retrieve key from the SAM (InitSecureComm)";
  - 17: " Hash code problem";
  - 18: " Could not store the key into the SAM (InstallKey)";

- 19: " Frame is too large";
- 1A: " Unit powered up in authentication state but POS must resend the InitSecureComm command";
- 1B: " The EEPROM may not be initialized because SecCommInterface does not make sense";
- 1C: " Problem encoding APDU Module-Specific Status Codes ";
- 20: " Unsupported Index (ILM) SAM Transceiver error - problem communicating with the SAM (Key Mgr)";
- 21: " Unexpected Sequence Counter in multiple frames for single bitmap (ILM)Length error in data returned from the SAM (Key Mgr) - 22: " Improper bit map (ILM)"; - 23: " Request Online Authorization"; - 24: " ViVOCard3 raw data read successful"; - 25: " Message index not available (ILM) ViVOcomm activate transaction card type (ViVOcomm)"; - 26: " Version Information Mismatch (ILM)"; - 27: " Not sending commands in correct index message index (ILM)"; - 28: " Time out or next expected message not received (ILM)"; - 29: " ILM languages not available for viewing (ILM)"; - 2A: " Other language not supported (ILM)"; - 41: " from 41 to 4F, Module-specific errors for Key Manager";
- 50: " Auto-Switch OK";
- 51: " Auto-Switch failed";
- 70: " Antenna Error 80h Use another card";
- 81: " Insert or swipe card";
- 90: " Data encryption Key does not exist";
- 91: " Data encryption Key KSN exhausted";

12.6.4.51 int device_getKeyStatus (int * *newFormat,* BYTE * *status,* int * *statusLen*)

Get Key Status

Gets the status of loaded keys

Parameters

status	newFormat for Augusta and miniSmartII only 1: new format of key status 0: reserved format for support previous device
status	For L100, Augusta and miniSmartII: When the newFormat is 0, data format as follows. For Augusta and miniSmartII: byte 0: PIN DUKPT Key, Does not support, always 0 byte 1: PIN Master Key, Does not support, always 0 byte 2: PIN Session Key, Does not support, always 0 byte 3: Account/MSR DUKPT Key, 1 Exists, 0 None, 0xFF STOP byte 4: Account/ICC DUKPT Key, 1 Exists, 0 None, 0xFF STOP byte 5: Admin DUKPT Key, 1 Exists, 0 None, 0xFF STOP For L100: byte 0: PIN DUKPT Key byte 1: PIN Master Key byte 2: Standard PIN Session Key byte 3: Desjardins PIN Session Key byte 4: Account/MSR DUKPT Key, 1 Exists, 0 None, 0xFF STOP, Does not support, always 0 byte 5: Account/ICC DUKPT Key, 1 Exists, 0 None, 0xFF STOP, Does not support, always 0 byte 6: Admin DUKPT Key, 1 Exists, 0 None, 0xFF STOP byte 7: Data DUKPT Key, 1 Exists, 0 None, 0xFF STOP byte 8: MAC DUKPT Key, 1 Exists, 0 None, 0xFF STOP

when the newFormat is 1, data format as follows. [Block Length] [KeyStatusBlock1] [KeyStatusBlock2]...[KeyStatusBlockN] Where: [Block Length] is 2 bytes, format is Len_L Len_H, is KeyStatusBlock Number [KeyStatusBlockX> is 4 bytes, format is [Key Index and Key Name] [key slot] [key status]: [Key Index and Key Name] is 1 byte. Please refer to following table 0x14 LCL-KEK to Encrypt Other Keys 0x02 Data encryption Key to Encrypt ICC/MSR 0x05 MAC DUKPT Key for Host-Device - MAC Verification 0x05 MTK DUKPT Key for TTK Self-Test 0x0C RKI-KEK for Remote Key Injection [key slot] is 2 bytes. Range is 0 - 9999 the MTK DUKPT Key slot is 16, the others are all 0 [key status] is 1 byte. 0 - Not Exist 1 - Exist 0xFF - (Stop. Only Valid for DUKPT Key) For NEO2 and SREDKey2: Each unit of three bytes represents one key's parameters (index and slot). Key Name Index (1 byte): 0x14 - LCL-KEK 0x01 - Pin encryption Key (NEO2 only) 0x02 - Data encryption Key 0x05 - MAC DUKPT Key 0x0A - PCI Pairing Key (NEO2 only) Key Slot (2 bytes): Indicate different slots of a certain Key Name Example: slot =5 (0x00 0x05), slot=300 (0x01 0x2C) For BTPay380, slot is always 0 For example, 0x14 0x00 0x00 0x02 0x00 0x00 0x0A 0x00 0x00 will represent [KeyNameIndex=0x14,KeySlot=0x0000], [KeyNameIndex=0x02,KeySlot=0x0000] and [KeyNameIndex=0x0A,KeySlot=0x0000]

Parameters

statusLen	the length of status

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString

12.6.4.52 int device_getL100PassThroughMode ( )

Get L100 Pass Through Mode

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

Returns

RETURN_CODE: return 1 if L100 Pass Through Mode is TRUE, 0 if L100 Pass Through Mode is FALSE

12.6.4.53 int device_getMerchantRecord (IN int *index*, OUT BYTE * *record*)

DEPRECATED : please use device_getMerchantRecord_Len(IN int index, OUT BYTE * record, IN_OUT int *recordLen)

Get Merchant Record

Gets the merchant record for the device.

Parameters

index	Merchant Record index, valid values 1-6
record;	needs to have at least 99 bytes of memory response data from reader. Merchant Record Index: 1 byte enabled: 1 byte Merchant ID: 32 bytes Length of Merchant URL: 1 byte Merchant URL: 64 bytes

Returns

success or error code. Values can be parsed with device_getIDGStatusCodeString()

See Also

ErrorCode

12.6.4.54 int device_getMerchantRecord_Len (IN int *index*, OUT BYTE * *record*, IN_OUT int * *recordLen*)

Get Merchant Record

Gets the merchant record for the device.

Parameters

index	Merchant Record index, valid values 1-6
record	response data from reader. Merchant Record Index: 1 byte enabled: 1 byte Merchant ID: 32 bytes Length of Merchant URL: 1 byte Merchant URL: 64 bytes

recordLen	Length of record

Returns

success or error code. Values can be parsed with device_getIDGStatusCodeString()

See Also

ErrorCode

12.6.4.55 void device_getResponseCodeString (IN int *returnCode,* OUT char * *despcrition*)

Review the return code description.

Parameters

returnCode	the response result.
description	

Return values

the	string for description of response result

- 0: "no error, beginning task";
- 1: "no response from reader";
- 2: "invalid response data";
- 3: "time out for task or CMD";
- 4: "wrong parameter";
- 5: "SDK is doing MSR or ICC task";
- 6: "SDK is doing PINPad task";
- 7: "SDK is doing CTLS task";
- 8: "SDK is doing EMV task";
- 9: "SDK is doing Other task";
- 10: "err response or data";
- 11: "no reader attached";
- 12: "mono audio is enabled";
- 13: "did connection";
- 14: "audio volume is too low";
- 15: "task or CMD be canceled";
- 16: "UF wrong string format";
- 17: "UF file not found";
- 18: "UF wrong file format";
- 19: "Attempt to contact online host failed";
- 20: "Attempt to perform RKI failed";

- 22: "Buffer size is not enough";
- 0X300: "Key Type(TDES) of Session Key is not same as the related Master Key.";
- 0X400: "Related Key was not loaded.";
- 0X500: "Key Same.";
- 0X501: "Key is all zero";
- 0X502: "TR-31 format error";
- 0X702: "PAN is Error Key.";
- 0X705: "No Internal MSR PAN (or Internal MSR PAN is erased timeout)";
- 0X0C01: "Incorrect Frame Tag";
- 0X0C02: "Incorrect Frame Type";
- 0X0C03: "Unknown Frame Type";
- 0X0C04: "Unknown Command";
- 0X0C05: "Unknown Sub-Command";
- 0X0C06: "CRC Error";
- 0X0C07: "Failed";
- 0X0C08: "Timeout";
- 0X0C0A: "Incorrect Parameter";
- 0X0C0B: "Command Not Supported";
- 0X0C0C: "Sub-Command Not Supported";
- 0X0C0D: "Parameter Not Supported / Status Abort Command";
- 0X0C0F: "Sub-Command Not Allowed";
- 0X0D01: "Incorrect Header Tag";
- 0X0D02: "Unknown Command";
- 0X0D03: "Unknown Sub-Command";
- 0X0D04: "CRC Error in Frame";
- 0X0D05: "Incorrect Parameter";
- 0X0D06: "Parameter Not Supported";
- 0X0D07: "Mal-formatted Data";
- 0X0D08: "Timeout";
- 0X0D0A: "Failed / NACK";
- 0X0D0B: "Command not Allowed";
- 0X0D0C: "Sub-Command not Allowed";
- 0X0D0D: "Buffer Overflow (Data Length too large for reader buffer)";
- 0X0D0E: "User Interface Event";
- 0X0D11: "Communication type not supported, VT-1, burst, etc.";
- 0X0D12: "Secure interface is not functional or is in an intermediate state.";

- 0X0D13: "Data field is not mod 8";
- 0X0D14: "Pad - 0X80 not found where expected";
- 0X0D15: "Specified key type is invalid";
- 0X0D1: "Could not retrieve key from the SAM(InitSecureComm)";
- 0X0D17: "Hash code problem";
- 0X0D18: "Could not store the key into the SAM(InstallKey)";
- 0X0D19: "Frame is too large";
- 0X0D1A: "Unit powered up in authentication state but POS must resend the InitSecureComm command";
- 0X0D1B: "The EEPROM may not be initialized because SecCommInterface does not make sense";
- 0X0D1C: "Problem encoding APDU";
- 0X0D20: "Unsupported Index(ILM) SAM Transceiver error - problem communicating with the SAM(Key Mgr)";
- 0X0D2: "Unexpected Sequence Counter in multiple frames for single bitmap(ILM) Length error in data returned from the SAM(Key Mgr)";
- 0X0D22: "Improper bit map(ILM)";
- 0X0D23: "Request Online Authorization";
- 0X0D24: "ViVOCard3 raw data read successful";
- 0X0D25: "Message index not available(ILM) ViVOcomm activate transaction card type(ViVOcomm)";
- 0X0D26: "Version Information Mismatch(ILM)";
- 0X0D27: "Not sending commands in correct index message index(ILM)";
- 0X0D28: "Time out or next expected message not received(ILM)";
- 0X0D29: "ILM languages not available for viewing(ILM)";
- 0X0D2A: "Other language not supported(ILM)";
- 0X0D41: "Unknown Error from SAM";
- 0X0D42: "Invalid data detected by SAM";
- 0X0D43: "Incomplete data detected by SAM";
- 0X0D44: "Reserved";
- 0X0D45: "Invalid key hash algorithm";
- 0X0D46: "Invalid key encryption algorithm";
- 0X0D47: "Invalid modulus length";
- 0X0D48: "Invalid exponent";
- 0X0D49: "Key already exists";
- 0X0D4A: "No space for new RID";
- 0X0D4B: "Key not found";
- 0X0D4C: "Crypto not responding";
- 0X0D4D: "Crypto communication error";
- 0X0D4E: "Module-specific error for Key Manager";

- 0X0D4F: "All key slots are full (maximum number of keys has been installed)";
- 0X0D50: "Auto-Switch OK";
- 0X0D51: "Auto-Switch failed";
- 0X0D90: "Account DUKPT Key not exist";
- 0X0D91: "Account DUKPT Key KSN exausted";
- 0X0D00: "This Key had been loaded.";
- 0X0E00: "Base Time was loaded.";
- 0X0F00: "Encryption Or Decryption Failed.";
- 0X1000: "Battery Low Warning (It is High Priority Response while Battery is Low.)";
- 0X1800: "Send 'Cancel Command' after send 'Get Encrypted PIN' & 'Get Numeric' & 'Get Amount'; - 0X1900: "Press 'Cancel' key after send 'Get Encrypted PIN' & 'Get Numeric' & 'Get Amount';
- 0X30FF: "Security Chip is not connect";
- 0X3000: "Security Chip is deactivation & Device is In Removal Legally State.";
- 0X3101: "Security Chip is activation & Device is In Removal Legally State.";
- 0X5500: "No Admin DUKPT Key.";
- 0X5501: "Admin DUKPT Key STOP.";
- 0X5502: "Admin DUKPT Key KSN is Error.";
- 0X5503: "Get Authentication Code1 Failed.";
- 0X5504: "Validate Authentication Code Error.";
- 0X5505: "Encrypt or Decrypt data failed.";
- 0X5506: "Not Support the New Key Type.";
- 0X5507: "New Key Index is Error.";
- 0X5508: "Step Error.";
- 0X5509: "KSN Error";
- 0X550A: "MAC Error.";
- 0X550B: "Key Usage Error.";
- 0X550C: "Mode Of Use Error.";
- 0X550F: "Other Error.";
- 0X6000: "Save or Config Failed / Or Read Config Error.";
- 0X6200: "No Serial Number.";
- 0X6900: "Invalid Command - Protocol is right, but task ID is invalid.";
- 0X6A01: "Unsupported Command - Protocol and task ID are right, but command is invalid - In this State";
- 0X6A00: "Unsupported Command - Protocol and task ID are right, but command is invalid.";
- 0X6B00: "Unknown parameter in command - Protocol task ID and command are right, but parameter is invalid.";
- 0X6C00: "Unknown parameter in command - Protocol task ID and command are right, but length is out of the requirement.";

- 0X7200: "Device is suspend (MKSK suspend or press password suspend).";
- 0X7300: "PIN DUKPT is STOP (21 bit 1).";
- 0X7400: "Device is Busy.";
- 0XE100: "Can not enter sleep mode";
- 0XE200: "File has existed";
- 0XE300: "File has not existed";
- 0XE313: "IO line low -- Card error after session start";
- 0XE400: "Open File Error";
- 0XE500: "SmartCard Error";
- 0XE600: "Get MSR Card data is error";
- 0XE700: "Command time out";
- 0XE800: "File read or write is error";
- 0XE900: "Active 1850 error!";
- 0XEA00: "Load bootloader error";
- 0XEF00: "Protocol Error- STX or ETX or check error.";
- 0XEB00: "Picture is not exist";
- 0X2C02: "No Microprocessor ICC seated";
- 0X2C06: "no card seated to request ATR";
- 0X2D01: "Card Not Supported, ";
- 0X2D03: "Card Not Supported, wants CRC";
- 0X690D: "Command not supported on reader without ICC support";
- 0X8100: "ICC error time out on power-up";
- 0X8200: "invalid TS character received - Wrong operation step";
- 0X8300: "Decode MSR Error";
- 0X8400: "TriMagII no Response";
- 0X8500: "No Swipe MSR Card";
- 0X8510: "No Financial Card";
- 0X8600: "Unsupported F, D, or combination of F and D";
- 0X8700: "protocol not supported EMV TD1 out of range";
- 0X8800: "power not at proper level";
- 0X8900: "ATR length too long";
- 0X8B01: "EMV invalid TA1 byte value";
- 0X8B02: "EMV TB1 required";
- 0X8B03: "EMV Unsupported TB1 only 00 allowed";
- 0X8B04: "EMV Card Error, invalid BWI or CWI";
- 0X8B06: "EMV TB2 not allowed in ATR";

- 0X8B07: "EMV TC2 out of range";
- 0X8B08: "EMV TC2 out of range";
- 0X8B09: "per EMV96 TA3 must be > - 0XF";
- 0X8B10: "ICC error on power-up";
- 0X8B11: "EMV T=1 then TB3 required";
- 0X8B12: "Card Error, invalid BWI or CWI";
- 0X8B13: "Card Error, invalid BWI or CWI";
- 0X8B17: "EMV TC1/TB3 conflict-";
- 0X8B20: "EMV TD2 out of range must be T=1";
- 0X8C00: "TCK error";
- 0XA304: "connector has no voltage setting";
- 0XA305: "ICC error on power-up invalid (SBLK(IFSD) exchange";
- 0XE301: "ICC error after session start";
- 0XFF00: "Request to go online";
- 0XFF01: "EMV: Accept the offline transaction";
- 0XFF02: "EMV: Decline the offline transaction";
- 0XFF03: "EMV: Accept the online transaction";
- 0XFF04: "EMV: Decline the online transaction";
- 0XFF05: "EMV: Application may fallback to magstripe technology";
- 0XFF06: "EMV: ICC detected tah the conditions of use are not satisfied";
- 0XFF07: "EMV: ICC didn't accept transaction";
- 0XFF08: "EMV: Transaction was cancelled";
- 0XFF09: "EMV: Application was not selected by kernel or ICC format error or ICC missing data error";
- 0XFF0A: "EMV: Transaction is terminated";
- 0XFF0B: "EMV: Other EMV Error";
- 0XFFFF: "NO RESPONSE";
- 0XF002: "ICC communication timeout";
- 0XF003: "ICC communication Error";
- 0XF00F: "ICC Card Seated and Highest Priority, disable MSR work request";
- 0XF200: "AID List / Application Data is not exist";
- 0XF201: "Terminal Data is not exist";
- 0XF202: "TLV format is error";
- 0XF203: "AID List is full";
- 0XF204: "Any CA Key is not exist";
- 0XF205: "CA Key RID is not exist";
- 0XF206: "CA Key Index it not exist";

- 0XF207: "CA Key is full";
- 0XF208: "CA Key Hash Value is Error";
- 0XF209: "Transaction format error";
- 0XF20A: "The command will not be processing";
- 0XF20B: "CRL is not exist";
- 0XF20C: "CRL number exceed max number";
- 0XF20D: "Amount, Other Amount, Trasaction Type are missing";
- 0XF20E: "The Identification of algorithm is mistake";
- 0XF20F: "No Financial Card";
- 0XF210: "In Encrypt Result state, TLV total Length is greater than Max Length";
- 0X1001: "INVALID ARG";
- 0X1002: "FILE_OPEN_FAILED";
- 0X1003: "FILE OPERATION_FAILED";
- 0X2001: "MEMORY_NOT_ENOUGH";
- 0X3002: "SMARTCARD_FAIL";
- 0X3003: "SMARTCARD_INIT_FAILED";
- 0X3004: "FALLBACK_SITUATION";
- 0X3005: "SMARTCARD_ABSENT";
- 0X3006: "SMARTCARD_TIMEOUT";
- 0X3012: "EMV_RESULT_CODE_MSR_CARD_ERROR_FALLBACK";
- 0X5001: "EMV_PARSING_TAGS_FAILED";
- 0X5002: "EMV_DUPLICATE_CARD_DATA_ELEMENT";
- 0X5003: "EMV_DATA_FORMAT_INCORRECT";
- 0X5004: "EMV_NO_TERM_APP";
- 0X5005: "EMV_NO_MATCHING_APP";
- 0X5006: "EMV_MISSING_MANDATORY_OBJECT";
- 0X5007: "EMV_APP_SELECTION_RETRY";
- 0X5008: "EMV_GET_AMOUNT_ERROR";
- 0X5009: "EMV_CARD_REJECTED";
- 0X5010: "EMV_AIP_NOT_RECEIVED";
- 0X5011: "EMV_AFL_NOT_RECEIVED";
- 0X5012: "EMV_AFL_LEN_OUT_OF_RANGE";
- 0X5013: "EMV_SFI_OUT_OF_RANGE";
- 0X5014: "EMV_AFL_INCORRECT";
- 0X5015: "EMV_EXP_DATE_INCORRECT";
- 0X5016: "EMV_EFF_DATE_INCORRECT";

- 0X5017: "EMV_ISS_COD_TBL_OUT_OF_RANGE";
- 0X5018: "EMV_CRYPTOGRAM_TYPE_INCORRECT";
- 0X5019: "EMV_PSE_NOT_SUPPORTED_BY_CARD";
- 0X5020: "EMV_USER_SELECTED_LANGUAGE";
- 0X5021: "EMV_SERVICE_NOT_ALLOWED";
- 0X5022: "EMV_NO_TAG_FOUND";
- 0X5023: "EMV_CARD_BLOCKED";
- 0X5024: "EMV_LEN_INCORRECT";
- 0X5025: "CARD_COM_ERROR";
- 0X5026: "EMV_TSC_NOT_INCREASED";
- 0X5027: "EMV_HASH_INCORRECT";
- 0X5028: "EMV_NO_ARC";
- 0X5029: "EMV_INVALID_ARC";
- 0X5030: "EMV_NO_ONLINE_COMM";
- 0X5031: "TRAN_TYPE_INCORRECT";
- 0X5032: "EMV_APP_NO_SUPPORT";
- 0X5033: "EMV_APP_NOT_SELECT";
- 0X5034: "EMV_LANG_NOT_SELECT";
- 0X5035: "EMV_NO_TERM_DATA";
- 0X5039: "EMV_PIN_ENTRY_TIMEOUT";
- 0X6001: "CVM_TYPE_UNKNOWN";
- 0X6002: "CVM_AIP_NOT_SUPPORTED";
- 0X6003: "CVM_TAG_8E_MISSING";
- 0X6004: "CVM_TAG_8E_FORMAT_ERROR";
- 0X6005: "CVM_CODE_IS_NOT_SUPPORTED";
- 0X6006: "CVM_COND_CODE_IS_NOT_SUPPORTED";
- 0X6007: "NO_MORE_CVM";
- 0X6008: "PIN_BYPASSED_BEFORE";
- 0X7001: "PK_BUFFER_SIZE_TOO_BIG";
- 0X7002: "PK_FILE_WRITE_ERROR";
- 0X7003: "PK_HASH_ERROR";
- 0X8001: "NO_CARD_HOLDER_CONFIRMATION";
- 0X8002: "GET_ONLINE_PIN";
- 0XD000: "Data not exist";
- 0XD001: "Data access error";
- 0XD100: "RID not exist";

- 0XD101: "RID existed";
- 0XD102: "Index not exist";
- 0XD200: "Maximum exceeded";
- 0XD201: "Hash error";
- 0XD205: "System Busy";
- 0X0E01: "Unable to go online";
- 0X0E02: "Technical Issue";
- 0X0E03: "Declined";
- 0X0E04: "Issuer Referral transaction";
- 0X0F01: "Decline the online transaction";
- 0X0F02: "Request to go online";
- 0X0F03: "Transaction is terminated";
- 0X0F05: "Application was not selected by kernel or ICC format error or ICC missing data error";
- 0X0F07: "ICC didn't accept transaction";
- 0X0F0A: "Application may fallback to magstripe technology";
- 0X0F0C: "Transaction was cancelled";
- 0X0F0D: "Timeout";
- 0X0F0F: "Other EMV Error";
- 0X0F10: "Accept the offline transaction";
- 0X0F11: "Decline the offline transaction";
- 0X0F21: "ICC detected tah the conditions of use are not satisfied";
- 0X0F22: "No app were found on card matching terminal configuration";
- 0X0F23: "Terminal file does not exist";
- 0X0F24: "CAPK file does not exist";
- 0X0F25: "CRL Entry does not exist";
- 0X0FFE: "code when blocking is disabled";
- 0X0FFF: "code when command is not applicable on the selected device";
- 0XF005: "ICC Encrypted C-APDU Data Structure Length Error Or Format Error.";
- 0XBBE0: "CM100 Success";
- 0XBBE1: "CM100 Parameter Error";
- 0XBBE2: "CM100 Low Output Buffer";
- 0XBBE3: "CM100 Card Not Found";
- 0XBBE4: "CM100 Collision Card Exists";
- 0XBBE5: "CM100 Too Many Cards Exist";
- 0XBBE6: "CM100 Saved Data Does Not Exist";
- 0XBBE8: "CM100 No Data Available";

- 0XBBE9: "CM100 Invalid CID Returned";
- 0XBBEA: "CM100 Invalid Card Exists";
- 0XBBEC: "CM100 Command Unsupported";
- 0XBBED: "CM100 Error In Command Process";
- 0XBBEE: "CM100 Invalid Command";
- 0X9031: "Unknown command";
- 0X9032: "Wrong parameter (such as the length of the command is incorrect)";
- 0X9038: "Wait (the command couldnt be finished in BWT)";
- 0X9039: "Busy (a previously command has not been finished)";
- 0X903A: "Number of retries over limit";
- 0X9040: "Invalid Manufacturing system data";
- 0X9041: "Not authenticated";
- 0X9042: "Invalid Master DUKPT Key";
- 0X9043: "Invalid MAC Key";
- 0X9044: "Reserved for future use";
- 0X9045: "Reserved for future use";
- 0X9046: "Invalid DATA DUKPT Key";
- 0X9047: "Invalid PIN Pairing DUKPT Key";
- 0X9048: "Invalid DATA Pairing DUKPT Key";
- 0X9049: "No nonce generated";
- 0X9949: "No GUID available. Perform getVersion first.";
- 0X9950: "MAC Calculation unsuccessful. Check BDK value.";
- 0X904A: "Not ready";
- 0X904B: "Not MAC data";
- 0X9050: "Invalid Certificate";
- 0X9051: "Duplicate key detected";
- 0X9052: "AT checks failed";
- 0X9053: "TR34 checks failed";
- 0X9054: "TR31 checks failed";
- 0X9055: "MAC checks failed";
- 0X9056: "Firmware download failed";
- 0X9060: "Log is full";
- 0X9061: "Removal sensor unengaged";
- 0X9062: "Any hardware problems";
- 0X9070: "ICC communication timeout";
- 0X9071: "ICC data error (such check sum error)";
- 0X9072: "Smart Card not powered up";

12.6.4.56 int device_getSDKWaitTime ( )

Get SDK Wait Time

Get the SDK wait time for transactions

Returns

SDK wait time in seconds

12.6.4.57 int device_getThreadStackSize ( )

Get Thread Stack Size

Get the stack size setting for newly created threads

Returns

Thread Stack Size

12.6.4.58 int device_getTransactionResults (IDTMSRData * *cardData*)

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

Parameters

cardData	The transaction results

Returns

success or error code. Values can be parsed with device_getIDGStatusCodeString

See Also

ErrorCode

12.6.4.59 int device_init ( )

Initial the device by USB

It will detect the device and trying connect.

The connect status can be checked by device_isConnected().

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.6.4.60 int device_isAttached (int *deviceType*)

Check if the device is attached to the USB port The function device_init() must be called before this function.

Parameters

deviceType,the	device type of the USB device

Returns

1 if the device is attached, or 0 if the device is not attached

12.6.4.61 int device_isConnected ( )

Check the device conntected status

Returns

DEVICE_DISCONNECT=0, or DEVICE_CONNECTED = 1

12.6.4.62 int device_lcdDisplayClear ( )

Use this function to clear the LCD display

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.6.4.63 int device_lcdDisplayLine1Message (IN BYTE * *message,* IN int *messageLen*)

Use this function to display text on the LCD display. On the Vendi reader the LCD is a 2-line character display.

Parameters

message	Valid messages for the first line of text are between 1 and 16 printable characters long. If the text message is greater than 16 bytes but not more than 32 bytes, byte 17 and onward are displayed as a second row of text. All messages are left justified on the LCD display.
messageLen	Length of the message: 1 to 32 bytes

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.6.4.64 int device_lcdDisplayLine2Message (IN BYTE * *message,* IN int *messageLen*)

Use this function to display the message on line 2 of the LCD display. On the Vendi reader the LCD is a 2-line character display.

Parameters

message	Valid messages are between 1 and 16 printable characters long. All messages are left justified on the LCD display.
messageLen	Length of the message: 1 to 16 bytes

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.6.4.65 int device_listDirectory (IN char * *directoryName,* IN int *directoryNameLen,* IN int *recursive,* IN int *onSD,* OUT char * *directory,* IN_OUT int * *directoryLen*)

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
directoryNameLen	Directory Name Length. If null, root directory is listed
recursive	Included sub-directories
onSD	TRUE = use flash storage The returned directory information The returned directory information length

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.6.4.66 int device_pingDevice ( )

Ping Device

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

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.6.4.67 int device_pollForToken (IN int *timeout*, OUT BYTE * *respData*, IN_OUT int * *respDataLen*)

Poll for Token

Polls for a PICC

Parameters

timeout	timeout in milliseconds, must be multiple of 10 milliseconds. 30, 120, 630, or 1150 for example.
respData	Response data will be stored in respData. 1 byte of card type, and the Serial Number (or the UID) of the PICC if available.
respDataLen	Length of systemCode.

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.6.4.68 int device_queryFile (IN char * *directoryName*, IN int *directoryNameLen*, IN char * *fileName*, IN int *fileNameLen*, OUT int * *isExist*, OUT BYTE * *timeStamp*, IN_OUT int * *timeStampLen*, OUT char * *fileSize*, IN_OUT int * *fileSizeLen*)

Query File This command checks if the specified file exists in NAND Flash..

Parameters

directoryName	Directory name string. No longer than 32 bytes. ASCII string, terminated by 0x00.
directoryNameLen	Directory Name Length.

fileName	File name string. No longer than 32 bytes. ASCII string, terminated by 0x00.
fileNameLen	File Name Length.
isExist	File exists: 1, File not exists 0.
timeStamp	Latest time stamp of the file. 6 bytes BCD code if the file exists.
timeStampLen	Length of timeStamp. 6 if the file exists, 0 if the file does not exist.
fileSize	Zero-terminated ASCII string of the file size.
fileSizeLen	Length of filesSize.

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.6.4.69 void device_registerCameraCallBk (pCMR_callBack *pCMRf*)

To register the camera callback function to get the image data. (Pass NULL to disable the callback.)

12.6.4.70 void device_registerCardStatusFrontSwitchCallBk (pCSFS_callBack *pCSFSf*)

To register the card status and front switch callback function to get status. (Pass NULL to disable the callback.)

12.6.4.71 void device_registerFWCallBk (pFW_callBack *pFWf*)

To register the firmware update callback function to get the firmware update processing response. (Pass NULL to disable the callback.)

12.6.4.72 int device_SendDataCommandNEO (IN int *cmd,* IN int *subCmd,* IN BYTE * *data,* IN int *dataLen,* OUT BYTE * *response,* IN_OUT int * *respLen*)

Send a Command to NEO device

Sends a command to the NEO device .

Parameters

cmd	command to execute.
subCmd,sub	command to execute.
data	buffer of NEO command data.
dataLen,the	length of the buffer data.
response	Response data
respLen,the	length of Response data

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.6.4.73 int device_setBurstMode (IN BYTE *mode*)

Send Burst Mode

Sets the burst mode for the device.

Parameters

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

Returns

success or error code. Values can be parsed with device_getIDGStatusCodeString

See Also

ErrorCode

12.6.4.74 int device_setCancelTransactionMode (int *mode*)

Set Cancel Transaction Mode

Set the cancel transaction mode to be with or without LCD message

Parameters

mode	0: With LCD message 1: Without LCD message

Returns

success or error code. 1: Success, 0: Failed

12.6.4.75 int device_setConfigPath (const char * *path*)

Set the path to the config xml file(s) if any

Parameters

path	The path to the config xml files (such as "NEO2_Devices.xml" which contains the information of NEO2 devices). Only need to specify the path to the folder which contains the config files. File names are not needed. The maximum length of path is 200 characters including the '\0' at the end.

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.6.4.76 int device_setCurrentDevice (int *deviceType*)

Sets the current device to talk to

The connect status can be checked by device_isConnected().

Parameters

deviceType	Device to connect to
	
	`enum DEVICE_TYPE`
	`{`
	`IDT_DEVICE_UNKNOWN=0,`
	`IDT_DEVICE_AUGUSTA_HID,`
	`IDT_DEVICE_AUGUSTA_KB,`
	`IDT_DEVICE_AUGUSTA_S_HID,`
	`IDT_DEVICE_AUGUSTA_S_KB,`
	`IDT_DEVICE_AUGUSTA_S_TTK_HID,`
	`IDT_DEVICE_SPECTRUM_PRO,`
	`IDT_DEVICE_MINISMART_II,`
	`IDT_DEVICE_L100,`
	`IDT_DEVICE_UNIPAY,`
	`IDT_DEVICE_UNIPAY_I_V,`
	`IDT_DEVICE_VP3300_AJ,`
	`IDT_DEVICE_KIOSK_III,`
	`IDT_DEVICE_KIOSK_III_S,`
	`IDT_DEVICE_PIP_READER,`
	`IDT_DEVICE_VENDI,`
	`IDT_DEVICE_VP3300_USB,`
	`IDT_DEVICE_UNIPAY_I_V_TTK,`
	`IDT_DEVICE_VP3300_BT,`
	`IDT_DEVICE_VP8800,`
	`IDT_DEVICE_SREDKEY2_HID,`
	`IDT_DEVICE_SREDKEY2_KB,`
	`IDT_DEVICE_NEO2,`
	`IDT_DEVICE_MINISMART_II_COM = IDT_DEVICE_NEO2+5,`
	`IDT_DEVICE_SPECTRUM_PRO_COM,`
	`IDT_DEVICE_KIOSK_III_COM,`
	`IDT_DEVICE_KIOSK_III_S_COM,`
	`IDT_DEVICE_VP3300_COM,`
	`IDT_DEVICE_NEO2_COM,`
	`IDT_DEVICE_MAX_DEVICES = IDT_DEVICE_NEO2_COM+5`
	`};`

Returns

RETURN_CODE: 1: success, 0: failed

12.6.4.77 int device_setMerchantRecord (int *index*, int *enabled*, char * *merchantID*, char * *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 device_getIDGStatusCodeString()

12.6.4.78 int device_setNEO2DevicesConfigs (IN const char * *configs*, IN int *len*)

Pass the content of the config xml file ("NEO2_Devices.xml") as a string to the SDK instead of reading the config xml file by the SDK It needs to be called before device_init(), otherwise the SDK will try to read the config xml file.

Parameters

configs	The content read from the config xml file ("NEO2_Devices.xml" which contains the information of NEO2 devices).
len	The length of the string configs. The maximum length is 5000 bytes.

12.6.4.79 int device_setPollMode (IN BYTE *mode*)

Set 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 device_getIDGStatusCodeString()

12.6.4.80 void device_setSDKWaitTime (int *waitTime*)

Set SDK Wait Time

Set the SDK wait time for transactions

Parameters

waitTime	The SDK wait time for transaction in seconds

12.6.4.81 void device_setThreadStackSize (int *threadSize*)

Set Thread Stack Size

Set the stack size setting for newly created threads

12.6.4.82 void device_setTransactionExponent (int *exponent*)

Sets the transaction exponent to be used with device_startTransaction. Default value is 2

Parameters

exponent,The	exponent to use when calling device_startTransaction

12.6.4.83 int device_startListenNotifications ( )

Start Listen Notifications This function enables Card Status and Front Switch notifications.

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.6.4.84 int device_startQRCodeScan (IN int *_timeout*)

Start QR Code Scanning

Enables QR Code scanning, waiting for the QR code.

Parameters

timeout	QR Code Scan Timeout Value. Between 30 and 65536 seconds.

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString() Note: if auto poll mode is on, it will return command not allowed error

12.6.4.85 int device_startTakingPhoto (IN int *_timeout*)

Start Taking Photo

Enables the camera to take a photo.

Parameters

timeout	Photo taking Timeout Value. Between 30 and 65536 seconds.

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString() Note: if auto poll mode is on, it will return command not allowed error

12.6.4.86 int device_startTransaction (IN double *amount,* IN double *amtOther,* IN int *type,* IN const int *_timeout,* IN BYTE * *tags,* IN int *tagsLen*)

Start Transaction Request

Authorizes the transaction for an MSR/CTLS/ICC card

The tags will be returned in the callback routine.

Parameters

amount	Transaction amount value (tag value 9F02) • SEE IMPORTANT NOTE BELOW
amtOther	Other amount value, if any (tag value 9F03) • SEE IMPORTANT NOTE BELOW
type	Transaction type (tag value 9C).
timeout	Timeout value in seconds.
tags	Any other tags to be included in the request. Passed as TLV. 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.
tagsLen	The length of tags data buffer.

>>>>>IMPORTANT: parameters for amount and amtOther MUST BE PASSED AS A DOUBLE VALUE WITH DECIMAL POINT. Example, do not pass 1, but instead pass 1.0 or 1.00. Otherwise, results will be unpredictable

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString() Note: if auto poll is on, it will returm the error IDG_P2_STATUS_CODE_COMMAND_NOT_ALLOWED

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 device_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

12.6.4.87 int device_stopListenNotifications ( )

Stop Listen Notifications This function disables Card Status and Front Switch notifications.

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.6.4.88 int device_stopQRCodeScan ( )

Stop QR Code Scanning Cancels QR Code scanning request.

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.6.4.89 int device_stopTakingPhoto ( )

Stop Taking Photo Cancels Photo Taking request.

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.6.4.90 void device_toSDCard (int *forSDCard*)

To SD Card

Set the destination of the file or directory function

Parameters

forSDCard	0: for internal memory, 1: for SD card

12.6.4.91 int device_transferFile (IN char * *fileName,* IN int *fileNameLen,* IN BYTE * *file,* IN int *fileLen*)

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 (/).
filenameLen	File Name Length.
file	The data file.
fileLen	File Length.

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.6.4.92 int device_turnOffYellowLED ( )

Use this function to 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_getIDGStatusCodeString()

12.6.4.93 int device_turnOnYellowLED ( )

Use this function to 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_getIDGStatusCodeString()

12.6.4.94 int device_updateFirmware (IN BYTE * *firmwareData,* IN int *firmwareDataLen,* IN char * *firmwareName,* IN int *encryptionType,* IN BYTE * *keyBlob,* IN int *keyBlobLen*)

Update Firmware Updates the firmware of NEO 2 devices.

Parameters

firmwareData	Signed binary data of a firmware file provided by IDTech
firmwareData-Len	Length of firmwareData
firmwareName	Firmware name. • For example "VP5300_v1.00.023.0167.S_Test.fm"
encryptionType	Encryption type • 0 : Plaintext • 1 : TDES ECB, PKCS#5 padding • 2 : TDES CBC, PKCS#5, IV is all 0
keyBlob	Encrypted firmware session key blob, TR-31 Rev B, wrapped by FW Key (Optional, none if firmware is plaintext)
keyBlobLen	Length of keyBlob

Returns

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

Firmware update status is returned in the callback with the following values: sender = device type state = DEVICE-_FIRMWARE_UPDATE current block total blocks ResultCode:

- RETURN_CODE_DO_SUCCESS = Firmware Update Completed Successfully
- RETURN_CODE_BLOCK_TRANSFER_SUCCESS = Current block transferred successfully
- Any other return code represents an error condition

12.6.4.95 int emv_activateTransaction (IN int *timeout,* IN BYTE * *tags,* IN int *tagsLen,* IN int *forceOnline*)

Activate 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	Tags to be included in the request. Passed as a TLV stream. Example, tag 9F02 with amount 0x000000000100 would be 0x9F0206000000000100
tagsLen	Length of tags
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 = DFEE1A079F029-F36959F37

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.6.4.96 void emv_allowFallback (IN int *allow*)

Allow fallback for EMV transactions. Default is TRUE

Parameters

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

12.6.4.97 int emv_authenticateTransaction (IN BYTE * *updatedTLV,* IN int *updatedTLVLen*)

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.

Parameters

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, 9F37 to be included in response = DFEE1A079F029F36959F37

Parameters

updatedTLVLen	

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.6.4.98 int emv_authenticateTransactionWithTimeout (IN int *timeout,* IN BYTE * *updatedTLV,* IN int *updatedTLVLen*)

Authenticate EMV Transaction Request with Timeout

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.

Parameters

timeout	Timeout value in seconds.
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, 9F37 to be included in response = DFEE1-A079F029F36959F37
updatedTLVLen	

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.6.4.99 int emv_cancelTransaction ( )

Cancel EMV Transaction

Cancels the currently executing EMV transaction.

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.6.4.100 int emv_completeTransaction (IN int *commError*, IN BYTE * *authCode*, IN int *authCodeLen*, IN BYTE * *iad*, IN int *iadLen*, IN BYTE * *tlvScripts*, IN int *tlvScriptsLen*, IN BYTE * *tlv*, IN int *tlvLen*)

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(1) if host was unreachable, or FALSE(0) 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
authCodeLen	the length of authCode
iad	Issuer Authentication Data, if any. Example 0x11223344556677883030 (tag value 91).
iadLen	the length of iadLen
tlvScripts	71/72 scripts, if any
tlvScriptsLen	the length of tlvScriptsLen
tlv	Additional TLV data to return with transaction results (if any)
tlvLen	the length of tlv

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.6.4.101 int emv_getAutoAuthenticateTransaction ( )

Gets auto authenticate value for EMV transactions.

Returns

RETURN_CODE: TRUE = auto authenticate, FALSE = manually authenticate

12.6.4.102 int emv_getAutoCompleteTransaction ( )

Gets auto complete value for EMV transactions.

Returns

RETURN_CODE: TRUE = auto complete, FALSE = manually complete

12.6.4.103 int emv_getEMVConfigurationCheckValue (OUT BYTE * *checkValue,* IN_OUT int * *checkValueLen*)

Get EMV Kernel configuration check value info

Parameters

checkValue	Response returned of Kernel configuration check value info
checkValueLen	the length of checkValue

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.6.4.104 int emv_getEMVKernelCheckValue (OUT BYTE * *checkValue,* IN_OUT int * *checkValueLen*)

Get EMV Kernel check value info

Parameters

checkValue	Response returned of Kernel check value info
checkValueLen	the length of checkValue

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString

12.6.4.105 int emv_getEMVKernelVersion (OUT char * *version*)

DEPRECATED : please use emv_getEMVKernelVersion_Len(OUT char* version, IN_OUT int *versionLen)

Polls device for EMV Kernel Version

Parameters

version	Response returned of Kernel Version; needs to have at least 128 bytes of memory.

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.6.4.106 int emv_getEMVKernelVersion_Len (OUT char * *version,* IN_OUT int * *versionLen*)

Polls device for EMV Kernel Version

Parameters

version	Response returned of Kernel Version
versionLen	Length of version

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.6.4.107 void emv_registerCallBk (pEMV_callBack *pEMVf*)

To register the emv callback function to get the EMV processing response. (Pass NULL to disable the callback.)

12.6.4.108 int emv_removeAllApplicationData ( )

Remove All Application Data

Removes all the Application Data

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.6.4.109 int emv_removeAllCAPK ( )

Remove All Certificate Authority Public Key

Removes all the CAPK

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.6.4.110 int emv_removeAllCRL ( )

Remove All Certificate Revocation List Entries

Removes all CRLEntry entries

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.6.4.111 int emv_removeApplicationData (IN BYTE * *AID,* IN int *AIDLen*)

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

AIDLen	the length of AID data buffer

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.6.4.112 int emv_removeCAPK (IN BYTE * *capk*, IN int *capkLen*)

Remove Certificate Authority Public Key

Removes the CAPK as specified by the RID/Index

Parameters

capk	6 byte CAPK = 5 bytes RID + 1 byte INDEX
capkLen	the length of capk data buffer

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.6.4.113 int emv_removeCRL (IN BYTE * *list*, IN int *lsLen*)

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
lssLen	the length of list data buffer

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.6.4.114 int emv_retrieveAIDList (OUT BYTE * *AIDList*, IN_OUT int * *AIDListLen*)

Retrieve AID list

Returns all the AID names installed on the terminal for CTLS. .

Parameters

AIDList	array of AID name byte arrays
AIDListLen	the length of AIDList array buffer

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.6.4.115 int emv_retrieveApplicationData (IN BYTE * *AID*, IN int *AIDLen*, OUT BYTE * *tlv*, IN_OUT int * *tlvLen*)

Retrieve Application Data by AID

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

Parameters

AID	Name of ApplicationID. Must be between 5 and 16 bytes
AIDLen	the length of AID data buffer.
tlv	The TLV elements of the requested AID
tlvLen	the length of tlv data buffer.

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.6.4.116 int emv_retrieveCAPK (IN BYTE * *capk*, IN int *capkLen*, OUT BYTE * *key*, IN_OUT int * *keyLen*)

Retrieve Certificate Authority Public Key

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

Parameters

capk	6 bytes CAPK = 5 bytes RID + 1 byte Index
capkLen	the length of capk data buffer
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.

keyLen	the length of key data buffer

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.6.4.117 int emv_retrieveCAPKList (OUT BYTE * *keys*, IN_OUT int * *keysLen*)

Retrieve the Certificate Authority Public Key list

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

Parameters

keys	[key1][key2]...[keyn], each key 6 bytes where key = 5 bytes RID + 1 byte index
keysLen	the length of keys data buffer

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.6.4.118 int emv_retrieveCRL (OUT BYTE * *list*, IN_OUT int * *lssLen*)

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
lssLen	the length of list data buffer

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.6.4.119 int emv_retrieveTerminalData (OUT BYTE * *tlv*, IN_OUT int * *tlvLen*)

Retrieve Terminal Data

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

Parameters

tlv	Response returned as a TLV
tlvLen	the length of tlv data buffer

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.6.4.120 int emv_retrieveTransactionResult (IN BYTE * *tags*, IN int *tagsLen*, OUT IDTTransactionData * *cardData*)

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
tagsLen	Length of tag list
cardData	All requested tags returned as unencrypted, encrypted and masked TLV data in IDTTransactionData object

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.6.4.121 int emv_setApplicationData (IN BYTE * *name*, IN int *nameLen*, IN BYTE * *tlv*, IN int *tlvLen*)

Set Application Data by AID

Sets the Application Data as specified by the application name and TLV data

Parameters

name	Application name, 10-32 ASCII hex characters representing 5-16 bytes Example "a0000000031010"
nameLen	the length of name data buffer of Application name,
tlv	Application data in TLV format
tlvLen	the length of tlv data buffer

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.6.4.122 int emv_setApplicationDataTLV (IN BYTE * *tlv*, IN int *tlvLen*)

Set Application Data by TLV

Sets the Application Data as specified by the 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) Example valid TLV, for Group #2, AID a0000000035010-: "dfee2d01029f0607a0000000051010ffe10101ffe50110ffe30114ffe20106"
tlvLen	the length of tlv data buffer

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.6.4.123 void emv_setAutoAuthenticateTransaction (IN int *authenticate*)

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

Parameters

authenticate	TRUE = auto authenticate, FALSE = manually authenticate

12.6.4.124 void emv_setAutoCompleteTransaction (IN int *complete*)

Enables complete for EMV transactions. If a emv_authenticateTranaction results in code 0x0004 (go online), then emv_completeTransaction can automatically execute if parameter is set to TRUE

Parameters

complete	TRUE = auto complete, FALSE = manually complete

12.6.4.125 int emv_setCAPK (IN BYTE * *capk,* IN int *capkLen*)

Set Certificate Authority Public Key

Sets the CAPK as specified by the CAKey structure

Parameters

capk	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.

capkLen	the length of capk data buffer

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.6.4.126 int emv_setCRL (IN BYTE * *list*, IN int *lsLen*)

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]
lsLen	the length of list data buffer

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.6.4.127 int emv_setTerminalData (IN BYTE * *tlv*, IN int *tlvLen*)

Set Terminal Data

Sets the Terminal Data for CTLS as specified by the TLV. The first TLV must be Configuration Group Number (Tag FFE4). The terminal global data is group 0, so the first TLV would be FFE40100. Other groups can be defined using this method (1 or greater), and those can be retrieved with emv_getConfigurationGroup(int group), and deleted with emv_removeConfigurationGroup(int group). You cannot delete group 0.

Parameters

tlv	TerminalData configuration file
tlvLen	the length of tlv data buffer

Return values

RETURN_CODE	Values can be parsed with device_getIDGStatusCodeString()

12.6.4.128 int emv_setTerminalMajorConfiguration (IN int *configuration*)

Sets the terminal major configuration in ICS .

Parameters

configuration	A configuration value, range 1-23 • 1 = 1C • 2 = 2C • 3 = 3C • 4 = 4C • 5 = 5C ... • 23 = 23C

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.6.4.129 void emv_setTransactionParameters (IN double *amount*, IN double *amtOther*, IN int *type*, IN int *timeout*, IN BYTE * *tags*, IN int *tagsLen*)

Set EMV Transaction Parameters

Set the parameters to be used on EMV transactions for an ICC card when Auto Poll is on

The tags will be returned in the callback routine.

Parameters

amount	Transaction amount value (tag value 9F02)
amtOther	Other amount value, if any (tag value 9F03)
type	Transaction type (tag value 9C).
timeout	Timeout value in seconds.
tags	Any other tags to be included in the request (Maximum Length = 500 bytes). Passed as a string. Example, tag 9F02 with amount 0x000000000100 would be "9F0206000000000100" 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. 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 = DFEE1A079F029F36959F37
tagsLen	the length of tags

12.6.4.130 int emv_startTransaction (IN double *amount*, IN double *amtOther*, IN int *exponent*, IN int *type*, IN int *timeout*, IN BYTE * *tags*, IN int *tagsLen*, IN int *forceOnline*)

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) - SEE IMPORTANT NOTE BELOW
amtOther	Other amount value, if any (tag value 9F03) - SEE IMPORTANT NOTE BELOW
exponent	Number of characters after decimal point
type	Transaction type (tag value 9C).
timeout	Timeout value in seconds.
tags	Any other tags to be included in the request. Passed as a TLV stream. Example, tag 9F02 with amount 0x000000000100 would be 0x9F0206000000000100 If tags 9F02 (amount),9-F03 (other amount), or 9C (transaction type) are included, they will take priority over these values supplied as individual parameters to this method.
tagsLen	Length of tags
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 = DFEE1A079F029F36959F37

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString >>>>>IMPORTANT: parameters for amount and amtOther MUST BE PASSED AS A DOUBLE VALUE WITH DECIMAL POINT. Example, do not pass 1, but instead pass 1.0 or 1.00. Otherwise, results will be unpredictable

12.6.4.131 int felica_authentication (IN BYTE * *key*, IN int *keyLen*)

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

Parameters

key	16-byte key used for MAC generation of Read or Write with MAC
keyLen	length of key, must be 16 bytes

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.6.4.132 int felica_poll (IN BYTE * *systemCode*, IN int *systemCodeLen*, OUT BYTE * *respData*, OUT int * *respDataLen*)

FeliCa Poll for Card

Polls for a Felica Card

Parameters

systemCode	System Code.
systemCodeLen	Length of systemCode. Must be 2 bytes
respData	response data will be stored in respData. Poll response as explained in FeliCA Lite-S User's Manual
respDataLen	Length of systemCode.

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.6.4.133 int felica_read (IN BYTE * *serviceCodeList*, IN int *serviceCodeListLen*, IN int *blockCnt*, IN BYTE * *blockList*, IN int *blockListLen*, OUT BYTE * *blockData*, OUT int * *blockDataLen*)

FeliCa Read

Reads up to 4 blocks.

Parameters

serviceCodeList	Service Code List. Each service code in Service Code List = 2 bytes of data
serviceCodeList-Len	Length of serviceCodeList
blockCnt	Number of blocks in blockList. Maximum 4 block requests
blockList	Block to read. Each block in blockList = 2 or 3 bytes of data.
blockListLen	Length of blockList.
blockData	Blocks read will be stored in blockData. Each block 16 bytes.
blockDataLen	Length of blockData.

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.6.4.134 int felica_readWithMac (IN int *blockCnt*, IN BYTE * *blockList*, IN int *blockListLen*, OUT BYTE * *blockData*, OUT int * *blockDataLen*)

FeliCa Read with MAC Generation

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

Parameters

blockCnt	Number of blocks in blockList. Maximum 3 block requests
blockList	Block to read. Each block in blockList = 2 or 3 bytes of data.
blockListLen	Length of blockList.
blockData	Blocks read will be stored in blockData. Each block is 16 bytes.
blockDataLen	Length of blockData.

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.6.4.135 int felica_requestService (IN BYTE * *nodeCode*, IN int *nodeCodeLen*, OUT BYTE * *respData*, OUT int * *respDataLen*)

FeliCa Request Service

Request Service for a Felica Card

Parameters

nodeCode	Node Code List. Each node 2 bytes
nodeCodeLen	Length of nodeCode.
respData	response data will be stored in respData. Response as explained in FeliCA Lite-S User's Manual
respDataLen	Length of respData.

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.6.4.136 int felica_SendCommand (IN BYTE * *command*, IN int *commandLen*, OUT BYTE * *respData*, OUT int * *respDataLen*)

FeliCa Send Command

Send a Felica Command

Parameters

command	Command data from settlement center to be sent to felica card
commandLen	Length of command data
respData	Response data from felica card.
respDataLen	Length of respData.

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.6.4.137 int felica_write (IN BYTE * *serviceCodeList*, IN int *serviceCodeListLen*, IN int *blockCnt*, IN BYTE * *blockList*, IN int *blockListLen*, IN BYTE * *blockData*, IN int *blockDataLen*, OUT BYTE * *statusFlag*, OUT int * *statusFlagLen*)

FeliCa Write

Writes a block

Parameters

serviceCodeList	Service Code List. Each service code in Service Code List = 2 bytes of data
serviceCodeList-Len	Length of serviceCodeList
blockCnt	Number of blocks in blockList. Currently only support 1 block.
blockList	Block list. Each block in blockList = 2 or 3 bytes of data.
blockListLen	Length of blockData.
blockData	Block to write.
blockDataLen	Length of blockData. Must be 16 bytes.
respData	If successful, the Status Flag (2 bytes) is stored in respData.resData. Status flag response as explained in FeliCA Lite-S User's Manual, Section 4.5

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.6.4.138 int felica_writeWithMac (IN BYTE *blockNum*, IN BYTE * *blockData*, IN int *blockDataLen*)

FeliCa Write with MAC Generation

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

Parameters

blockNum	Number of block
blockData	Block to write.
blockDataLen	Length of blockData. Must be 16 bytes.

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.6.4.139 int icc_exchangeAPDU (IN BYTE * *c_APDU*, IN int *cLen*, OUT BYTE * *reData*, IN_OUT int * *reLen*)

Exchange APDU with plain text For Non-SRED Augusta Only

Sends an APDU packet to the ICC. If successful, response is the APDU data in response parameter.

Parameters

c_APDU	APDU data packet
cLen	APDU data packet length
reData	Unencrypted APDU response
reLen	Unencrypted APDU response data length

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString

12.6.4.140 int icc_getICCReaderStatus (OUT 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_getIDGStatusCodeString

12.6.4.141 int icc_powerOffICC ( )

Power Off ICC

Powers down the ICC

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

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

12.6.4.142 int icc_powerOnICC (OUT BYTE * *ATR,* IN_OUT int * *inLen*)

Power On ICC

Power up the currently selected microprocessor card in the ICC reader

Parameters

ATR,the	ATR data response when succeeded power on ICC,
inLen,the	length of ATR data

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.6.4.143 int lcd_addButton (IN char * *screenName,* IN int *screenNameLen,* IN char * *buttonName,* IN int *buttonNameLen,* IN BYTE *type,* IN BYTE *alignment,* IN int *xCord,* IN int *yCord,* IN char * *label,* IN int *labelLen,* OUT IDTLCDItem * *returnItem*)

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
screenNameLen	Length of screenName
buttonName	Button name that will be the target of add button
buttonNameLen	Length of buttonName
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
label	Label to show on the button. Required for Large/Medium buttons. Not used for Invisible buttons.
labelLen	Length of label
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

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.6.4.144 int lcd_addEthernet (IN char * *screenName*, IN int *screenNameLen*, IN char * *objectName*, IN int *objectNameLen*, IN BYTE *alignment*, IN int *xCord*, IN int *yCord*, OUT IDTLCDItem * *returnItem*)

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
screenNameLen	Length of screenName
objectName	Object name that will be the target of add widget
objectNameLen	Length of objectName
alignment	Position for widget • 0 = Display object at the horizon center of specified y, while x ignored • 1 = Display object at specified x and y • 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

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

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

12.6.4.145 int lcd_addImage (IN char * *screenName*, IN int *screenNameLen*, IN char * *objectName*, IN int *objectNameLen*, IN BYTE *alignment*, IN int *xCord*, IN int *yCord*, IN char * *filename*, IN int *filenameLen*, OUT IDTLCDItem * *returnItem*)

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
screenNameLen	Length of screenName
objectName	Object name that will be the target of add image
objectNameLen	Length of objectName
alignment	Position for Image • 0 = Display object at the horizon center of specified y, while x ignored • 1 = Display object at specified x and y • 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.
filenameLen	Length of filename
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

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

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

12.6.4.146 int lcd_addLED (IN char * *screenName*, IN int *screenNameLen*, IN char * *objectName*, IN int *objectNameLen*, IN BYTE *alignment*, IN int *xCord*, IN int *yCord*, OUT IDTLCDItem * *returnItem*, IN BYTE * *LED*, IN int *LEDLen*)

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
screenNameLen	Length of screenName
objectName	Object name that will be the target of add LED
objectNameLen	Length of objectName
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
LEDLen	Length of LED

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

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

12.6.4.147 int lcd_addText (IN char * *screenName,* IN int *screenNameLen,* IN char * *objectName,* IN int *objectNameLen,* IN BYTE *alignment,* IN int *xCord,* IN int *yCord,* IN int *width,* IN int *height,* IN BYTE *fontID,* IN BYTE * *color,* IN int *colorLen,* IN char * *label,* IN int *labelLen,* OUT IDTLCDItem * *returnItem*)

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
screenNameLen	Length of screenName
objectName	Object name that will be the target of add text
objectNameLen	Length of objectName
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
colorLen	Length of color
label	Label to show on the text
labelLen	Length of label
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

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

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
14	RoundCondensed-Medium_12	RoundCondensed-Medium_0.ttf	12
15	RoundCondensed-Medium_18	RoundCondensed-Medium_0.ttf	18
16	RoundCondensed-Medium_24	RoundCondensed-Medium_0.ttf	24
17	RoundCondensed-Medium_36	RoundCondensed-Medium_0.ttf	36
18	RoundCondensed-Medium_48	RoundCondensed-Medium_0.ttf	48
19	RoundCondensed-Medium_60	RoundCondensed-Medium_0.ttf	60
20	RoundCondensed-Medium_72	RoundCondensed-Medium_0.ttf	72
21	RoundCondensed-Semibold_12	RoundCondensed-Semibold.ttf	12
22	RoundCondensed-Semibold_18	RoundCondensed-Semibold.ttf	18
23	RoundCondensed-Semibold_24	RoundCondensed-Semibold.ttf	24
24	RoundCondensed-Semibold_36	RoundCondensed-Semibold.ttf	36
25	RoundCondensed-Semibold_48	RoundCondensed-Semibold.ttf	48
26	RoundCondensed-Semibold_60	RoundCondensed-Semibold.ttf	60
27	RoundCondensed-Semibold_72	RoundCondensed-Semibold.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

12.6.4.148 int lcd_addVideo (IN char * *screenName*, IN int *screenNameLen*, IN char * *objectName*, IN int *objectNameLen*, IN BYTE *alignment*, IN int *xCord*, IN int *yCord*, IN char * *filename*, IN int *filenameLen*, OUT IDTLCDItem * *returnItem*)

Add Video

Adds a video 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 video
screenNameLen	Length of screenName
objectName	Object name that will be the target of add video
objectNameLen	Length of objectName
alignment	Position for Video • 0 = Display object at the horizon center of specified y, while x ignored • 1 = Display object at specified x and y • 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 Video, range 0-271
yCord	y-coordinate for Video, range 0-479
filename	Filename of the video. Must be available in the sd card.
filenameLen	Length of filename
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 video

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

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
video	1

12.6.4.149 int lcd_clearScreenInfo ( )

Clear Screen Info

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

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.6.4.150 int lcd_cloneScreen (IN char * *screenName*, IN int *screenNameLen*, IN char * *cloneName*, IN int *cloneNameLen*, OUT int * *cloneID*)

Clone Screen

Clones an existing screen.

Parameters

screenName	Screen name to clone.
screenNameLen	Length of screenName.
cloneName	Name of the cloned screen.
cloneNameLen	Length of cloneName.
cloneID	Screen ID of the cloned screen

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString

12.6.4.151 int lcd_createScreen (IN char * *screenName*, IN int *screenNameLen*, OUT int * *ScreenID*)

Create Screen

Creates a new screen.

Parameters

screenName	Screen name to use.
screenNameLen	Length of screenName.
screenID	Screen ID that was created.

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.6.4.152 int lcd_destroyScreen (IN char * *screenName*, IN int *screenNameLen*)

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.
screenNameLen	Length of screenName.

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.6.4.153 int lcd_getActiveScreen (OUT char * *screenName*, IN_OUT int * *screenNameLen*)

Get Active Screen

Returns the active screen ID.

Parameters

screenName	Screen name this is active.
screenNameLen	Length of screenName.

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.6.4.154 int lcd_getAllObjects (IN char * *screenName*, IN int *screenNameLen*, IN_OUT int * *objectNumbers*, OUT IDTObjectInfo * *objectInfo*)

Get All Objects on Screen

Get all created objects' name on certain screen

Parameters

screenName	Screen name to get all objects
screenNameLen	Length of screenName
objectNumbers	Number of created objects
returnObjects	Array of all created objects each element includes -objectID of a created object -objectName of a created object -objectNameLen of objectName

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.6.4.155 int lcd_getAllScreens (IN_OUT int * *screenNumbers*, OUT IDTScreenInfo * *screenInfo*)

Get All Screens

Get all created screens' name

Parameters

screenNumbers	Number of created screens
screenInfo	Array of all created screens each element includes -screenID of a created screen -screenName of a created screen -screenNameLen of screenName

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.6.4.156 int lcd_getButtonEvent (OUT int * *screenID*, OUT int * *objectID*, OUT char * *screenName*, IN_OUT int * *screenNameLen*, OUT char * *objectName*, IN_OUT int * *objectNameLen*, OUT int * *isLongPress*)

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
screenNameLen	Length of screenName
objectName	Button Name of the last clicked button
objectNameLen	Length of objectName
isLongPress	1 = Long Press, 0 = Short Press
ip	Optional: IP address to execute command on (for IP connected devices)

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.6.4.157 int lcd_loadScreenInfo ( )

Load Screen Info

Load all current screen information from flash to RAM

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.6.4.158 int lcd_queryObjectbyID (IN int *objectID*, OUT int * *objectNumbers*, OUT IDTScreenInfo * *screenInfo*)

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
objectNumbers	Number of the checked object
screenInfo	screen names containing the item

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.6.4.159 int lcd_queryObjectbyName (IN char * *objectName*, IN int *objectNameLen*, IN_OUT int * *objectNumbers*, OUT IDTScreenInfo * *screenInfo*)

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
objectNameLen	Length of objectName
objectNumbers	Number of the checked object

screenInfo	Array of all the screen names that contain the object

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.6.4.160 int lcd_queryScreenbyID (IN int *screenID*, OUT int * *result*, OUT int * *screenName*, IN_OUT int * *screenNameLen*)

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
screenNameLen	Length of screenName
ip	Optional: IP address to execute command on (for IP connected devices)

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.6.4.161 int lcd_queryScreenbyName (IN char * *screenName*, IN int *screenNameLen*, OUT int * *result*)

Queery Screen by Name

Check if the given screen exists or not

Parameters

screenName	Name of the checked screen
screenNameLen	Length of screenName
result	the checking result

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.6.4.162 void lcd_registerCallBk (pLCD_callBack *pLCDf*)

To register the lcd callback function to get the LCDItem. (Pass NULL to disable the callback.)

12.6.4.163 int lcd_removeItem (IN char * *screenName*, IN int *screenNameLen*, IN char * *objectName*, IN int *objectNameLen*)

Removed Item

Removes a component.

Parameters

screenName	Screen name where to remove the target from.
screenNameLen	Length of screenName.
objectName	Identifier of the component
objectNameLen	Length of objectName.

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.6.4.164 int lcd_setBacklight (IN BYTE *backlightVal*)

Set Backlight

Set backlight percentage. If the percent $>$ 100, it will be rejected. If 0 $<$ percent $<$ 10, backlight percent will be set to 10. If percent == 0, backlight will be turned off

Parameters

backlightVal	Backlight percent value to be sat

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.6.4.165 int lcd_showScreen (IN char * *screenName,* IN int *screenNameLen*)

Show Screen

Displays and makes active a previously created screen.

Parameters

screenName	Screen to display. The screen name is defined with lcd_createScreen.
screenNameLen	Length of screenName.

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.6.4.166 int lcd_storeScreenInfo ( )

Store Screen Info

Store all current screen information from RAM to flash

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.6.4.167 int lcd_updateColor (IN char * *screenName,* IN int *screenNameLen,* IN char * *objectName,* IN int *objectNameLen,* IN BYTE * *color,* IN int *colorLen*)

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
screenNameLen	Length of screenName.
objecName	Identifier of the component
objectNameLen	Length of objectName.
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
colorLen	Length of color.

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.6.4.168 int lcd_updateLabel (IN char * *screenName*, IN int *screenNameLen*, IN char * *objectName*, IN int *objectNameLen*, IN char * *label*, IN int *labelLen*)

Update Label

Updates the component label.

Parameters

screenName	Screen name that will be the target of update label
screenNameLen	Length of screenName.
objectName	Identifier of the component
objectNameLen	Length of objectName.
label	Label to show on the component
labelLen	Length of label.

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.6.4.169 int lcd_updatePosition (IN char * *screenName*, IN int *screenNameLen*, IN char * *objectName*, IN int *objectNameLen*, IN BYTE *alignment*, IN int *new_xCord*, IN int *new_yCord*)

Update Position

Updates the component position.

Parameters

screenName	Screen Name that will be the target of update position
screenNameLen	Length of screenName.
objectName	Identifier of the component
objectNameLen	Length of objectName.
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

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.6.4.170 int loyalty_cancelTransaction ( )

Cancel Loyalty Transaction Only for VP6800

Cancels the currently executing transaction.

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.6.4.171 int loyalty_cancelTransactionSilent (int *enable*)

Cancel Loyalty Transaction Silent Only for VP6800

Cancel transaction with or without showing the LCD message

Parameters

enable	0: With LCD message 1: Without LCD message

Returns

success or error code. Values can be parsed with device_getIDGStatusCodeString

12.6.4.172 void loyalty_registerCallBk (pEMV_callBack *pEMVf*)

To register the loyalty callback function to get the EMV processing response. (Pass NULL to disable the callback.) Only for VP6800

12.6.4.173 int loyalty_startTransaction (IN double *amount*, IN double *amtOther*, IN int *type*, IN const int *_timeout*, IN BYTE * *tags*, IN int *tagsLen*, IN const int *cardType*, IN const int *iccReadType*)

Start Loyalty Transaction Request Only for VP6800

Authorizes the transaction for an MSR/ICC card

The tags will be returned in the callback routine.

Parameters

amount	Transaction amount value (tag value 9F02) • SEE IMPORTANT NOTE BELOW
amtOther	Other amount value, if any (tag value 9F03) • SEE IMPORTANT NOTE BELOW
type	Transaction type (tag value 9C).
timeout	Timeout value in seconds.
tags	Any other tags to be included in the request. Passed as TLV. 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.
tagsLen	The length of tags data buffer.

>>>>>IMPORTANT: parameters for amount and amtOther MUST BE PASSED AS A DOUBLE VALUE WITH DECIMAL POINT. Example, do not pass 1, but instead pass 1.0 or 1.00. Otherwise, results will be unpredictable

Parameters

cardType	Set available card to accept. 0 = MSR only. 1 = MSR and ICC.
iccReadType	In case of ICC reading, this is how to behave. 0 = Same as device_startTransaction 1 = When reading ICC, read DF4F(JIS2EquivalentData) in ReadRecord.

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString() Note: if auto poll is on, it will returm the error IDG_P2_STATUS_CODE_COMMAND_NOT_ALLOWED

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 device_startTransaction. Tag FFEE06 must contain tag 9F26 and 9F22, and can optionanally contain tags 9F2B and DF01. 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

12.6.4.174 int msr_cancelMSRSwipe ( )

Disable MSR Swipe Cancels MSR swipe request.

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.6.4.175 void msr_registerCallBk (pMSR_callBack *pMSRf*)

To register the msr callback function to get the MSR card data. (Pass NULL to disable the callback.)

12.6.4.176 void msr_registerCallBkp (pMSR_callBackp *pMSRf*)

To register the msr callback function to get the MSR card data pointer. (Pass NULL to disable the callback.)

12.6.4.177 int msr_startMSRSwipe (IN int *_timeout*)

Start MSR Swipe

Enables MSR, waiting for swipe to occur. Allows track selection. Returns IDTMSRData instance to swipeMSRData()

Parameters

timeout	Swipe Timeout Value

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString() Note: if auto poll mode is on, it will return command not allowed error

12.6.4.178 void parseMSRData (IN BYTE * *resData,* IN int *resLen,* IN_OUT IDTMSRData * *cardData*)

Parser the MSR data from the buffer into IDTMSTData structure

Parameters

resData	MSR card data buffer
resLen	the length of resData
cardData	the parser result with IDTMSTData structure

12.6.4.179 int pin_cancelPINEntry ( )

Cancel PIN Entry

Cancels PIN entry request

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.6.4.180 int pin_capturePin (IN int *timeout*, IN int *type*, IN char * *PAN*, IN int *PANLen*, IN int *minPIN*, IN int *maxPIN*, IN char * *message*, IN int *messageLen*)

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)
PANLen	Length of PAN
minPIN	Minimum PIN Length
maxPIN	Maximum PIN Length
message	LCD Message
messageLen	Length of message

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.6.4.181 int pin_capturePinExt (IN int *type*, IN char * *PAN*, IN int *PANLen*, IN int *minPIN*, IN int *maxPIN*, IN char * *message*, IN int *messageLen*, IN char * *verify*, IN int *verifyLen*)

- Capture PIN Ext

Parameters

type	PAN and Key Type • 00h: MKSK to encrypt PIN, Internal PAN (from MSR or Manual PAN Entry or Contactless EMV Transaction) • 01h: DUKPT to encrypt PIN, Internal PAN (from MSR or Manual PAN Entry or Contactless EMV Transaction) • 10h: MKSK to encrypt PIN, External Plaintext PAN • 11h: DUKPT to encrypt PIN, External Plaintext PAN • 20h: MKSK to encrypt PIN, External Ciphertext PAN (for PIN pad only) • 21h: DUKPT to encrypt PIN, External Ciphertext PAN (for PIN pad only) • 80h: MKSK to encrypt PIN, Internal PAN, Verify PIN (from MSR or Manual PAN Entry or Contactless EMV Transaction) • 81h: DUKPT to encrypt PIN, Internal PAN, Verify PIN (from MSR or Manual PAN Entry or Contactless EMV Transaction) • 90h: MKSK to encrypt PIN, External Plaintext PAN, Verify PIN • 91h: DUKPT to encrypt PIN, External Plaintext PAN, Verify PIN
PAN	Personal Account Number (if internal, value is 0)
PANLen	Length of PAN
minPIN	Minimum PIN Length
maxPIN	Maximum PIN Length
message	LCD Message Up to 2 lines of text, each line 1-16, separated by 0x00
messageLen	Length of 1st scenario LCD message, valid in 00h∼21h (0∼33).If no LCD message input, length is 00h, and display default msg “PLEASE ENTER PIN”
verify	LCD Message Up to 2 lines of text, each line 1-16, separated by 0x00
verifyLen	Length of 2nd Scenario LCD message.valid in 00h∼21h (0∼33).This field is present only when PAN and Key Type has Verify PIN.If no LCD message input, length is 00h, and display default msg “ ENTER PIN AGAIN”

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.6.4.182 int pin_getPanEntry (IN int *csc,* IN int *expDate,* IN int *ADR,* IN int *ZIP,* IN int *mod10CK,* IN int *timeout,* IN int *encPANOnly*)

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
messageLen	Length of 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
signatureLen	Length of signature

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString() 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

csc	Request CSS
expDate	Request Expiration Date
ADR	Request Address
ZIP	Request Zip
mod10CK	Validate entered PAN passes MOD-10 checking before accepting
timeout	Number of seconds that the reader waits for the data entry session to complete, stored as a big-endian number. 0 = no timeout
encPANOnly	Output only encrypted PAN

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.6.4.183 int pin_inputFromPrompt (BYTE *mask*, BYTE *preClearText*, BYTE *postClearText*, int *minLen*, int *maxLen*, char * *lang*, BYTE *promptID*, char * *defaultResponse*, int *defaultResponseLen*, int *timeout*)

Get PIN Input from Prompt Results returned to PIN Callback. If successful function key capture, data is returned as IDTPINData.keyString.

Parameters

mask	0 = no masking, 1 = Display "*" for numeric key according to Pre-Cleartext and Post-Cleartext display
preClearText	Range 0-6 Characters to mask at start of text if masking is on;
postClearText	Range 0-6 Characters to mask at end of text if masking is on;
minLen	Minimum number of digits required to input
maxLen	Maximum number of digits allowed to be input
lang	Valid values; “EN” is English display message “JP” is Japanese display message “ES” is Spanish display message “FR” is French display message “ZH” is Chinese display message “PT” is Portuguese display message
promptID	Valid values: 0x00: Enter Phone Number 0x01: Enter IP Address 0x02: Enter Subnet Mask 0x03: Enter Default Gateway 0x04: Enter Odometer Reading/Mileage 0x05: Enter Employee ID 0x06: Enter Password for Default Factory Setting 0x07: Enter Email Address (Full keyboard)

defaultResponse	Default String on input field
default-ResponseLen	Length of defaultResponse
timeout	Timeout, in seconds

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.6.4.184 int pin_promptForNumericKey (IN int *timeout,* IN int *maskInput,* IN int *minLen,* IN int *maxLen,* IN char * *message,* IN int *messageLen,* BYTE * *signature,* IN int *signatureLen*)

Capture Numeric Input

Parameters

timeout	Timeout, in seconds. Value of 0 will use system timeout, if any
maskInput	• 0 = Display numeric for numeric key on LCD • 1 = Display * for numeric key on LCD
minLen	Minimum input Length
maxLen	Maximum input Length
message	Plaintext Display Message. - 16 chars max
messageLen	Length of 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
signatureLen	Length of signature

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.6.4.185 int pin_promptForNumericKeyWithSwipe (IN int *timeout,* IN BYTE *function,* IN int *minLen,* IN int *maxLen,* IN char * *line1,* IN int *line1Len,* IN char * *line2,* IN int *line2Len,* BYTE * *signature,* IN int *signatureLen*)

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
line1	Line 1 of LCD Message - 16 chars max
line1Len	Length of line1
line2	Line 2 of LCD Message - 16 chars max
line2Len	Length of line2
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
signatureLen	Length of signature

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.6.4.186 void pin_registerCallBk (pPIN_callBack *pPINf*)

To register the pin callback function to get the PINPad data. (Pass NULL to disable the callback.)

12.6.4.187 int pin_setKeyValues (int *mode*)

Set Key Values

Set return key values on or off

Parameters

mode	On: 1, Off: 0

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.6.4.188 void registerHotplugCallBk (pMessageHotplug *pMsgHotplug*)

To register the USB HID hot-plug callback function which implemented in the application to monitor the hotplug message from the SDK.

12.6.4.189 void registerLogCallBk (pSendDataLog *pFSend*, pReadDataLog *pFRead*)

To register the log callback function which implemented in the application to monitor sending/reading data between application and reader.

12.6.4.190 int rs232_device_init (int *deviceType*, int *port_number*, int *brate*)

Initial the device by RS232

It will try to connect to the device with provided deviceType, port_number, and brate.

Parameters

deviceType	Device to connect to
port_number	Port number of the device

Port nr. | **Linux** | **Windows**

| 0 | ttyS0 | COM1 | | 1 | ttyS1 | COM2 | | 2 | ttyS2 | COM3 | | 3 | ttyS3 | COM4 | | 4 | ttyS4 | COM5 | | 5 | ttyS5 | COM6 | | 6 | ttyS6 | COM7 | | 7 | ttyS7 | COM8 | | 8 | ttyS8 | COM9 | | 9 | ttyS9 | COM10 | | 10 | ttyS10 | COM11 | | 11 | ttyS11 | COM12 | | 12 | ttyS12 | COM13 | | 13 | ttyS13 | COM14 | | 14 | ttyS14 | COM15 | | 15 | ttyS15 | COM16 | | 16 | ttyUSB0 | n.a. | | 17 | ttyUSB1 | n.a. | | 18 | ttyUSB2 | n.a. | | 19 | ttyUSB3 | n.a. | | 20 | ttyUSB4 | n.a. | | 21 | ttyUSB5 | n.a. | | 22 | ttyAMA0 | n.a. | | 23 | ttyAMA1 | n.a. | | 24 | ttyACM0 | n.a. | | 25 | ttyACM1 | n.a. | | 26 | rfcomm0 | n.a. | | 27 | rfcomm1 | n.a. | | 28 | ircomm0 | n.a. | | 29 | ircomm1 | n.a. | | 30 | cuau0 | n.a. | | 31 | cuau1 | n.a. | | 32 | cuau2 | n.a. | | 33 | cuau3 | n.a. | | 34 | cuaU0 | n.a. | | 35 | cuaU1 | n.a. | | 36 | cuaU2 | n.a. | | 37 | cuaU3 | n.a. | | 38 | ttyIDG | n.a. |

Parameters

brate	Bitrate of the device

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.6.4.191 char* SDK_Version ( )

To Get SDK version

Returns

return the SDK version string

12.6.4.192 int setAbsoluteLibraryPath (const char * *absoluteLibraryPath*)

Set the path to use when searching for ID TECH's libraries. If this is not set, the libraries will be searched for with the system's default procedures.

Parameters

absoluteLibrary-Path	The absolute path to ID TECH's libraries.

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.7 Source_C/libIDT_PipReader.h File Reference

PipReader API.

```
#include "IDTDef.h"
```

Macros

- #define IN
- #define OUT
- #define IN_OUT

Typedefs

- typedef void(∗ pMessageHotplug)(int, int)
- typedef void(∗ pSendDataLog)(unsigned char ∗, int)
- typedef void(∗ pReadDataLog)(unsigned char ∗, int)
- typedef void(∗ pEMV_callBack)(int, int, unsigned char ∗, int, IDTTransactionData ∗, EMV_Callback ∗, int)
- typedef void(∗ pMSR_callBack)(int, IDTMSRData)
- typedef void(∗ pMSR_callBackp)(int, IDTMSRData ∗)
- typedef void(∗ pPIN_callBack)(int, IDTPINData ∗)
- typedef void(∗ pCMR_callBack)(int, IDTCMRData ∗)
- typedef void(∗ pCSFS_callBack)(BYTE status)
- typedef void(∗ ftpComm_callBack)(int, int, int)
- typedef void(∗ httpComm_callBack)(BYTE ∗, int)
- typedef void(∗ v4Comm_callBack)(BYTE, BYTE, BYTE ∗, int)

Functions

- void registerHotplugCallBk (pMessageHotplug pMsgHotplug)
- void registerLogCallBk (pSendDataLog pFSend, pReadDataLog pFRead)
- void emv_registerCallBk (pEMV_callBack pEMVf)
- void ctls_registerCallBk (pMSR_callBack pCTLSf)
- void ctls_registerCallBkp (pMSR_callBackp pCTLSf)
- void pin_registerCallBk (pPIN_callBack pPINf)
- void device_registerCameraCallBk (pCMR_callBack pCMRf)
- void device_registerCardStatusFrontSwitchCallBk (pCSFS_callBack pCSFSf)
- char ∗ SDK_Version ()
- int setAbsoluteLibraryPath (const char ∗absoluteLibraryPath)
- int device_init ()
- int rs232_device_init (int deviceType, int port_number, int brate)
- int device_setCurrentDevice (int deviceType)
- int device_close ()
- void device_getIDGStatusCodeString (IN int returnCode, OUT char ∗despcrition)
- int device_isConnected ()
- int device_isAttached (int deviceType)
- int device_getFirmwareVersion (OUT char ∗firmwareVersion)
- int device_getFirmwareVersion_Len (OUT char ∗firmwareVersion, IN_OUT int ∗firmwareVersionLen)
- int device_pingDevice ()
- int device_controlUserInterface (IN BYTE ∗values)
- int device_getCurrentDeviceType ()
- int device_SendDataCommandNEO (IN int cmd, IN int subCmd, IN BYTE ∗data, IN int dataLen, OUT BYTE ∗response, IN_OUT int ∗respLen)

- int device_enablePassThrough (int enablePassThrough)
- int device_setBurstMode (IN BYTE mode)
- int device_setPollMode (IN BYTE mode)
- int device_setMerchantRecord (int index, int enabled, char ∗merchantID, char ∗merchantURL)
- int device_getTransactionResults (IDTMSRData ∗cardData)
- int device_getMerchantRecord (IN int index, OUT BYTE ∗record)
- int device_getMerchantRecord_Len (IN int index, OUT BYTE ∗record, IN_OUT int ∗recordLen)
- int device_getSDKWaitTime ()
- void device_setSDKWaitTime (int waitTime)
- int config_getSerialNumber (OUT char ∗sNumber)
- int config_getSerialNumber_Len (OUT char ∗sNumber, IN_OUT int ∗sNumberLen)
- int ctls_startTransaction (IN double amount, IN double amtOther, IN int type, IN const int _timeout, IN BYTE ∗tags, IN int tagsLen)
- int ctls_activateTransaction (IN const int _timeout, IN BYTE ∗tags, IN int tagsLen)
- int ctls_cancelTransaction ()
- int ctls_retrieveApplicationData (IN BYTE ∗AID, IN int AIDLen, OUT BYTE ∗tlv, IN_OUT int ∗tlvLen)
- int ctls_setApplicationData (IN BYTE ∗tlv, IN int tlvLen)
- int ctls_removeApplicationData (IN BYTE ∗AID, IN int AIDLen)
- int ctls_removeAllApplicationData ()
- int ctls_retrieveAIDList (OUT BYTE ∗AIDList, IN_OUT int ∗AIDListLen)
- int ctls_retrieveTerminalData (OUT BYTE ∗tlv, IN_OUT int ∗tlvLen)
- int ctls_setTerminalData (IN BYTE ∗tlv, IN int tlvLen)
- int ctls_retrieveCAPK (IN BYTE ∗capk, IN int capkLen, OUT BYTE ∗key, IN_OUT int ∗keyLen)
- int ctls_setCAPK (IN BYTE ∗capk, IN int capkLen)
- int ctls_removeCAPK (IN BYTE ∗capk, IN int capkLen)
- int ctls_removeAllCAPK ()
- int ctls_retrieveCAPKList (OUT BYTE ∗keys, IN_OUT int ∗keysLen)
- int ctls_setConfigurationGroup (IN BYTE ∗tlv, IN int tlvLen)
- int ctls_getConfigurationGroup (IN int group, OUT BYTE ∗tlv, OUT int ∗tlvLen)
- int ctls_getAllConfigurationGroups (OUT BYTE ∗tlv, IN_OUT int ∗tlvLen)
- int ctls_removeConfigurationGroup (int group)
- void parseMSRData (IN BYTE ∗resData, IN int resLen, IN_OUT IDTMSRData ∗cardData)

12.7.1 Detailed Description

PipReader API. PipReader Global API methods.

12.7.2 Macro Definition Documentation

12.7.2.1 #define IN

INPUT parameter.

12.7.2.2 #define IN_OUT

INPUT / OUTPUT PARAMETER.

12.7.2.3 #define OUT

OUTPUT parameter.

12.7.3 Typedef Documentation

12.7.3.1 typedef void(∗ ftpComm_callBack)(int, int, int)

Define the comm callback function to get FTP file transfer status

It should be passed as a parameter in a FTP request, Signature (int, int, int) = response code, current block, total blocks RESPONSE CODES: 100 = FILE DOWNLOAD STARTED 101 = FILE BLOCK XX OF XX RECEIVED 102 = FILE DOWNLOAD COMPLETED 103 = FILE DOWNLOAD TERMINATED PREMATURELY

12.7.3.2 typedef void(∗ httpComm_callBack)(BYTE ∗, int)

Define the comm callback function to get the async url data

It should be registered using the comm_registerHTTPCallback

12.7.3.3 typedef void(∗ pCMR_callBack)(int, IDTCMRData ∗)

Define the camera callback function to get the image data

It should be registered using the device_registerCameraCallBk,

12.7.3.4 typedef void(∗ pCSFS_callBack)(BYTE status)

Define the card status and front switch callback function to get card and front switch status

It should be registered using the device_registerCardStatusFrontSwitchCallBk,

12.7.3.5 typedef void(∗ pEMV_callBack)(int, int, unsigned char ∗, int, IDTTransactionData ∗, EMV_Callback ∗, int)

Define the EMV callback function to get the transaction message/data/result.

It should be registered using the emv_registerCallBk,

12.7.3.6 typedef void(∗ pMessageHotplug)(int, int)

Define the USB hot-plug callback function to monitor the info when plug in/out the reader.

It should be registered using the registerHotplugCallBk, The first integer parameter is device type, and the second integer parameter is either 0: Device Plugged Out or 1: Device Plugged In

12.7.3.7 typedef void(∗ pMSR_callBack)(int, IDTMSRData)

Define the MSR callback function to get the MSR card data

It should be registered using the msr_registerCallBk, this callback function is for backward compatibility

12.7.3.8 typedef void(∗ pMSR_callBackp)(int, IDTMSRData ∗)

Define the MSR callback function to get pointer to the MSR card data

It should be registered using the msr_registerCallBk, this callback function is recommended instead of pMSR_callBack

12.7.3.9 typedef void(∗ pPIN_callBack)(int, IDTPINData ∗)

Define the PINPad callback function to get the input PIN Pad data

It should be registered using the pin_registerCallBk,

12.7.3.10 typedef void(∗ pReadDataLog)(unsigned char ∗, int)

Define the read response callback function to monitor the reading response from the reader.

It should be registered using the registerLogCallBk,

12.7.3.11 typedef void(∗ pSendDataLog)(unsigned char ∗, int)

Define the send command callback function to monitor the sending command into the reader.

It should be registered using the registerLogCallBk,

12.7.3.12 typedef void(∗ v4Comm_callBack)(BYTE, BYTE, BYTE ∗, int)

Define the comm callback function to receive the V4 Protocol packets received by the device from an external source (IP/USB/RS-232) It should be registered using the comm_registerV4Callback, Data callback will contain command, sub-command, and data from V4 packet

12.7.4 Function Documentation

12.7.4.1 int config_getSerialNumber (OUT char ∗ *sNumber*)

DEPRECATED : please use config_getSerialNumber_Len(OUT char∗ sNumber, IN_OUT int ∗sNumberLen)

Polls device for Serial Number

Parameters

sNumber	Returns Serial Number; needs to have at least 64 bytes of memory

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString

12.7.4.2 int config_getSerialNumber_Len (OUT char ∗ *sNumber,* IN_OUT int ∗ *sNumberLen*)

Polls device for Serial Number

Parameters

sNumber	Returns Serial Number
sNumberLen	Length of Serial Number

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString

12.7.4.3 int ctls_activateTransaction (IN const int *_timeout,* IN BYTE * *tags,* IN int *tagsLen*)

Start CTLS Transaction Request

Authorizes the CTLS 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 stream. Example, tag 9F0C with amount 0x000000000100 would be 0x9F0C06000000000100
tagsLen	The length of tags data buffer.

>>>>>IMPORTANT: parameters for amount and amtOther MUST BE PASSED AS A DOUBLE VALUE WITH DECIMAL POINT. Example, do not pass 1, but instead pass 1.0 or 1.00. Otherwise, results will be unpredictable

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString() Note: if auto poll is on, it will returm the error IDG_P2_STATUS_CODE_COMMAND_NOT_ALLOWED

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

12.7.4.4 int ctls_cancelTransaction ( )

Cancel EMV Transaction

Cancels the currently executing EMV transaction.

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.7.4.5 int ctls_getAllConfigurationGroups (OUT BYTE * *tlv*, IN_OUT int * *tlvLen*)

Retrieve All Configuration Groups

Returns all the Configuration Groups installed on the terminal for CTLS

Parameters

tlv	The TLV elements data
tlvLen	the length of tlv data buffer.

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.7.4.6 int ctls_getConfigurationGroup (IN int *group*, OUT BYTE * *tlv*, OUT int * *tlvLen*)

Get Configuration Group

Retrieves the Configuration for the specified Group.

Parameters

group	Configuration Group
tlv	return data
tlvLen	the length of tlv data buffer

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.7.4.7 void ctls_registerCallBk (pMSR_callBack *pCTLSf*)

To register the ctls callback function to get the MSR card data. (Pass NULL to disable the callback.)

12.7.4.8 void ctls_registerCallBkp (pMSR_callBackp *pCTLSf*)

To register the ctls callback function to get the MSR card data pointer. (Pass NULL to disable the callback.)

12.7.4.9 int ctls_removeAllApplicationData ( )

Remove All Application Data

Removes all the Application Data

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.7.4.10 int ctls_removeAllCAPK ( )

Remove All Certificate Authority Public Key

Removes all the CAPK

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.7.4.11 int ctls_removeApplicationData (IN BYTE * *AID,* IN int *AIDLen*)

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
AIDLen	the length of AID data buffer

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.7.4.12 int ctls_removeCAPK (IN BYTE * *capk,* IN int *capkLen*)

Remove Certificate Authority Public Key

Removes the CAPK as specified by the RID/Index

Parameters

capk	6 byte CAPK = 5 bytes RID + 1 byte INDEX
capkLen	the length of capk data buffer

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.7.4.13 int 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	Values can be parsed with device_getIDGStatusCodeString()

12.7.4.14 int ctls_retrieveAIDList (OUT BYTE * *AIDList,* IN_OUT int * *AIDListLen*)

Retrieve AID list

Returns all the AID names installed on the terminal for CTLS. .

Parameters

AIDList	array of AID name byte arrays
AIDListLen	the length of AIDList array buffer

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.7.4.15 int ctls_retrieveApplicationData (IN BYTE * *AID*, IN int *AIDLen*, OUT BYTE * *tlv*, IN_OUT int * *tlvLen*)

Retrieve Application Data by AID

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

Parameters

AID	Name of ApplicationID. Must be between 5 and 16 bytes
AIDLen	the length of AID data buffer.
tlv	The TLV elements of the requested AID
tlvLen	the length of tlv data buffer.

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.7.4.16 int ctls_retrieveCAPK (IN BYTE * *capk*, IN int *capkLen*, OUT BYTE * *key*, IN_OUT int * *keyLen*)

Retrieve Certificate Authority Public Key

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

Parameters

capk	6 bytes CAPK = 5 bytes RID + 1 byte Index
capkLen	the length of capk data buffer
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.
keyLen	the length of key data buffer •

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.7.4.17 int ctls_retrieveCAPKList (OUT BYTE * *keys*, IN_OUT int * *keysLen*)

Retrieve the Certificate Authority Public Key list

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

Parameters

keys	[key1][key2]...[keyn], each key 6 bytes where key = 5 bytes RID + 1 byte index
keysLen	the length of keys data buffer

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.7.4.18 int ctls_retrieveTerminalData (OUT BYTE * *tlv*, IN_OUT int * *tlvLen*)

Retrieve Terminal Data

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

Parameters

tlv	Response returned as a TLV
tlvLen	the length of tlv data buffer

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.7.4.19 int ctls_setApplicationData (IN BYTE * *tlv*, IN int *tlvLen*)

Set Application Data by AID

Sets the Application Data for CTLS as specified by TLV data

Parameters

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

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

Parameters

tlvLen	the length of tlv data buffer

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.7.4.20 int ctls_setCAPK (IN BYTE * *capk*, IN int *capkLen*)

Set Certificate Authority Public Key

Sets the CAPK as specified by the CAKey structure

Parameters

capk	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.
capkLen	the length of capk data buffer

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.7.4.21 int ctls_setConfigurationGroup (IN BYTE * *tlv*, IN int *tlvLen*)

Set Configuration Group

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

Parameters

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

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.7.4.22 int ctls_setTerminalData (IN BYTE * *tlv*, IN int *tlvLen*)

Set Terminal Data

Sets the Terminal Data for CTLS as specified by the TLV. The first TLV must be Configuration Group Number (Tag FFE4). The terminal global data is group 0, so the first TLV would be FFE40100. Other groups can be defined using this method (1 or greater), and those can be retrieved with ctls_getConfigurationGroup(int group), and deleted with ctls_removeConfigurationGroup(int group). You cannot delete group 0.

Parameters

tlv	TerminalData configuration file
tlvLen	the length of tlv data buffer

Return values

RETURN_CODE	Values can be parsed with device_getIDGStatusCodeString()

12.7.4.23 int ctls_startTransaction (IN double *amount,* IN double *amtOther,* IN int *type,* IN const int *_timeout,* IN BYTE * *tags,* IN int *tagsLen*)

Start CTLS Transaction Request

Authorizes the CTLS transaction for an ICC card

The tags will be returned in the callback routine.

Parameters

amount	Transaction amount value (tag value 9F02) - SEE IMPORTANT NOTE BELOW
amtOther	Other amount value, if any (tag value 9F03) - SEE IMPORTANT NOTE BELOW
type	Transaction type (tag value 9C).
timeout	Timeout value in seconds.
tags	Any other tags to be included in the request. Passed as TLV stream. Example, tag 9F0C with amount 0x000000000100 would be 0x9F0C06000000000100 If tags 9F02 (amount),9-F03 (other amount), or 9C (transaction type) are included, they will take priority over these values supplied as individual parameters to this method.
tagsLen	The length of tags data buffer.

>>>>>IMPORTANT: parameters for amount and amtOther MUST BE PASSED AS A DOUBLE VALUE WITH DECIMAL POINT. Example, do not pass 1, but instead pass 1.0 or 1.00. Otherwise, results will be unpredictable

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString() Note: if auto poll is on, it will returm the error IDG_P2_STATUS_CODE_COMMAND_NOT_ALLOWED

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

12.7.4.24 int device_close ( )

Close the device

Returns

RETURN_CODE: 0: success, 0x0A: failed

12.7.4.25 int device_controlUserInterface (IN BYTE * *values*)

Control User Interface

Controls the User Interface: Display, Beep, LED

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

```
- 00h: LED 0 (Power LED) 01h: LED 1
- 02h: LED 2
- 03h: LED 3
- FFh: All LEDs
Byte[3] = LED Status
- 00h: LED Off
- 01h: LED On
```

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.7.4.26 int device_enablePassThrough (int *enablePassThrough*)

Start Remote Key Injection

Starts a remote key injection request with IDTech RKI servers. This function is reserved and not implemented.

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

Enable Pass Through - NEO

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

Parameters

enablePass-Through	1 = pass through ON, 0 = pass through OFF

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.7.4.27 int device_getCurrentDeviceType ( )

Get current active device type

Returns

: return the device type defined as DEVICE_TYPE in the IDTDef.h

12.7.4.28 int device_getFirmwareVersion (OUT char * *firmwareVersion*)

DEPRECATED : please use device_getFirmwareVersion_Len(OUT char* firmwareVersion, IN_OUT int *firmware-VersionLen)

Polls device for Firmware Version

Parameters

firmwareVersion	Response returned of Firmware Version; needs to have at least 128 bytes of memory

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.7.4.29 int device_getFirmwareVersion_Len (OUT char * *firmwareVersion,* IN_OUT int * *firmwareVersionLen*)

Polls device for Firmware Version

Parameters

firmwareVersion	Response returned of Firmware Version
firmwareVersion-Len	Length of Firmware Version

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.7.4.30 void device_getIDGStatusCodeString (IN int *returnCode,* OUT char * *despcrition*)

Review the return code description.

Parameters

returnCode	the response result.
description	

Return values

the	string for description of response result

- 0: "no error, beginning task";
- 1: "no response from reader";
- 2: "invalid response data";
  - 01: " Incorrect Header Tag";
  - 02: " Unknown Command";
  - 03: " Unknown Sub-Command";
  - 04: " CRC Error in Frame";
  - 05: " Incorrect Parameter";
  - 06: " Parameter Not Supported";
  - 07: " Mal-formatted Data";
  - 08: " Timeout";
  - 0A: " Failed / NACK";
  - 0B: " Command not Allowed";
  - 0C: " Sub-Command not Allowed";
  - 0D: " Buffer Overflow (Data Length too large for reader buffer)";
  - 0E: " User Interface Event";
  - 10: " Need clear firmware(apply in boot loader only)";
  - 11: " Communication type not supported, VT-1, burst, etc. Need encrypted firmware (apply in boot loader only)";
  - 12: " Secure interface is not functional or is in an intermediate state.";
  - 13: " Data field is not mod 8";
  - 14: " Pad 0x80 not found where expected";
  - 15: " Specified key type is invalid";
  - 16: " Could not retrieve key from the SAM (InitSecureComm)";
  - 17: " Hash code problem";
  - 18: " Could not store the key into the SAM (InstallKey)";

- 19: " Frame is too large";
- 1A: " Unit powered up in authentication state but POS must resend the InitSecureComm command";
- 1B: " The EEPROM may not be initialized because SecCommInterface does not make sense";
- 1C: " Problem encoding APDU Module-Specific Status Codes ";
- 20: " Unsupported Index (ILM) SAM Transceiver error - problem communicating with the SAM (Key Mgr)";
- 21: " Unexpected Sequence Counter in multiple frames for single bitmap (ILM)Length error in data returned from the SAM (Key Mgr) - 22: " Improper bit map (ILM)"; - 23: " Request Online Authorization"; - 24: " ViVOCard3 raw data read successful"; - 25: " Message index not available (ILM) ViVOcomm activate transaction card type (ViVOcomm)"; - 26: " Version Information Mismatch (ILM)"; - 27: " Not sending commands in correct index message index (ILM)"; - 28: " Time out or next expected message not received (ILM)"; - 29: " ILM languages not available for viewing (ILM)"; - 2A: " Other language not supported (ILM)"; - 41: " from 41 to 4F, Module-specific errors for Key Manager";
- 50: " Auto-Switch OK";
- 51: " Auto-Switch failed";
- 70: " Antenna Error 80h Use another card";
- 81: " Insert or swipe card";
- 90: " Data encryption Key does not exist";
- 91: " Data encryption Key KSN exhausted";

12.7.4.31 int device_getMerchantRecord (IN int *index,* OUT BYTE * *record*)

DEPRECATED : please use device_getMerchantRecord_Len(IN int index, OUT BYTE * record, IN_OUT int *recordLen)

Get Merchant Record

Gets the merchant record for the device.

Parameters

index	Merchant Record index, valid values 1-6
record;	needs to have at least 99 bytes of memory response data from reader. Merchant Record Index: 1 byte enabled: 1 byte Merchant ID: 32 bytes Length of Merchant URL: 1 byte Merchant URL: 64 bytes

Returns

success or error code. Values can be parsed with device_getIDGStatusCodeString()

See Also

ErrorCode

12.7.4.32 int device_getMerchantRecord_Len (IN int *index,* OUT BYTE * *record,* IN_OUT int * *recordLen*)

Get Merchant Record

Gets the merchant record for the device.

Parameters

index	Merchant Record index, valid values 1-6
record	response data from reader. Merchant Record Index: 1 byte enabled: 1 byte Merchant ID: 32 bytes Length of Merchant URL: 1 byte Merchant URL: 64 bytes
recordLen	Length of record

Returns

success or error code. Values can be parsed with device_getIDGStatusCodeString()

See Also

ErrorCode

12.7.4.33 int device_getSDKWaitTime ( )

Get SDK Wait Time

Get the SDK wait time for transactions

Returns

SDK wait time in seconds

12.7.4.34 int device_getTransactionResults (IDTMSRData * *cardData*)

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

Parameters

cardData	The transaction results

Returns

success or error code. Values can be parsed with device_getResponseCodeString

See Also

ErrorCode

12.7.4.35 int device_init ( )

Initial the device by USB

It will detect the device and trying connect.

The connect status can be checked by device_isConnected().

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.7.4.36 int device_isAttached (int *deviceType*)

Check if the device is attached to the USB port The function device_init() must be called before this function.

Parameters

deviceType,the	device type of the USB device

Returns

1 if the device is attached, or 0 if the device is not attached

12.7.4.37 int device_isConnected ( )

Check the device conntected status

Returns

DEVICE_DISCONNECT=0, or DEVICE_CONNECTED = 1

12.7.4.38 int device_pingDevice ( )

Ping Device

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

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.7.4.39 void device_registerCameraCallBk (pCMR_callBack *pCMRf*)

To register the camera callback function to get the image data. (Pass NULL to disable the callback.)

12.7.4.40 void device_registerCardStatusFrontSwitchCallBk (pCSFS_callBack *pCSFSf*)

To register the card status and front switch callback function to get status. (Pass NULL to disable the callback.)

12.7.4.41 int device_SendDataCommandNEO (IN int *cmd,* IN int *subCmd,* IN BYTE * *data,* IN int *dataLen,* OUT BYTE * *response,* IN_OUT int * *respLen*)

Send a Command to NEO device

Sends a command to the NEO device .

Parameters

cmd	command to execute.
subCmd,sub	command to execute.
data	buffer of NEO command data.
dataLen,the	length of the buffer data.
response	Response data
respLen,the	length of Response data

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.7.4.42 int device_setBurstMode (IN BYTE *mode*)

Send Burst Mode - NEO

Sets the burst mode for the device.

Parameters

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

Returns

success or error code. Values can be parsed with device_getIDGStatusCodeString

See Also

ErrorCode

12.7.4.43 int device_setCurrentDevice (int *deviceType*)

Sets the current device to talk to

The connect status can be checked by device_isConnected().

Parameters

deviceType	Device to connect to
	
	`enum DEVICE_TYPE`
	`{`
	`IDT_DEVICE_UNKNOWN=0,`
	`IDT_DEVICE_AUGUSTA_HID,`
	`IDT_DEVICE_AUGUSTA_KB,`
	`IDT_DEVICE_AUGUSTA_S_HID,`
	`IDT_DEVICE_AUGUSTA_S_KB,`
	`IDT_DEVICE_AUGUSTA_S_TTK_HID,`
	`IDT_DEVICE_SPECTRUM_PRO,`
	`IDT_DEVICE_MINISMART_II,`
	`IDT_DEVICE_L100,`
	`IDT_DEVICE_UNIPAY,`
	`IDT_DEVICE_UNIPAY_I_V,`
	`IDT_DEVICE_VP3300_AJ,`
	`IDT_DEVICE_KIOSK_III,`
	`IDT_DEVICE_KIOSK_III_S,`
	`IDT_DEVICE_PIP_READER,`
	`IDT_DEVICE_VENDI,`
	`IDT_DEVICE_VP3300_USB,`
	`IDT_DEVICE_UNIPAY_I_V_TTK,`
	`IDT_DEVICE_VP3300_BT,`
	`IDT_DEVICE_VP8800,`
	`IDT_DEVICE_SREDKEY2_HID,`
	`IDT_DEVICE_SREDKEY2_KB,`
	`IDT_DEVICE_NEO2,`
	`IDT_DEVICE_MINISMART_II_COM = IDT_DEVICE_NEO2+5,`
	`IDT_DEVICE_SPECTRUM_PRO_COM,`
	`IDT_DEVICE_KIOSK_III_COM,`
	`IDT_DEVICE_KIOSK_III_S_COM,`
	`IDT_DEVICE_VP3300_COM,`
	`IDT_DEVICE_NEO2_COM,`
	`IDT_DEVICE_MAX_DEVICES = IDT_DEVICE_NEO2_COM+5`
	`};`

Returns

RETURN_CODE: 1: success, 0: failed

12.7.4.44 int device_setMerchantRecord (int *index*, int *enabled*, char * *merchantID*, char * *merchantURL*)

Set Merchant Record - NEO 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 device_getIDGStatusCodeString()

12.7.4.45 int device_setPollMode (IN BYTE *mode*)

Set Poll Mode - NEO

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 device_getIDGStatusCodeString()

12.7.4.46 void device_setSDKWaitTime (int *waitTime*)

Set SDK Wait Time

Set the SDK wait time for transactions

Parameters

waitTime	The SDK wait time for transaction in seconds

12.7.4.47 void emv_registerCallBk (pEMV_callBack *pEMVf*)

To register the emv callback function to get the EMV processing response. (Pass NULL to disable the callback.)

12.7.4.48 void parseMSRData (IN BYTE * *resData*, IN int *resLen*, IN_OUT IDTMSRData * *cardData*)

Parser the MSR data from the buffer into IDTMSTData structure

Parameters

resData	MSR card data buffer
resLen	the length of resData
cardData	the parser result with IDTMSTData structure

12.7.4.49 void pin_registerCallBk (pPIN_callBack *pPINf*)

To register the pin callback function to get the PINPad data. (Pass NULL to disable the callback.)

12.7.4.50 void registerHotplugCallBk (pMessageHotplug *pMsgHotplug*)

To register the USB HID hot-plug callback function which implemented in the application to monitor the hotplug message from the SDK.

12.7.4.51 void registerLogCallBk (pSendDataLog *pFSend,* pReadDataLog *pFRead*)

To register the log callback function which implemented in the application to monitor sending/reading data between application and reader.

12.7.4.52 int rs232_device_init (int *deviceType,* int *port_number,* int *brate*)

Initial the device by RS232

It will try to connect to the device with provided deviceType, port_number, and brate.

Parameters

deviceType	Device to connect to
port_number	Port number of the device

Port nr. | **Linux** | **Windows**

| 0 | ttyS0 | COM1 | | 1 | ttyS1 | COM2 | | 2 | ttyS2 | COM3 | | 3 | ttyS3 | COM4 | | 4 | ttyS4 | COM5 | | 5 | ttyS5 | COM6 | | 6 | ttyS6 | COM7 | | 7 | ttyS7 | COM8 | | 8 | ttyS8 | COM9 | | 9 | ttyS9 | COM10 | | 10 | ttyS10 | COM11 | | 11 | ttyS11 | COM12 | | 12 | ttyS12 | COM13 | | 13 | ttyS13 | COM14 | | 14 | ttyS14 | COM15 | | 15 | ttyS15 | COM16 | | 16 | ttyUSB0 | n.a. | | 17 | ttyUSB1 | n.a. | | 18 | ttyUSB2 | n.a. | | 19 | ttyUSB3 | n.a. | | 20 | ttyUSB4 | n.a. | | 21 | ttyUSB5 | n.a. | | 22 | ttyAMA0 | n.a. | | 23 | ttyAMA1 | n.a. | | 24 | ttyACM0 | n.a. | | 25 | ttyACM1 | n.a. | | 26 | rfcomm0 | n.a. | | 27 | rfcomm1 | n.a. | | 28 | ircomm0 | n.a. | | 29 | ircomm1 | n.a. | | 30 | cuau0 | n.a. | | 31 | cuau1 | n.a. | | 32 | cuau2 | n.a. | | 33 | cuau3 | n.a. | | 34 | cuaU0 | n.a. | | 35 | cuaU1 | n.a. | | 36 | cuaU2 | n.a. | | 37 | cuaU3 | n.a. |

Parameters

brate	Bitrate of the device

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.7.4.53 char* SDK_Version ( )

To Get SDK version

Returns

return the SDK version string

12.7.4.54 int setAbsoluteLibraryPath (const char * *absoluteLibraryPath*)

Set the path to use when searching for ID TECH's libraries. If this is not set, the libraries will be searched for with the system's default procedures.

Parameters

absoluteLibrary-Path	The absolute path to ID TECH's libraries.

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.8 Source_C/libIDT_SpectrumPro.h File Reference

SpectrumPro API.

```
#include "IDTDef.h"
```

Macros

- #define IN
- #define OUT
- #define IN_OUT

Typedefs

- typedef void(∗ pMessageHotplug)(int, int)
- typedef void(∗ pSendDataLog)(unsigned char ∗, int)
- typedef void(∗ pReadDataLog)(unsigned char ∗, int)
- typedef void(∗ pEMV_callBack)(int, int, unsigned char ∗, int, IDTTransactionData ∗, EMV_Callback ∗, int)
- typedef void(∗ pMSR_callBack)(int, IDTMSRData)
- typedef void(∗ pMSR_callBackp)(int, IDTMSRData ∗)
- typedef void(∗ pPIN_callBack)(int, IDTPINData ∗)
- typedef void(∗ pCMR_callBack)(int, IDTCMRData ∗)
- typedef void(∗ pCSFS_callBack)(BYTE status)
- typedef void(∗ ftpComm_callBack)(int, int, int)
- typedef void(∗ httpComm_callBack)(BYTE ∗, int)
- typedef void(∗ v4Comm_callBack)(BYTE, BYTE, BYTE ∗, int)

Functions

- void registerHotplugCallBk (pMessageHotplug pMsgHotplug)
- void registerLogCallBk (pSendDataLog pFSend, pReadDataLog pFRead)
- void emv_registerCallBk (pEMV_callBack pEMVf)
- void msr_registerCallBk (pMSR_callBack pMSRf)
- void msr_registerCallBkp (pMSR_callBackp pMSRf)
- void pin_registerCallBk (pPIN_callBack pPINf)
- void device_registerCameraCallBk (pCMR_callBack pCMRf)
- void device_registerCardStatusFrontSwitchCallBk (pCSFS_callBack pCSFSf)
- char ∗ SDK_Version ()
- int setAbsoluteLibraryPath (const char ∗absoluteLibraryPath)
- int device_init ()
- int rs232_device_init (int deviceType, int port_number, int brate)
- int device_setCurrentDevice (int deviceType)
- int device_close ()

- void device_getResponseCodeString (IN int returnCode, OUT char *despcrition)
- int device_isConnected ()
- int device_isAttached (int deviceType)
- int device_getFirmwareVersion (OUT char *firmwareVersion)
- int device_getFirmwareVersion_Len (OUT char *firmwareVersion, IN_OUT int *firmwareVersionLen)
- int device_getCurrentDeviceType ()
- int device_SendDataCommand (IN BYTE *cmd, IN int cmdLen, IN BYTE *data, IN int dataLen, OUT BYTE *response, IN_OUT int *respLen)
- int device_rebootDevice ()
- int device_updateFirmware (IN BYTE *firmwareData, IN int firmwareDataLen, IN char *firmwareName, IN int encryptionType, IN BYTE *keyBlob, IN int keyBlobLen)
- int config_getModelNumber (OUT char *sNumber)
- int config_getModelNumber_Len (OUT char *sNumber, IN_OUT int *sNumberLen)
- int config_getSerialNumber (OUT char *sNumber)
- int config_getSerialNumber_Len (OUT char *sNumber, IN_OUT int *sNumberLen)
- int device_pollCardReader (OUT BYTE *status)
- int device_pollCardReader_Len (OUT BYTE *status, IN_OUT int *statusLen)
- int device_getSpectrumProKSN (IN int type, OUT BYTE *KSN)
- int device_getSpectrumProKSN_Len (IN int type, OUT BYTE *KSN, IN_OUT int *KSNLen)
- int device_getSDKWaitTime ()
- void device_setSDKWaitTime (int waitTime)
- int device_getThreadStackSize ()
- void device_setThreadStackSize (int threadSize)
- int icc_powerOnICC (OUT BYTE *ATR, IN_OUT int *inLen)
- int icc_powerOffICC ()
- int icc_getICCReaderStatus (OUT BYTE *status)
- int emv_getEMVKernelVersion (OUT char *version)
- int emv_getEMVKernelVersion_Len (OUT char *version, IN_OUT int *versionLen)
- int emv_getEMVKernelCheckValue (OUT BYTE *checkValue, IN_OUT int *checkValueLen)
- void emv_setAutoAuthenticateTransaction (IN int authenticate)
- void emv_setAutoCompleteTransaction (IN int complete)
- int emv_getAutoAuthenticateTransaction ()
- int emv_getAutoCompleteTransaction ()
- int emv_getEMVConfigurationCheckValue (OUT BYTE *checkValue, IN_OUT int *checkValueLen)
- void emv_allowFallback (IN int allow)
- int emv_startTransaction (IN double amount, IN double amtOther, IN int exponent, IN int type, IN int timeout, IN BYTE *tags, IN int tagsLen, IN int forceOnline)
- int emv_activateTransaction (IN int timeout, IN BYTE *tags, IN int tagsLen, IN int forceOnline)
- int emv_authenticateTransaction (IN BYTE *updatedTLV, IN int updatedTLVLen)
- int emv_authenticateTransactionWithTimeout (IN int timeout, IN BYTE *updatedTLV, IN int updatedTLVLen)
- int emv_completeTransaction (IN int commError, IN BYTE *authCode, IN int authCodeLen, IN BYTE *iad, IN int iadLen, IN BYTE *tlvScripts, IN int tlvScriptsLen, IN BYTE *tlv, IN int tlvLen)
- int emv_cancelTransaction ()
- int emv_retrieveTransactionResult (IN BYTE *tags, IN int tagsLen, IDTTransactionData *cardData)
- int emv_callbackResponseLCD (IN int type, byte selection)
- int emv_callbackResponseMSR (IN BYTE *MSR, IN_OUT int MSRLen)
- int emv_retrieveApplicationData (IN BYTE *AID, IN int AIDLen, OUT BYTE *tlv, IN_OUT int *tlvLen)
- int emv_setApplicationData (IN BYTE *name, IN int nameLen, IN BYTE *tlv, IN int tlvLen)
- int emv_removeApplicationData (IN BYTE *AID, IN int AIDLen)
- int emv_removeAllApplicationData ()
- int emv_retrieveAIDList (OUT BYTE *AIDList, IN_OUT int *AIDListLen)
- int emv_retrieveTerminalData (OUT BYTE *tlv, IN_OUT int *tlvLen)
- int emv_setTerminalData (IN BYTE *tlv, IN int tlvLen)
- int emv_removeTerminalData ()
- int emv_retrieveCAPK (IN BYTE *capk, IN int capkLen, OUT BYTE *key, IN_OUT int *keyLen)

- int emv_setCAPK (IN BYTE *capk, IN int capkLen)
- int emv_removeCAPK (IN BYTE *capk, IN int capkLen)
- int emv_removeAllCAPK ()
- int emv_retrieveCAPKList (OUT BYTE *keys, IN_OUT int *keysLen)
- int emv_retrieveTerminalID (OUT char *terminalID)
- int emv_retrieveTerminalID_Len (OUT char *terminalID, IN_OUT int *terminalIDLen)
- int emv_setTerminalID (IN char *terminalID)
- int emv_retrieveCRL (OUT BYTE *list, IN_OUT int *lssLen)
- int emv_setCRL (IN BYTE *list, IN int lsLen)
- int emv_removeCRL (IN BYTE *list, IN int lsLen)
- int emv_removeAllCRL ()
- int msr_clearMSRData ()
- int msr_getMSRData (OUT BYTE *reData, IN_OUT int *reLen)
- int msr_cancelMSRSwipe ()
- int msr_startMSRSwipe (IN int _timeout)
- void parseMSRData (IN BYTE *resData, IN int resLen, IN_OUT IDTMSRData *cardData)
- int pin_getPIN (IN int mode, IN int PANSource, IN char *iccPAN, IN int IN iccPANLen, int startTimeout, IN int entryTimeout, IN char *language, IN int languageLen)
- int pin_cancelPINEntry ()
- void parsePINBlockData (IN BYTE *resData, IN int resLen, IN_OUT IDTPINData *cardData)
- void parsePINData (IN BYTE *resData, IN int resLen, IN_OUT IDTPINData *cardData)

12.8.1 Detailed Description

SpectrumPro API. Spectrum Pro Global API methods.

12.8.2 Macro Definition Documentation

12.8.2.1 #define IN

INPUT parameter.

12.8.2.2 #define IN_OUT

INPUT / OUTPUT PARAMETER.

12.8.2.3 #define OUT

OUTPUT parameter.

12.8.3 Typedef Documentation

12.8.3.1 typedef void(* ftpComm_callBack)(int, int, int)

Define the comm callback function to get FTP file transfer status

It should be passed as a parameter in a FTP request, Signature (int, int, int) = response code, current block, total blocks RESPONSE CODES: 100 = FILE DOWNLOAD STARTED 101 = FILE BLOCK XX OF XX RECEIVED 102 = FILE DOWNLOAD COMPLETED 103 = FILE DOWNLOAD TERMINATED PREMATURELY

12.8.3.2 typedef void(∗ httpComm_callBack)(BYTE ∗, int)

Define the comm callback function to get the async url data

It should be registered using the comm_registerHTTPCallback

12.8.3.3 typedef void(∗ pCMR_callBack)(int, IDTCMRData ∗)

Define the camera callback function to get the image data

It should be registered using the device_registerCameraCallBk,

12.8.3.4 typedef void(∗ pCSFS_callBack)(BYTE status)

Define the card status and front switch callback function to get card and front switch status

It should be registered using the device_registerCardStatusFrontSwitchCallBk,

12.8.3.5 typedef void(∗ pEMV_callBack)(int, int, unsigned char ∗, int, IDTTransactionData ∗, EMV_Callback ∗, int)

Define the EMV callback function to get the transaction message/data/result.

It should be registered using the emv_registerCallBk,

12.8.3.6 typedef void(∗ pMessageHotplug)(int, int)

Define the USB hot-plug callback function to monitor the info when plug in/out the reader.

It should be registered using the registerHotplugCallBk, The first integer parameter is device type, and the second integer parameter is either 0: Device Plugged Out or 1: Device Plugged In

12.8.3.7 typedef void(∗ pMSR_callBack)(int, IDTMSRData)

Define the MSR callback function to get the MSR card data

It should be registered using the msr_registerCallBk, this callback function is for backward compatibility

12.8.3.8 typedef void(∗ pMSR_callBackp)(int, IDTMSRData ∗)

Define the MSR callback function to get pointer to the MSR card data

It should be registered using the msr_registerCallBk, this callback function is recommended instead of pMSR_callBack

12.8.3.9 typedef void(∗ pPIN_callBack)(int, IDTPINData ∗)

Define the PINPad callback function to get the input PIN Pad data

It should be registered using the pin_registerCallBk,

12.8.3.10 typedef void(∗ pReadDataLog)(unsigned char ∗, int)

Define the read response callback function to monitor the reading response from the reader.

It should be registered using the registerLogCallBk,

12.8.3.11 typedef void(∗ pSendDataLog)(unsigned char ∗, int)

Define the send command callback function to monitor the sending command into the reader.

It should be registered using the registerLogCallBk,

12.8.3.12 typedef void(∗ v4Comm_callBack)(BYTE, BYTE, BYTE ∗, int)

Define the comm callback function to receive the V4 Protocol packets received by the device from an external source (IP/USB/RS-232) It should be registered using the comm_registerV4Callback, Data callback will contain command, sub-command, and data from V4 packet

12.8.4 Function Documentation

12.8.4.1 int config_getModelNumber (OUT char ∗ *sNumber*)

DEPRECATED : please use config_getModelNumber_Len(OUT char∗ sNumber, IN_OUT int ∗sNumberLen)

Polls device for Model Number

Parameters

sNumber	Returns Model Number; needs to have at least 64 bytes of memory

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.8.4.2 int config_getModelNumber_Len (OUT char ∗ *sNumber,* IN_OUT int ∗ *sNumberLen*)

Polls device for Model Number

Parameters

sNumber	Returns Model Number
sNumber	length of Model Number

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.8.4.3 int config_getSerialNumber (OUT char ∗ *sNumber*)

DEPRECATED : please use config_getSerialNumber_Len(OUT char∗ sNumber, IN_OUT int ∗sNumberLen)

Polls device for Serial Number

Parameters

sNumber	Returns Serial Number; needs to have at least 64 bytes of memory

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.8.4.4 int config_getSerialNumber_Len (OUT char ∗ *sNumber,* IN_OUT int ∗ *sNumberLen*)

Polls device for Serial Number

Parameters

sNumber	Returns Serial Number
sNumberLen	Length of Serial Number

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.8.4.5 int device_close ( )

Close the device

Returns

RETURN_CODE: 0: success, 0x0A: failed

12.8.4.6 int device_getCurrentDeviceType ( )

Get current active device type

Returns

: return the device type defined as DEVICE_TYPE in the IDTDef.h

12.8.4.7 int device_getFirmwareVersion (OUT char * *firmwareVersion*)

DEPRECATED : please use device_getFirmwareVersion_Len(OUT char* firmwareVersion, IN_OUT int *firmware-VersionLen)

Polls device for Firmware Version

Parameters

firmwareVersion	Response returned of Firmware Version; needs to have at least 128 bytes of memory

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.8.4.8 int device_getFirmwareVersion_Len (OUT char * *firmwareVersion,* IN_OUT int * *firmwareVersionLen*)

Polls device for Firmware Version

Parameters

firmwareVersion	Response returned of Firmware Version
firmwareVersion-Len	Length of Firmware Version

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.8.4.9 void device_getResponseCodeString (IN int *returnCode,* OUT char * *despcrition*)

Review the return code description.

Parameters

returnCode	the response result.
description	

Return values

the	string for description of response result

- 0: "no error, beginning task";
- 1: "no response from reader";
- 2: "invalid response data";
- 3: "time out for task or CMD";
- 4: "wrong parameter";
- 5: "SDK is doing MSR or ICC task";
- 6: "SDK is doing PINPad task";
- 7: "SDK is doing CTLS task";
- 8: "SDK is doing EMV task";
- 9: "SDK is doing Other task";
- 10: "err response or data";
- 11: "no reader attached";
- 12: "mono audio is enabled";
- 13: "did connection";
- 14: "audio volume is too low";
- 15: "task or CMD be canceled";
- 16: "UF wrong string format";
- 17: "UF file not found";
- 18: "UF wrong file format";
- 19: "Attempt to contact online host failed";
- 20: "Attempt to perform RKI failed";
- 22: "Buffer size is not enough";
- 0X300: "Key Type(TDES) of Session Key is not same as the related Master Key.";
- 0X400: "Related Key was not loaded.";
- 0X500: "Key Same.";
- 0X501: "Key is all zero";
- 0X502: "TR-31 format error";
- 0X702: "PAN is Error Key.";
- 0X705: "No Internal MSR PAN (or Internal MSR PAN is erased timeout)";
- 0X0C01: "Incorrect Frame Tag";

- 0X0C02: "Incorrect Frame Type";
- 0X0C03: "Unknown Frame Type";
- 0X0C04: "Unknown Command";
- 0X0C05: "Unknown Sub-Command";
- 0X0C06: "CRC Error";
- 0X0C07: "Failed";
- 0X0C08: "Timeout";
- 0X0C0A: "Incorrect Parameter";
- 0X0C0B: "Command Not Supported";
- 0X0C0C: "Sub-Command Not Supported";
- 0X0C0D: "Parameter Not Supported / Status Abort Command";
- 0X0C0F: "Sub-Command Not Allowed";
- 0X0D01: "Incorrect Header Tag";
- 0X0D02: "Unknown Command";
- 0X0D03: "Unknown Sub-Command";
- 0X0D04: "CRC Error in Frame";
- 0X0D05: "Incorrect Parameter";
- 0X0D06: "Parameter Not Supported";
- 0X0D07: "Mal-formatted Data";
- 0X0D08: "Timeout";
- 0X0D0A: "Failed / NACK";
- 0X0D0B: "Command not Allowed";
- 0X0D0C: "Sub-Command not Allowed";
- 0X0D0D: "Buffer Overflow (Data Length too large for reader buffer)";
- 0X0D0E: "User Interface Event";
- 0X0D11: "Communication type not supported, VT-1, burst, etc.";
- 0X0D12: "Secure interface is not functional or is in an intermediate state.";
- 0X0D13: "Data field is not mod 8";
- 0X0D14: "Pad - 0X80 not found where expected";
- 0X0D15: "Specified key type is invalid";
- 0X0D1: "Could not retrieve key from the SAM(InitSecureComm)";
- 0X0D17: "Hash code problem";
- 0X0D18: "Could not store the key into the SAM(InstallKey)";
- 0X0D19: "Frame is too large";
- 0X0D1A: "Unit powered up in authentication state but POS must resend the InitSecureComm command";
- 0X0D1B: "The EEPROM may not be initialized because SecCommInterface does not make sense";

- 0X0D1C: "Problem encoding APDU";
- 0X0D20: "Unsupported Index(ILM) SAM Transceiver error - problem communicating with the SAM(Key Mgr)";
- 0X0D2: "Unexpected Sequence Counter in multiple frames for single bitmap(ILM) Length error in data returned from the SAM(Key Mgr)";
- 0X0D22: "Improper bit map(ILM)";
- 0X0D23: "Request Online Authorization";
- 0X0D24: "ViVOCard3 raw data read successful";
- 0X0D25: "Message index not available(ILM) ViVOcomm activate transaction card type(ViVOcomm)";
- 0X0D26: "Version Information Mismatch(ILM)";
- 0X0D27: "Not sending commands in correct index message index(ILM)";
- 0X0D28: "Time out or next expected message not received(ILM)";
- 0X0D29: "ILM languages not available for viewing(ILM)";
- 0X0D2A: "Other language not supported(ILM)";
- 0X0D41: "Unknown Error from SAM";
- 0X0D42: "Invalid data detected by SAM";
- 0X0D43: "Incomplete data detected by SAM";
- 0X0D44: "Reserved";
- 0X0D45: "Invalid key hash algorithm";
- 0X0D46: "Invalid key encryption algorithm";
- 0X0D47: "Invalid modulus length";
- 0X0D48: "Invalid exponent";
- 0X0D49: "Key already exists";
- 0X0D4A: "No space for new RID";
- 0X0D4B: "Key not found";
- 0X0D4C: "Crypto not responding";
- 0X0D4D: "Crypto communication error";
- 0X0D4E: "Module-specific error for Key Manager";
- 0X0D4F: "All key slots are full (maximum number of keys has been installed)";
- 0X0D50: "Auto-Switch OK";
- 0X0D51: "Auto-Switch failed";
- 0X0D90: "Account DUKPT Key not exist";
- 0X0D91: "Account DUKPT Key KSN exausted";
- 0X0D00: "This Key had been loaded.";
- 0X0E00: "Base Time was loaded.";
- 0X0F00: "Encryption Or Decryption Failed.";
- 0X1000: "Battery Low Warning (It is High Priority Response while Battery is Low.)";

- 0X1800: "Send 'Cancel Command' after send 'Get Encrypted PIN' & 'Get Numeric' & 'Get Amount'; - 0X1900: "Press 'Cancel' key after send 'Get Encrypted PIN' & 'Get Numeric' & 'Get Amount';
- 0X30FF: "Security Chip is not connect";
- 0X3000: "Security Chip is deactivation & Device is In Removal Legally State.";
- 0X3101: "Security Chip is activation & Device is In Removal Legally State.";
- 0X5500: "No Admin DUKPT Key.";
- 0X5501: "Admin DUKPT Key STOP.";
- 0X5502: "Admin DUKPT Key KSN is Error.";
- 0X5503: "Get Authentication Code1 Failed.";
- 0X5504: "Validate Authentication Code Error.";
- 0X5505: "Encrypt or Decrypt data failed.";
- 0X5506: "Not Support the New Key Type.";
- 0X5507: "New Key Index is Error.";
- 0X5508: "Step Error.";
- 0X5509: "KSN Error";
- 0X550A: "MAC Error.";
- 0X550B: "Key Usage Error.";
- 0X550C: "Mode Of Use Error.";
- 0X550F: "Other Error.";
- 0X6000: "Save or Config Failed / Or Read Config Error.";
- 0X6200: "No Serial Number.";
- 0X6900: "Invalid Command - Protocol is right, but task ID is invalid.";
- 0X6A01: "Unsupported Command - Protocol and task ID are right, but command is invalid - In this State";
- 0X6A00: "Unsupported Command - Protocol and task ID are right, but command is invalid.";
- 0X6B00: "Unknown parameter in command - Protocol task ID and command are right, but parameter is invalid.";
- 0X6C00: "Unknown parameter in command - Protocol task ID and command are right, but length is out of the requirement.";
- 0X7200: "Device is suspend (MKSK suspend or press password suspend).";
- 0X7300: "PIN DUKPT is STOP (21 bit 1).";
- 0X7400: "Device is Busy.";
- 0XE100: "Can not enter sleep mode";
- 0XE200: "File has existed";
- 0XE300: "File has not existed";
- 0XE313: "IO line low -- Card error after session start";
- 0XE400: "Open File Error";
- 0XE500: "SmartCard Error";

- 0XE600: "Get MSR Card data is error";
- 0XE700: "Command time out";
- 0XE800: "File read or write is error";
- 0XE900: "Active 1850 error!";
- 0XEA00: "Load bootloader error";
- 0XEF00: "Protocol Error- STX or ETX or check error.";
- 0XEB00: "Picture is not exist";
- 0X2C02: "No Microprocessor ICC seated";
- 0X2C06: "no card seated to request ATR";
- 0X2D01: "Card Not Supported,";
- 0X2D03: "Card Not Supported, wants CRC";
- 0X690D: "Command not supported on reader without ICC support";
- 0X8100: "ICC error time out on power-up";
- 0X8200: "invalid TS character received - Wrong operation step";
- 0X8300: "Decode MSR Error";
- 0X8400: "TriMagII no Response";
- 0X8500: "No Swipe MSR Card";
- 0X8510: "No Financial Card";
- 0X8600: "Unsupported F, D, or combination of F and D";
- 0X8700: "protocol not supported EMV TD1 out of range";
- 0X8800: "power not at proper level";
- 0X8900: "ATR length too long";
- 0X8B01: "EMV invalid TA1 byte value";
- 0X8B02: "EMV TB1 required";
- 0X8B03: "EMV Unsupported TB1 only 00 allowed";
- 0X8B04: "EMV Card Error, invalid BWI or CWI";
- 0X8B06: "EMV TB2 not allowed in ATR";
- 0X8B07: "EMV TC2 out of range";
- 0X8B08: "EMV TC2 out of range";
- 0X8B09: "per EMV96 TA3 must be $>$ - 0XF";
- 0X8B10: "ICC error on power-up";
- 0X8B11: "EMV T=1 then TB3 required";
- 0X8B12: "Card Error, invalid BWI or CWI";
- 0X8B13: "Card Error, invalid BWI or CWI";
- 0X8B17: "EMV TC1/TB3 conflict-";
- 0X8B20: "EMV TD2 out of range must be T=1";

- 0X8C00: "TCK error";
- 0XA304: "connector has no voltage setting";
- 0XA305: "ICC error on power-up invalid (SBLK(IFSD) exchange";
- 0XE301: "ICC error after session start";
- 0XFF00: "Request to go online";
- 0XFF01: "EMV: Accept the offline transaction";
- 0XFF02: "EMV: Decline the offline transaction";
- 0XFF03: "EMV: Accept the online transaction";
- 0XFF04: "EMV: Decline the online transaction";
- 0XFF05: "EMV: Application may fallback to magstripe technology";
- 0XFF06: "EMV: ICC detected tah the conditions of use are not satisfied";
- 0XFF07: "EMV: ICC didn't accept transaction";
- 0XFF08: "EMV: Transaction was cancelled";
- 0XFF09: "EMV: Application was not selected by kernel or ICC format error or ICC missing data error";
- 0XFF0A: "EMV: Transaction is terminated";
- 0XFF0B: "EMV: Other EMV Error";
- 0XFFFF: "NO RESPONSE";
- 0XF002: "ICC communication timeout";
- 0XF003: "ICC communication Error";
- 0XF00F: "ICC Card Seated and Highest Priority, disable MSR work request";
- 0XF200: "AID List / Application Data is not exist";
- 0XF201: "Terminal Data is not exist";
- 0XF202: "TLV format is error";
- 0XF203: "AID List is full";
- 0XF204: "Any CA Key is not exist";
- 0XF205: "CA Key RID is not exist";
- 0XF206: "CA Key Index it not exist";
- 0XF207: "CA Key is full";
- 0XF208: "CA Key Hash Value is Error";
- 0XF209: "Transaction format error";
- 0XF20A: "The command will not be processing";
- 0XF20B: "CRL is not exist";
- 0XF20C: "CRL number exceed max number";
- 0XF20D: "Amount,Other Amount,Trasaction Type are missing";
- 0XF20E: "The Identification of algorithm is mistake";
- 0XF20F: "No Financial Card";

- 0XF210: "In Encrypt Result state, TLV total Length is greater than Max Length";
- 0X1001: "INVALID ARG";
- 0X1002: "FILE_OPEN_FAILED";
- 0X1003: "FILE OPERATION_FAILED";
- 0X2001: "MEMORY_NOT_ENOUGH";
- 0X3002: "SMARTCARD_FAIL";
- 0X3003: "SMARTCARD_INIT_FAILED";
- 0X3004: "FALLBACK_SITUATION";
- 0X3005: "SMARTCARD_ABSENT";
- 0X3006: "SMARTCARD_TIMEOUT";
- 0X3012: "EMV_RESULT_CODE_MSR_CARD_ERROR_FALLBACK";
- 0X5001: "EMV_PARSING_TAGS_FAILED";
- 0X5002: "EMV_DUPLICATE_CARD_DATA_ELEMENT";
- 0X5003: "EMV_DATA_FORMAT_INCORRECT";
- 0X5004: "EMV_NO_TERM_APP";
- 0X5005: "EMV_NO_MATCHING_APP";
- 0X5006: "EMV_MISSING_MANDATORY_OBJECT";
- 0X5007: "EMV_APP_SELECTION_RETRY";
- 0X5008: "EMV_GET_AMOUNT_ERROR";
- 0X5009: "EMV_CARD_REJECTED";
- 0X5010: "EMV_AIP_NOT_RECEIVED";
- 0X5011: "EMV_AFL_NOT_RECEIVED";
- 0X5012: "EMV_AFL_LEN_OUT_OF_RANGE";
- 0X5013: "EMV_SFI_OUT_OF_RANGE";
- 0X5014: "EMV_AFL_INCORRECT";
- 0X5015: "EMV_EXP_DATE_INCORRECT";
- 0X5016: "EMV_EFF_DATE_INCORRECT";
- 0X5017: "EMV_ISS_COD_TBL_OUT_OF_RANGE";
- 0X5018: "EMV_CRYPTOGRAM_TYPE_INCORRECT";
- 0X5019: "EMV_PSE_NOT_SUPPORTED_BY_CARD";
- 0X5020: "EMV_USER_SELECTED_LANGUAGE";
- 0X5021: "EMV_SERVICE_NOT_ALLOWED";
- 0X5022: "EMV_NO_TAG_FOUND";
- 0X5023: "EMV_CARD_BLOCKED";
- 0X5024: "EMV_LEN_INCORRECT";
- 0X5025: "CARD_COM_ERROR";

- 0X5026: "EMV_TSC_NOT_INCREASED";
- 0X5027: "EMV_HASH_INCORRECT";
- 0X5028: "EMV_NO_ARC";
- 0X5029: "EMV_INVALID_ARC";
- 0X5030: "EMV_NO_ONLINE_COMM";
- 0X5031: "TRAN_TYPE_INCORRECT";
- 0X5032: "EMV_APP_NO_SUPPORT";
- 0X5033: "EMV_APP_NOT_SELECT";
- 0X5034: "EMV_LANG_NOT_SELECT";
- 0X5035: "EMV_NO_TERM_DATA";
- 0X5039: "EMV_PIN_ENTRY_TIMEOUT";
- 0X6001: "CVM_TYPE_UNKNOWN";
- 0X6002: "CVM_AIP_NOT_SUPPORTED";
- 0X6003: "CVM_TAG_8E_MISSING";
- 0X6004: "CVM_TAG_8E_FORMAT_ERROR";
- 0X6005: "CVM_CODE_IS_NOT_SUPPORTED";
- 0X6006: "CVM_COND_CODE_IS_NOT_SUPPORTED";
- 0X6007: "NO_MORE_CVM";
- 0X6008: "PIN_BYPASSED_BEFORE";
- 0X7001: "PK_BUFFER_SIZE_TOO_BIG";
- 0X7002: "PK_FILE_WRITE_ERROR";
- 0X7003: "PK_HASH_ERROR";
- 0X8001: "NO_CARD_HOLDER_CONFIRMATION";
- 0X8002: "GET_ONLINE_PIN";
- 0XD000: "Data not exist";
- 0XD001: "Data access error";
- 0XD100: "RID not exist";
- 0XD101: "RID existed";
- 0XD102: "Index not exist";
- 0XD200: "Maximum exceeded";
- 0XD201: "Hash error";
- 0XD205: "System Busy";
- 0X0E01: "Unable to go online";
- 0X0E02: "Technical Issue";
- 0X0E03: "Declined";
- 0X0E04: "Issuer Referral transaction";

- 0X0F01: "Decline the online transaction";
- 0X0F02: "Request to go online";
- 0X0F03: "Transaction is terminated";
- 0X0F05: "Application was not selected by kernel or ICC format error or ICC missing data error";
- 0X0F07: "ICC didn't accept transaction";
- 0X0F0A: "Application may fallback to magstripe technology";
- 0X0F0C: "Transaction was cancelled";
- 0X0F0D: "Timeout";
- 0X0F0F: "Other EMV Error";
- 0X0F10: "Accept the offline transaction";
- 0X0F11: "Decline the offline transaction";
- 0X0F21: "ICC detected tah the conditions of use are not satisfied";
- 0X0F22: "No app were found on card matching terminal configuration";
- 0X0F23: "Terminal file does not exist";
- 0X0F24: "CAPK file does not exist";
- 0X0F25: "CRL Entry does not exist";
- 0X0FFE: "code when blocking is disabled";
- 0X0FFF: "code when command is not applicable on the selected device";
- 0XF005: "ICC Encrypted C-APDU Data Structure Length Error Or Format Error.";
- 0XBBE0: "CM100 Success";
- 0XBBE1: "CM100 Parameter Error";
- 0XBBE2: "CM100 Low Output Buffer";
- 0XBBE3: "CM100 Card Not Found";
- 0XBBE4: "CM100 Collision Card Exists";
- 0XBBE5: "CM100 Too Many Cards Exist";
- 0XBBE6: "CM100 Saved Data Does Not Exist";
- 0XBBE8: "CM100 No Data Available";
- 0XBBE9: "CM100 Invalid CID Returned";
- 0XBBEA: "CM100 Invalid Card Exists";
- 0XBBEC: "CM100 Command Unsupported";
- 0XBBED: "CM100 Error In Command Process";
- 0XBBEE: "CM100 Invalid Command";
- 0X9031: "Unknown command";
- 0X9032: "Wrong parameter (such as the length of the command is incorrect)";
- 0X9038: "Wait (the command couldnt be finished in BWT)";
- 0X9039: "Busy (a previously command has not been finished)";

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

12.8.4.10 int device_getSDKWaitTime ( )

Get SDK Wait Time

Get the SDK wait time for transactions

Returns

SDK wait time in seconds

12.8.4.11 int device_getSpectrumProKSN (IN int *type*, OUT BYTE * *KSN*)

DEPRECATED : please use device_getSpectrumProKSN_Len(IN int type, OUT BYTE * KSN, IN_OUT int *KSN-Len)

Get DUKPT KSN

Returns the KSN for the provided key index

Parameters

type	Key type: • 0: Key Encryption Key (Master Key or KEK) • 2: Data Encryption Key (DEK) • 5: MAC Key (MAK) • 10: RKL Key Encryption Key (REK) • 20: HSM DUKPT Key
KSN	Key Serial Number; needs to have at least 10 bytes of memory

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.8.4.12 int device_getSpectrumProKSN_Len (IN int *type*, OUT BYTE * *KSN*, IN_OUT int * *KSNLen*)

Get DUKPT KSN

Returns the KSN for the provided key index

Parameters

type	Key type: • 0: Key Encryption Key (Master Key or KEK) • 2: Data Encryption Key (DEK) • 5: MAC Key (MAK) • 10: RKL Key Encryption Key (REK) • 20: HSM DUKPT Key

KSN	Key Serial Number
KSNLen	Length of KSN

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.8.4.13 int device_getThreadStackSize ( )

Get Thread Stack Size

Get the stack size setting for newly created threads

Returns

Thread Stack Size

12.8.4.14 int device_init ( )

Initial the device by USB

It will detect the device and trying connect.

The connect status can be checked by device_isConnected().

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.8.4.15 int device_isAttached (int *deviceType*)

Check if the device is attached to the USB port The function device_init() must be called before this function.

Parameters

deviceType,the	device type of the USB device

Returns

1 if the device is attached, or 0 if the device is not attached

12.8.4.16 int device_isConnected ( )

Check the device conntected status

Returns

DEVICE_DISCONNECT=0, or DEVICE_CONNECTED = 1

12.8.4.17 int device_pollCardReader (OUT BYTE * *status*)

DEPRECATED : please use device_pollCardReader_Len(OUT BYTE * status, IN_OUT int *statusLen)

Poll Card Reader

Provides information about the state of the Card Reader

Parameters

status	Six bytes indicating card reader information Byte 0: • Bit 0: Device Manufacturing CA data valid • Bit 1: Device Manufacturing Secure data valid • Bit 2: HOST_CR_MASTER_DUKPT Key valid • Bit 3: HOST_CR_MAC Keys valid (Authenticated) • Bit 4: RFU • Bit 5: RFU • Bit 6: DATA_DUKPT Key Valid • Bit 7: Key is initialized (MFK and RSA Key pairs)

Byte 1:

- Bit 0: Firmware Key Valid
- Bit 1: RFU
- Bit 2: CR_PINPAD_MASTER_DUKPT Key valid
- Bit 3: CR_PINPAD_MAC Keys valid (Authenticated)
- Bit 4: DATA Pairing DUKPT Key valid
- Bit 5: PIN Pairing DUKPT Key Valid
- Bit 6: RFU
- Bit 7: RFU

Byte 2:

- Bit 0: RFU
- Bit 1: Tamper Switch #1 Error
- Bit 2: Battery Backup Error
- Bit 3: Temperature Error
- Bit 4: Voltage Sensor Error
- Bit 5: Firmware Authentication Error
- Bit 6: Tamper Switch #2 Error
- Bit 7: Removal Tamper Error

Byte 3:

- Battery Voltage (example 0x32 = 3.2V, 0x24 = 2.4V)

Byte 4:

- Bit 0: Log is Full
- Bit 1: Mag Data Present

- Bit 2: Card Insert
- Bit 3: Removal Sensor connected
- Bit 4: Card Seated
- Bit 5: Latch Mechanism Active
- Bit 6: Removal Sensor Active
- Bit 7: Tamper Detector Active

Byte 5:

- Bit 0: SAM Available
- Bit 1: Chip Card Reader Available
- Bit 2: Host Connected
- Bit 3: Contactless Available
- Bit 4: PINPAD connected
- Bit 5: MSR Header connected
- Bit 6: RFU
- Bit 7: Production Unit

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.8.4.18 int device_pollCardReader_Len (OUT BYTE * *status*, IN_OUT int * *statusLen*)

Poll Card Reader

Provides information about the state of the Card Reader

Parameters

status	Six bytes indicating card reader information Byte 0: • Bit 0: Device Manufacturing CA data valid • Bit 1: Device Manufacturing Secure data valid • Bit 2: HOST_CR_MASTER_DUKPT Key valid • Bit 3: HOST_CR_MAC Keys valid (Authenticated) • Bit 4: RFU • Bit 5: RFU • Bit 6: DATA_DUKPT Key Valid • Bit 7: Key is initialized (MFK and RSA Key pairs)

Byte 1:

- Bit 0: Firmware Key Valid
- Bit 1: RFU

- Bit 2: CR_PINPAD_MASTER_DUKPT Key valid
- Bit 3: CR_PINPAD_MAC Keys valid (Authenticated)
- Bit 4: DATA Pairing DUKPT Key valid
- Bit 5: PIN Pairing DUKPT Key Valid
- Bit 6: RFU
- Bit 7: RFU

Byte 2:

- Bit 0: RFU
- Bit 1: Tamper Switch #1 Error
- Bit 2: Battery Backup Error
- Bit 3: Temperature Error
- Bit 4: Voltage Sensor Error
- Bit 5: Firmware Authentication Error
- Bit 6: Tamper Switch #2 Error
- Bit 7: Removal Tamper Error

Byte 3:

- Battery Voltage (example 0x32 = 3.2V, 0x24 = 2.4V)

Byte 4:

- Bit 0: Log is Full
- Bit 1: Mag Data Present
- Bit 2: Card Insert
- Bit 3: Removal Sensor connected
- Bit 4: Card Seated
- Bit 5: Latch Mechanism Active
- Bit 6: Removal Sensor Active
- Bit 7: Tamper Detector Active

Byte 5:

- Bit 0: SAM Available
- Bit 1: Chip Card Reader Available
- Bit 2: Host Connected
- Bit 3: Contactless Available
- Bit 4: PINPAD connected
- Bit 5: MSR Header connected
- Bit 6: RFU
- Bit 7: Production Unit

Parameters

statusLen	Length of status

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.8.4.19 int 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()

12.8.4.20 void device_registerCameraCallBk (pCMR_callBack *pCMRf*)

To register the camera callback function to get the image data. (Pass NULL to disable the callback.)

12.8.4.21 void device_registerCardStatusFrontSwitchCallBk (pCSFS_callBack *pCSFSf*)

To register the card status and front switch callback function to get status. (Pass NULL to disable the callback.)

12.8.4.22 int device_SendDataCommand (IN BYTE * *cmd*, IN int *cmdLen*, IN BYTE * *data*, IN int *dataLen*, OUT BYTE * *response*, IN_OUT int * *respLen*)

Send a Command to device

Sends a command to the device .

Parameters

cmd	buffer of command to execute.
cmdLen,the	length of the buffer cmd.
data	buffer of IDG command data.
dataLen,the	length of the buffer data.
response	Response data
respLen,the	length of Response data

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.8.4.23 int device_setCurrentDevice (int *deviceType*)

Sets the current device to talk to

The connect status can be checked by device_isConnected().

Parameters

deviceType	Device to connect to
	
	`enum DEVICE_TYPE`
	`{`
	`IDT_DEVICE_UNKNOWN=0,`
	`IDT_DEVICE_AUGUSTA_HID,`
	`IDT_DEVICE_AUGUSTA_KB,`
	`IDT_DEVICE_AUGUSTA_S_HID,`
	`IDT_DEVICE_AUGUSTA_S_KB,`
	`IDT_DEVICE_AUGUSTA_S_TTK_HID,`
	`IDT_DEVICE_SPECTRUM_PRO,`
	`IDT_DEVICE_MINISMART_II,`
	`IDT_DEVICE_L100,`
	`IDT_DEVICE_UNIPAY,`
	`IDT_DEVICE_UNIPAY_I_V,`
	`IDT_DEVICE_VP3300_AJ,`
	`IDT_DEVICE_KIOSK_III,`
	`IDT_DEVICE_KIOSK_III_S,`
	`IDT_DEVICE_PIP_READER,`
	`IDT_DEVICE_VENDI,`
	`IDT_DEVICE_VP3300_USB,`
	`IDT_DEVICE_UNIPAY_I_V_TTK,`
	`IDT_DEVICE_VP3300_BT,`
	`IDT_DEVICE_VP8800,`
	`IDT_DEVICE_SREDKEY2_HID,`
	`IDT_DEVICE_SREDKEY2_KB,`
	`IDT_DEVICE_NEO2,`
	`IDT_DEVICE_MINISMART_II_COM = IDT_DEVICE_NEO2+5,`
	`IDT_DEVICE_SPECTRUM_PRO_COM,`
	`IDT_DEVICE_KIOSK_III_COM,`
	`IDT_DEVICE_KIOSK_III_S_COM,`
	`IDT_DEVICE_VP3300_COM,`
	`IDT_DEVICE_NEO2_COM,`
	`IDT_DEVICE_MAX_DEVICES = IDT_DEVICE_NEO2_COM+5`
	`};`

Returns

RETURN_CODE: 1: success, 0: failed

12.8.4.24 void device_setSDKWaitTime (int *waitTime*)

Set SDK Wait Time

Set the SDK wait time for transactions

Parameters

waitTime	The SDK wait time for transaction in seconds

12.8.4.25 void device_setThreadStackSize (int *threadSize*)

Set Thread Stack Size

Set the stack size setting for newly created threads

12.8.4.26 int device_updateFirmware (IN BYTE * *firmwareData,* IN int *firmwareDataLen,* IN char * *firmwareName,* IN int *encryptionType,* IN BYTE * *keyBlob,* IN int *keyBlobLen*)

Update Firmware Updates the firmware of the Spectrum Pro K21 HUB or Maxq1050.

Parameters

firmwareData	Signed binary data of a firmware file provided by IDTech
firmwareDataLen	Length of firmwareData
firmwareName	Firmware name. Must be one of the following two strings (with appropriate version information) • "SP K21 APP Vx.xx.xxx" • "SP MAX APP Vx.xx.xxx"
encryptionType	Encryption type • 0 : Plaintext • 1 : TDES ECB, PKCS#5 padding • 2 : TDES CBC, PKCS#5, IV is all 0
keyBlob	Encrypted firmware session key blob, TR-31 Rev B, wrapped by FW Key (Optional, none if firmware is plaintext)
keyBlobLen	Length of keyBlob

Returns

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

Firmware update status is returned in the callback with the following values: sender = SPECTRUM_PRO state = DeviceState.FirmwareUpdate data = File Progress. Two bytes, with byte[0] = current block, and byte[1] = total blocks. 0x0310 = block 3 of 16 transactionResultCode:

- RETURN_CODE_DO_SUCCESS = Firmware Update Completed Successfully
- RETURN_CODE_BLOCK_TRANSFER_SUCCESS = Current block transferred successfully
- Any other return code represents an error condition

12.8.4.27 int emv_activateTransaction (IN int *timeout,* IN BYTE * *tags,* IN int *tagsLen,* IN int *forceOnline*)

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	Tags to be included in the request. Passed as a TLV stream. Example, tag 9F0C with amount 0x000000000100 would be 0x9F0C06000000000100
tagsLen	Length of tags
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

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString >>>>>IMPORTANT: parameters for amount and amtOther MUST BE PASSED AS A DOUBLE VALUE WITH DECIMAL POINT. Example, do not pass 1, but instead pass 1.0 or 1.00. Otherwise, results will be unpredictable

12.8.4.28 void emv_allowFallback (IN int *allow*)

Allow fallback for EMV transactions. Default is TRUE

Parameters

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

12.8.4.29 int emv_authenticateTransaction (IN BYTE * *updatedTLV,* IN int *updatedTLVLen*)

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.

Parameters

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, 9F37 to be included in response = DFEE1-A079F029F36959F37
updatedTLVLen	

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.8.4.30 int emv_authenticateTransactionWithTimeout (IN int *timeout,* IN BYTE * *updatedTLV,* IN int *updatedTLVLen*)

Authenticate EMV Transaction Request with Timeout

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.

Parameters

timeout	Timeout value in seconds.
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, 9F37 to be included in response = DFEE1-A079F029F36959F37
updatedTLVLen	

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.8.4.31 int emv_callbackResponseLCD (IN int *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_DISPLAY_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 device_getResponseCodeString()

12.8.4.32 int emv_callbackResponseMSR (IN BYTE * *MSR*, IN_OUT int *MSRLen*)

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
MSRLen	the length of Swiped track data

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.8.4.33 int emv_cancelTransaction ( )

Cancel EMV Transaction

Cancels the currently executing EMV transaction.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.8.4.34 int emv_completeTransaction (IN int *commError*, IN BYTE * *authCode*, IN int *authCodeLen*, IN BYTE * *iad*, IN int *iadLen*, IN BYTE * *tlvScripts*, IN int *tlvScriptsLen*, IN BYTE * *tlv*, IN int *tlvLen*)

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(1) if host was unreachable, or FALSE(0) 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
authCodeLen	the length of authCode
iad	Issuer Authentication Data, if any. Example 0x11223344556677883030 (tag value 91).
iadLen	the length of iadLen
tlvScripts	71/72 scripts, if any
tlvScriptsLen	the length of tlvScriptsLen
tlv	Additional TLV data to return with transaction results (if any)
tlvLen	the length of tlv

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.8.4.35 int emv_getAutoAuthenticateTransaction ( )

Gets auto authenticate value for EMV transactions.

Returns

RETURN_CODE: TRUE = auto authenticate, FALSE = manually authenticate

12.8.4.36 int emv_getAutoCompleteTransaction ( )

Gets auto complete value for EMV transactions.

Returns

RETURN_CODE: TRUE = auto complete, FALSE = manually complete

12.8.4.37 int emv_getEMVConfigurationCheckValue (OUT BYTE * *checkValue,* IN_OUT int * *checkValueLen*)

Get EMV Kernel configuration check value info

Parameters

checkValue	Response returned of Kernel configuration check value info
checkValueLen	the length of checkValue

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.8.4.38 int emv_getEMVKernelCheckValue (OUT BYTE * *checkValue,* IN_OUT int * *checkValueLen*)

Get EMV Kernel check value info

Parameters

checkValue	Response returned of Kernel check value info
checkValueLen	the length of checkValue

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.8.4.39 int emv_getEMVKernelVersion (OUT char * *version*)

DEPRECATED : please use emv_getEMVKernelVersion_Len(OUT char* version, IN_OUT int *versionLen)

Polls device for EMV Kernel Version

Parameters

version	Response returned of Kernel Version; needs to have at least 128 bytes of memory.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.8.4.40 int emv_getEMVKernelVersion_Len (OUT char * *version*, IN_OUT int * *versionLen*)

Polls device for EMV Kernel Version

Parameters

version	Response returned of Kernel Version
versionLen	Length of version

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.8.4.41 void emv_registerCallBk (pEMV_callBack *pEMVf*)

To register the emv callback function to get the EMV processing response. (Pass NULL to disable the callback.)

12.8.4.42 int emv_removeAllApplicationData ( )

Remove All Application Data

Removes all the Application Data

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.8.4.43 int emv_removeAllCAPK ( )

Remove All Certificate Authority Public Key

Removes all the CAPK

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.8.4.44 int emv_removeAllCRL ( )

Remove All Certificate Revocation List Entries

Removes all CRLEntry entries

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.8.4.45 int emv_removeApplicationData (IN BYTE * *AID*, IN int *AIDLen*)

Remove Application Data by AID

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

Parameters

AID	Name of ApplicationID Must be between 5 and 16 bytes
AIDLen	the length of AID data buffer

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.8.4.46 int emv_removeCAPK (IN BYTE * *capk*, IN int *capkLen*)

Remove Certificate Authority Public Key

Removes the CAPK as specified by the RID/Index

Parameters

capk	6 byte CAPK = 5 bytes RID + 1 byte INDEX
capkLen	the length of capk data buffer

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.8.4.47 int emv_removeCRL (IN BYTE * *list*, IN int *lsLen*)

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
lsLen	the length of list data buffer

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.8.4.48 int emv_removeTerminalData ( )

Remove Terminal Data

Removes the Terminal Data

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.8.4.49 int emv_retrieveAIDList (OUT BYTE * *AIDList*, IN_OUT int * *AIDListLen*)

Retrieve AID list

Returns all the AID names installed on the terminal.

Parameters

AIDList	array of AID name byte arrays
AIDListLen	the length of AIDList array buffer

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.8.4.50 int emv_retrieveApplicationData (IN BYTE * *AID*, IN int *AIDLen*, OUT BYTE * *tlv*, IN_OUT int * *tlvLen*)

Retrieve Application Data by AID

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

Parameters

AID	Name of ApplicationID. Must be between 5 and 16 bytes
AIDLen	the length of AID data buffer.
tlv	The TLV elements of the requested AID
tlvLen	the length of tlv data buffer.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.8.4.51 int emv_retrieveCAPK (IN BYTE * *capk*, IN int *capkLen*, OUT BYTE * *key*, IN_OUT int * *keyLen*)

Retrieve Certificate Authority Public Key

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

Parameters

capk	6 bytes CAPK = 5 bytes RID + 1 byte Index
capkLen	the length of capk data buffer
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.
keyLen	the length of key data buffer •

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.8.4.52 int emv_retrieveCAPKList (OUT BYTE * *keys*, IN_OUT int * *keysLen*)

Retrieve the Certificate Authority Public Key list

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

Parameters

keys	[key1][key2]...[keyn], each key 6 bytes where key = 5 bytes RID + 1 byte index
keysLen	the length of keys data buffer

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.8.4.53 int emv_retrieveCRL (OUT BYTE * *list*, IN_OUT int * *lssLen*)

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
lssLen	the length of list data buffer

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.8.4.54 int emv_retrieveTerminalData (OUT BYTE * *tlv*, IN_OUT int * *tlvLen*)

Retrieve Terminal Data

Retrieves the Terminal Data.

Parameters

tlv	Response returned as a TLV
tlvLen	the length of tlv data buffer

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.8.4.55 int emv_retrieveTerminalID (OUT char * *terminalID*)

DEPRECATED : please use emv_retrieveTerminalID_Len(OUT char* terminalID, IN_OUT int *terminalIDLen)

Gets the terminal ID as printable characters .

Parameters

terminalID	Terminal ID string; needs to have at least 30 bytes of memory

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.8.4.56 int emv_retrieveTerminalID_Len (OUT char * *terminalID*, IN_OUT int * *terminalIDLen*)

Gets the terminal ID as printable characters .

Parameters

terminalID	Terminal ID string
terminalIDLen	Length of terminalID

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.8.4.57 int emv_retrieveTransactionResult (IN BYTE * *tags*, IN int *tagsLen*, IDTTransactionData * *cardData*)

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
tagsLen	Length of tag list
cardData	All requested tags returned as unencrypted, encrypted and masked TLV data in IDTransactionData object

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.8.4.58 int emv_setApplicationData (IN BYTE * *name*, IN int *nameLen*, IN BYTE * *tlv*, IN int *tlvLen*)

Set Application Data by AID

Sets the Application Data as specified by the application name and TLV data

Parameters

name	Application name, 10-32 ASCII hex characters representing 5-16 bytes Example "a0000000031010"
nameLen	the length of name data buffer of Application name,
tlv	Application data in TLV format
tlvLen	the length of tlv data buffer

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.8.4.59 void emv_setAutoAuthenticateTransaction (IN int *authenticate*)

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

Parameters

authenticate	TRUE = auto authenticate, FALSE = manually authenticate

12.8.4.60 void emv_setAutoCompleteTransaction (IN int *complete*)

Enables complete for EMV transactions. If a emv_authenticateTranaction results in code 0x0004 (go online), then emv_completeTransaction can automatically execute if parameter is set to TRUE

Parameters

complete	TRUE = auto complete, FALSE = manually complete

12.8.4.61 int emv_setCAPK (IN BYTE * *capk*, IN int *capkLen*)

Set Certificate Authority Public Key

Sets the CAPK as specified by the CAKey structure

Parameters

capk	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.
capkLen	the length of capk data buffer

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.8.4.62 int emv_setCRL (IN BYTE * *list*, IN int *lsLen*)

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]
lsLen	the length of list data buffer

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.8.4.63 int emv_setTerminalData (IN BYTE * *tlv*, IN int *tlvLen*)

Set Terminal Data

Sets the Terminal Data as specified by the TerminalData structure passed as a parameter

Parameters

tlv	TerminalData configuration file
tlvLen	the length of tlv data buffer

Return values

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

12.8.4.64 int emv_setTerminalID (IN char * *terminalID*)

Sets the terminal ID as printable characters .

Parameters

terminalID	Terminal ID to set

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.8.4.65 int emv_startTransaction (IN double *amount,* IN double *amtOther,* IN int *exponent,* IN int *type,* IN int *timeout,* IN BYTE * *tags,* IN int *tagsLen,* IN int *forceOnline*)

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) - SEE IMPORTANT NOTE BELOW
amtOther	Other amount value, if any (tag value 9F03) - SEE IMPORTANT NOTE BELOW
exponent	Number of characters after decimal point
type	Transaction type (tag value 9C).
timeout	Timeout value in seconds.
tags	Any other tags to be included in the request. Passed as a TLV stream. Example, tag 9F0C with amount 0x000000000100 would be 0x9F0C06000000000100 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.
tagsLen	Length of tags
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 = DFEE1A079F029F36959F37

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString >>>>>IMPORTANT: parameters for amount and amtOther MUST BE PASSED AS A DOUBLE VALUE WITH DECIMAL POINT. Example, do not pass 1, but instead pass 1.0 or 1.00. Otherwise, results will be unpredictable

12.8.4.66 int icc_getICCReaderStatus (OUT 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

12.8.4.67 int icc_powerOffICC ( )

Power Off ICC

Powers down the ICC

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

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

12.8.4.68 int icc_powerOnICC (OUT BYTE * *ATR,* IN_OUT int * *inLen*)

Power On ICC

Power up the currently selected microprocessor card in the ICC reader

Parameters

ATR,the	ATR data response when succeeded power on ICC,
inLen,the	length of ATR data

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.8.4.69 int msr_cancelMSRSwipe ( )

Disable MSR Swipe Cancels MSR swipe request.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.8.4.70 int msr_clearMSRData ( )

Clear MSR Data Clears the MSR Data buffer

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.8.4.71 int msr_getMSRData (OUT BYTE * *reData,* IN_OUT int * *reLen*)

Get MSR Data Reads the MSR Data buffer

Parameters

reData	Card data
reLen	Card data length

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.8.4.72 void msr_registerCallBk (pMSR_callBack *pMSRf*)

To register the msr callback function to get the MSR card data. (Pass NULL to disable the callback.)

12.8.4.73 void msr_registerCallBkp (pMSR_callBackp *pMSRf*)

To register the msr callback function to get the MSR card data pointer. (Pass NULL to disable the callback.)

12.8.4.74 int msr_startMSRSwipe (IN int *_timeout*)

Start MSR Swipe Enables MSR, waiting for swipe to occur. Allows track selection. Returns IDTMSRData instance to deviceDelegate::swipeMSRData:()

Parameters

timeout	Swipe Timeout Value

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.8.4.75 void parseMSRData (IN BYTE * *resData,* IN int *resLen,* IN_OUT IDTMSRData * *cardData*)

Parser the MSR data from the buffer into IDTMSTData structure

Parameters

resData	MSR card data buffer
resLen	the length of resData
cardData	the parser result with IDTMSTData structure

12.8.4.76 void parsePINBlockData (IN BYTE * *resData,* IN int *resLen,* IN_OUT IDTPINData * *cardData*)

Parse the PIN block data from the buffer into IDTPINData structure

Parameters

resData	PIN card data buffer
resLen	the length of resData
cardData	the parser result with IDTPINData structure

12.8.4.77 void parsePINData (IN BYTE * *resData,* IN int *resLen,* IN_OUT IDTPINData * *cardData*)

Parse the PIN data from the buffer into IDTPINData structure

Parameters

resData	PIN card data buffer
resLen	the length of resData
cardData	the parser result with IDTPINData structure

12.8.4.78 int pin_cancelPINEntry ( )

Cancel PIN Entry

Cancels PIN entry request

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.8.4.79 int pin_getPIN (IN int *mode*, IN int *PANSource*, IN char * *iccPAN*, IN int IN *iccPANLen*, int *startTimeout*, IN int *entryTimeout*, IN char * *language*, IN int *languageLen*)

Get Encrypted PIN

Requests PIN Entry

Parameters

mode	• 0x00- Cancel: Cancels PIN entry = also can execute pin_cancelPINEntry(). All other parameters for this method will be ignored • 0x01- Online PIN DUKPT • 0x02- Online PIN MKSK • 0x03- Offline PIN (No need to define PAN Source or ICC PAN)
PANSource	• 0x00- ICC: PAN Captured from ICC and must be provided in iccPAN parameter • 0x01- MSR: PAN Captured from MSR swipe and will be inserted by Spectrum Pro. No need to provide iccPAN parameter.
iccPAN	PAN captured from ICC. When PAN captured from MSR, this parameter will be ignored
iccPANLen	the length of iccPAN
startTimeout	The amount of time allowed to start PIN entry before timeout
entryTimeout	The amount of time to enter the PIN after first digit selected before timeout
language	Valid values "EN" for English, "ES" for Spanish, "ZH" for Chinese, "FR" for French
languageLen	the length of language

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.8.4.80 void pin_registerCallBk (pPIN_callBack *pPINf*)

To register the pin callback function to get the PINPad data. (Pass NULL to disable the callback.)

12.8.4.81 void registerHotplugCallBk (pMessageHotplug *pMsgHotplug*)

To register the USB HID hot-plug callback function which implemented in the application to monitor the hotplug message from the SDK.

12.8.4.82 void registerLogCallBk (pSendDataLog *pFSend,* pReadDataLog *pFRead*)

To register the log callback function which implemented in the application to monitor sending/reading data between application and reader.

12.8.4.83 int rs232_device_init (int *deviceType,* int *port_number,* int *brate*)

Initial the device by RS232

It will try to connect to the device with provided deviceType, port_number, and brate.

Parameters

deviceType	Device to connect to
port_number	Port number of the device

Port nr. | **Linux** | **Windows**

| 0 | ttyS0 | COM1 | | 1 | ttyS1 | COM2 | | 2 | ttyS2 | COM3 | | 3 | ttyS3 | COM4 | | 4 | ttyS4 | COM5 | | 5 | ttyS5 | COM6 | | 6 | ttyS6 | COM7 | | 7 | ttyS7 | COM8 | | 8 | ttyS8 | COM9 | | 9 | ttyS9 | COM10 | | 10 | ttyS10 | COM11 | | 11 | ttyS11 | COM12 | | 12 | ttyS12 | COM13 | | 13 | ttyS13 | COM14 | | 14 | ttyS14 | COM15 | | 15 | ttyS15 | COM16 | | 16 | ttyUSB0 | n.a. | | 17 | ttyUSB1 | n.a. | | 18 | ttyUSB2 | n.a. | | 19 | ttyUSB3 | n.a. | | 20 | ttyUSB4 | n.a. | | 21 | ttyUSB5 | n.a. | | 22 | ttyAMA0 | n.a. | | 23 | ttyAMA1 | n.a. | | 24 | ttyACM0 | n.a. | | 25 | ttyACM1 | n.a. | | 26 | rfcomm0 | n.a. | | 27 | rfcomm1 | n.a. | | 28 | ircomm0 | n.a. | | 29 | ircomm1 | n.a. | | 30 | cuau0 | n.a. | | 31 | cuau1 | n.a. | | 32 | cuau2 | n.a. | | 33 | cuau3 | n.a. | | 34 | cuaU0 | n.a. | | 35 | cuaU1 | n.a. | | 36 | cuaU2 | n.a. | | 37 | cuaU3 | n.a. |

Parameters

brate	Bitrate of the device

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.8.4.84 char* SDK_Version ( )

To Get SDK version

Returns

return the SDK version string

12.8.4.85 int setAbsoluteLibraryPath (const char * *absoluteLibraryPath*)

Set the path to use when searching for ID TECH's libraries. If this is not set, the libraries will be searched for with the system's default procedures.

Parameters

absoluteLibrary-Path	The absolute path to ID TECH's libraries.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.9 Source_C/libIDT_SREDKey2.h File Reference

SREDKey2 API.

```
#include "IDTDef.h"
```

Macros

- #define IN
- #define OUT
- #define IN_OUT

Typedefs

- typedef void(∗ pMessageHotplug)(int, int)
- typedef void(∗ pSendDataLog)(unsigned char ∗, int)
- typedef void(∗ pReadDataLog)(unsigned char ∗, int)
- typedef void(∗ pEMV_callBack)(int, int, unsigned char ∗, int, IDTTransactionData ∗, EMV_Callback ∗, int)
- typedef void(∗ pFW_callBack)(int, int, int, int, int)
- typedef void(∗ pMSR_callBack)(int, IDTMSRData)
- typedef void(∗ pMSR_callBackp)(int, IDTMSRData ∗)
- typedef void(∗ pPIN_callBack)(int, IDTPINData ∗)
- typedef void(∗ pCMR_callBack)(int, IDTCMRData ∗)
- typedef void(∗ pCSFS_callBack)(BYTE status)
- typedef void(∗ pLCD_callBack)(int, IDTLCDItem ∗)
- typedef void(∗ ftpComm_callBack)(int, int, int)
- typedef void(∗ httpComm_callBack)(BYTE ∗, int)
- typedef void(∗ v4Comm_callBack)(BYTE, BYTE, BYTE ∗, int)

Functions

- void registerHotplugCallBk (pMessageHotplug pMsgHotplug)
- void registerLogCallBk (pSendDataLog pFSend, pReadDataLog pFRead)
- void device_registerFWCallBk (pFW_callBack pFWf)
- void device_registerCameraCallBk (pCMR_callBack pCMRf)
- void device_registerCardStatusFrontSwitchCallBk (pCSFS_callBack pCSFSf)
- void emv_registerCallBk (pEMV_callBack pEMVf)
- void msr_registerCallBk (pMSR_callBack pMSRf)
- void msr_registerCallBkp (pMSR_callBackp pMSRf)
- void ctls_registerCallBk (pMSR_callBack pCTLSf)
- void ctls_registerCallBkp (pMSR_callBackp pCTLSf)
- void pin_registerCallBk (pPIN_callBack pPINf)
- void lcd_registerCallBk (pLCD_callBack pLCDf)

- void comm_registerHTTPCallback (httpComm_callBack cBack)
- void comm_registerV4Callback (v4Comm_callBack cBack)
- char * SDK_Version ()
- int setAbsoluteLibraryPath (const char *absoluteLibraryPath)
- int device_setConfigPath (const char *path)
- int device_setNEO2DevicesConfigs (IN const char *configs, IN int len)
- int device_init ()
- int rs232_device_init (int deviceType, int port_number, int brate)
- int device_setCurrentDevice (int deviceType)
- int device_isAttached (int deviceType)
- int device_close ()
- void device_getIDGStatusCodeString (IN int returnCode, OUT char *despcrition)
- int device_isConnected ()
- int device_getFirmwareVersion (OUT char *firmwareVersion)
- int device_getFirmwareVersion_Len (OUT char *firmwareVersion, IN_OUT int *firmwareVersionLen)
- int device_pingDevice ()
- int device_getCurrentDeviceType ()
- int device_SendDataCommandNEO (IN int cmd, IN int subCmd, IN BYTE *data, IN int dataLen, OUT BYTE *response, IN_OUT int *respLen)
- int device_SendDataCommand (IN BYTE *cmd, IN int cmdLen, IN BYTE *data, IN int dataLen, OUT BYTE *response, IN_OUT int *respLen)
- int device_rebootDevice ()
- int device_SendDataCommandITP (IN BYTE *cmd, IN int cmdLen, IN BYTE *data, IN int dataLen, OUT BYTE *response, IN_OUT int *respLen)
- void device_setTransactionExponent (int exponent)
- int device_getKeyStatus (int *newFormat, BYTE *status, int *statusLen)
- int device_updateFirmware (IN BYTE *firmwareData, IN int firmwareDataLen, IN char *firmwareName, IN int encryptionType, IN BYTE *keyBlob, IN int keyBlobLen)
- int config_getModelNumber (OUT char *sNumber)
- int config_getModelNumber_Len (OUT char *sNumber, IN_OUT int *sNumberLen)
- int config_getSerialNumber (OUT char *sNumber)
- int config_getSerialNumber_Len (OUT char *sNumber, IN_OUT int *sNumberLen)
- int device_setSystemLanguage (char *language)
- int msr_setExpirationMask (int mask)
- int msr_getExpirationMask (BYTE *value)
- int msr_setClearPANID (BYTE val)
- int msr_getClearPANID (BYTE *value)
- int msr_setSwipeForcedEncryptionOption (int track1, int track2, int track3, int track3card0)
- int msr_getSwipeForcedEncryptionOption (BYTE *option)
- int msr_setSwipeMaskOption (int track1, int track2, int track3)
- int msr_getSwipeMaskOption (BYTE *option)
- int msr_getFunctionStatus (int *enable, int *isBufferMode, int *withNotification)
- int msr_disable ()

12.9.1 Detailed Description

SREDKey2 API. SREDKey2 Global API methods.

12.9.2 Macro Definition Documentation

12.9.2.1 #define IN

INPUT parameter.

12.9.2.2 #define IN_OUT

INPUT / OUTPUT PARAMETER.

12.9.2.3 #define OUT

OUTPUT parameter.

12.9.3 Typedef Documentation

12.9.3.1 typedef void(* ftpComm_callBack)(int, int, int)

Define the comm callback function to get FTP file transfer status

It should be passed as a parameter in a FTP request, Signature (int, int, int) = response code, current block, total blocks RESPONSE CODES: 100 = FILE DOWNLOAD STARTED 101 = FILE BLOCK XX OF XX RECEIVED 102 = FILE DOWNLOAD COMPLETED 103 = FILE DOWNLOAD TERMINATED PREMATURELY

12.9.3.2 typedef void(* httpComm_callBack)(BYTE *, int)

Define the comm callback function to get the async url data

It should be registered using the comm_registerHTTPCallback

12.9.3.3 typedef void(* pCMR_callBack)(int, IDTCMRData *)

Define the camera callback function to get the image data

It should be registered using the device_registerCameraCallBk,

12.9.3.4 typedef void(* pCSFS_callBack)(BYTE status)

Define the card status and front switch callback function to get card and front switch status

It should be registered using the device_registerCardStatusFrontSwitchCallBk,

12.9.3.5 typedef void(* pEMV_callBack)(int, int, unsigned char *, int, IDTTransactionData *, EMV_Callback *, int)

Define the EMV callback function to get the transaction message/data/result.

It should be registered using the emv_registerCallBk

12.9.3.6 typedef void(* pFW_callBack)(int, int, int, int, int)

Define the firmware update callback function to get the firmware update status

It should be registered using the device_registerFWCallBk

12.9.3.7 typedef void(* pLCD_callBack)(int, IDTLCDItem *)

Define the LCD callback function to get the input LCDItem

It should be registered using the lcd_registerCallBk,

12.9.3.8 typedef void(* pMessageHotplug)(int, int)

Define the USB hot-plug callback function to monitor the info when plug in/out the reader.

It should be registered using the registerHotplugCallBk, The first integer parameter is device type, and the second integer parameter is either 0: Device Plugged Out or 1: Device Plugged In

12.9.3.9 typedef void(* pMSR_callBack)(int, IDTMSRData)

Define the MSR callback function to get the MSR card data

It should be registered using the msr_registerCallBk, this callback function is for backward compatibility

12.9.3.10 typedef void(* pMSR_callBackp)(int, IDTMSRData *)

Define the MSR callback function to get pointer to the MSR card data

It should be registered using the msr_registerCallBk, this callback function is recommended instead of pMSR_callBack

12.9.3.11 typedef void(* pPIN_callBack)(int, IDTPINData *)

Define the PINPad callback function to get the input PIN Pad data

It should be registered using the pin_registerCallBk,

12.9.3.12 typedef void(* pReadDataLog)(unsigned char *, int)

Define the read response callback function to monitor the reading response from the reader.

It should be registered using the registerLogCallBk

12.9.3.13 typedef void(* pSendDataLog)(unsigned char *, int)

Define the send command callback function to monitor the sending command into the reader.

It should be registered using the registerLogCallBk

12.9.3.14 typedef void(* v4Comm_callBack)(BYTE, BYTE, BYTE *, int)

Define the comm callback function to receive the V4 Protocol packets received by the device from an external source (IP/USB/RS-232) It should be registered using the comm_registerV4Callback, Data callback will contain command, sub-command, and data from V4 packet

12.9.4 Function Documentation

12.9.4.1 void comm_registerHTTPCallback (httpComm_callBack *cBack*)

Register Comm HTTP Async Callback

Parameters

cBack	- HTTP Comm callback

12.9.4.2 void comm_registerV4Callback (v4Comm_callBack *cBack*)

Register External V4 Protocol commands Callback

Parameters

cBack	- V4 Protocol Comm callback

12.9.4.3 int config_getModelNumber (OUT char * *sNumber*)

DEPRECATED : please use config_getModelNumber_Len(OUT char* sNumber, IN_OUT int *sNumberLen)

Polls device for Model Number

Parameters

sNumber	Returns Model Number; needs to have at least 64 bytes of memory

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.9.4.4 int config_getModelNumber_Len (OUT char * *sNumber,* IN_OUT int * *sNumberLen*)

Polls device for Model Number

Parameters

sNumber	Returns Model Number
sNumber	length of Model Number

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.9.4.5 int config_getSerialNumber (OUT char * *sNumber*)

DEPRECATED : please use config_getSerialNumber_Len(OUT char* sNumber, IN_OUT int *sNumberLen)

Polls device for Serial Number

Parameters

sNumber	Returns Serial Number; needs to have at least 64 bytes of memory

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.9.4.6 int config_getSerialNumber_Len (OUT char * *sNumber,* IN_OUT int * *sNumberLen*)

Polls device for Serial Number

Parameters

sNumber	Returns Serial Number
sNumberLen	Length of Serial Number

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.9.4.7 void ctls_registerCallBk (pMSR_callBack *pCTLSf*)

To register the ctls callback function to get the MSR card data. (Pass NULL to disable the callback.)

12.9.4.8 void ctls_registerCallBkp (pMSR_callBackp *pCTLSf*)

To register the ctls callback function to get the MSR card data pointer. (Pass NULL to disable the callback.)

12.9.4.9 int device_close ( )

Close the device

Returns

RETURN_CODE: 0: success, 0x0A: failed

12.9.4.10 int device_getCurrentDeviceType ( )

Get current active device type

Returns

: return the device type defined as DEVICE_TYPE in the IDTDef.h

12.9.4.11 int device_getFirmwareVersion (OUT char * *firmwareVersion*)

DEPRECATED : please use device_getFirmwareVersion_Len(OUT char* firmwareVersion, IN_OUT int *firmware-VersionLen)

Polls device for Firmware Version

Parameters

firmwareVersion	Response returned of Firmware Version; needs to have at least 128 bytes of memory

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.9.4.12 int device_getFirmwareVersion_Len (OUT char * *firmwareVersion,* IN_OUT int * *firmwareVersionLen*)

Polls device for Firmware Version

Parameters

firmwareVersion	Response returned of Firmware Version
firmwareVersion-Len	Length of Firmware Version

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.9.4.13 void device_getIDGStatusCodeString (IN int *returnCode,* OUT char * *despcrition*)

Review the return code description.

Parameters

returnCode	the response result.
description	

Return values

the	string for description of response result

- 0: "no error, beginning task";
- 1: "no response from reader";
- 2: "invalid response data";
  - 01: " Incorrect Header Tag";
  - 02: " Unknown Command";
  - 03: " Unknown Sub-Command";
  - 04: " CRC Error in Frame";
  - 05: " Incorrect Parameter";
  - 06: " Parameter Not Supported";
  - 07: " Mal-formatted Data";
  - 08: " Timeout";
  - 0A: " Failed / NACK";
  - 0B: " Command not Allowed";
  - 0C: " Sub-Command not Allowed";
  - 0D: " Buffer Overflow (Data Length too large for reader buffer)";
  - 0E: " User Interface Event";
  - 10: " Need clear firmware(apply in boot loader only)";
  - 11: " Communication type not supported, VT-1, burst, etc. Need encrypted firmware (apply in boot loader only)";
  - 12: " Secure interface is not functional or is in an intermediate state.";
  - 13: " Data field is not mod 8";
  - 14: " Pad 0x80 not found where expected";
  - 15: " Specified key type is invalid";
  - 16: " Could not retrieve key from the SAM (InitSecureComm)";
  - 17: " Hash code problem";
  - 18: " Could not store the key into the SAM (InstallKey)";

- 19: " Frame is too large";
- 1A: " Unit powered up in authentication state but POS must resend the InitSecureComm command";
- 1B: " The EEPROM may not be initialized because SecCommInterface does not make sense";
- 1C: " Problem encoding APDU Module-Specific Status Codes ";
- 20: " Unsupported Index (ILM) SAM Transceiver error - problem communicating with the SAM (Key Mgr)";
- 21: " Unexpected Sequence Counter in multiple frames for single bitmap (ILM)Length error in data returned from the SAM (Key Mgr) - 22: " Improper bit map (ILM)"; - 23: " Request Online Authorization"; - 24: " ViVOCard3 raw data read successful"; - 25: " Message index not available (ILM) ViVOcomm activate transaction card type (ViVOcomm)"; - 26: " Version Information Mismatch (ILM)"; - 27: " Not sending commands in correct index message index (ILM)"; - 28: " Time out or next expected message not received (ILM)"; - 29: " ILM languages not available for viewing (ILM)"; - 2A: " Other language not supported (ILM)"; - 41: " from 41 to 4F, Module-specific errors for Key Manager";
- 50: " Auto-Switch OK";
- 51: " Auto-Switch failed";
- 70: " Antenna Error 80h Use another card";
- 81: " Insert or swipe card";
- 90: " Data encryption Key does not exist";
- 91: " Data encryption Key KSN exhausted";

12.9.4.14 int device_getKeyStatus (int * *newFormat,* BYTE * *status,* int * *statusLen*)

Get Key Status

Gets the status of loaded keys

Parameters

status	newFormat for Augusta and miniSmartII only 1: new format of key status 0: reserved format for support previous device
status	For L100, Augusta and miniSmartII: When the newFormat is 0, data format as follows. For Augusta and miniSmartII: byte 0: PIN DUKPT Key, Does not support, always 0 byte 1: PIN Master Key, Does not support, always 0 byte 2: PIN Session Key, Does not support, always 0 byte 3: Account/MSR DUKPT Key, 1 Exists, 0 None, 0xFF STOP byte 4: Account/ICC DUKPT Key, 1 Exists, 0 None, 0xFF STOP byte 5: Admin DUKPT Key, 1 Exists, 0 None, 0xFF STOP For L100: byte 0: PIN DUKPT Key byte 1: PIN Master Key byte 2: Standard PIN Session Key byte 3: Desjardins PIN Session Key byte 4: Account/MSR DUKPT Key, 1 Exists, 0 None, 0xFF STOP, Does not support, always 0 byte 5: Account/ICC DUKPT Key, 1 Exists, 0 None, 0xFF STOP, Does not support, always 0 byte 6: Admin DUKPT Key, 1 Exists, 0 None, 0xFF STOP byte 7: Data DUKPT Key, 1 Exists, 0 None, 0xFF STOP byte 8: MAC DUKPT Key, 1 Exists, 0 None, 0xFF STOP

when the newFormat is 1, data format as follows. [Block Length] [KeyStatusBlock1] [KeyStatusBlock2]...[KeyStatusBlockN] Where: [Block Length] is 2 bytes, format is Len_L Len_H, is KeyStatusBlock Number [KeyStatusBlockX> is 4 bytes, format is [Key Index and Key Name] [key slot] [key status]: [Key Index and Key Name] is 1 byte. Please refer to following table 0x14 LCL-KEK to Encrypt Other Keys 0x02 Data encryption Key to Encrypt ICC/MSR 0x05 MAC DUKPT Key for Host-Device - MAC Verification 0x05 MTK DUKPT Key for TTK Self-Test 0x0C RKI-KEK for Remote Key Injection [key slot] is 2 bytes. Range is 0 - 9999 the MTK DUKPT Key slot is 16, the others are all 0 [key status] is 1 byte. 0 - Not Exist 1 - Exist 0xFF - (Stop. Only Valid for DUKPT Key) For NEO2 and SREDKey2: Each unit of three bytes represents one key's parameters (index and slot). Key Name Index (1 byte): 0x14 - LCL-KEK 0x01 - Pin encryption Key (NEO2 only) 0x02 - Data encryption Key 0x05 - MAC DUKPT Key 0x0A - PCI Pairing Key (NEO2 only) Key Slot (2 bytes): Indicate different slots of a certain Key Name Example: slot =5 (0x00 0x05), slot=300 (0x01 0x2C) For BTPay380, slot is always 0 For example, 0x14 0x00 0x00 0x02 0x00 0x00 0x0A 0x00 0x00 will represent [KeyNameIndex=0x14,KeySlot=0x0000], [KeyNameIndex=0x02,KeySlot=0x0000] and [KeyNameIndex=0x0A,KeySlot=0x0000]

Parameters

statusLen	the length of status

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.9.4.15 int device_init ( )

Initial the device by USB

It will detect the device and trying connect.

The connect status can be checked by device_isConnected().

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.9.4.16 int device_isAttached (int *deviceType*)

Check if the device is attached to the USB port The function device_init() must be called before this function.

Parameters

deviceType,the	device type of the USB device

Returns

1 if the device is attached, or 0 if the device is not attached

12.9.4.17 int device_isConnected ( )

Check the device conntected status

Returns

DEVICE_DISCONNECT=0, or DEVICE_CONNECTED = 1

12.9.4.18 int device_pingDevice ( )

Ping Device

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

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.9.4.19 int device_rebootDevice ( )

Reboot Device - NGA 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()

12.9.4.20 void device_registerCameraCallBk (pCMR_callBack *pCMRf*)

To register the camera callback function to get the image data. (Pass NULL to disable the callback.)

12.9.4.21 void device_registerCardStatusFrontSwitchCallBk (pCSFS_callBack *pCSFSf*)

To register the card status and front switch callback function to get status. (Pass NULL to disable the callback.)

12.9.4.22 void device_registerFWCallBk (pFW_callBack *pFWf*)

To register the firmware update callback function to get the firmware update processing response. (Pass NULL to disable the callback.)

12.9.4.23 int device_SendDataCommand (IN BYTE * *cmd*, IN int *cmdLen*, IN BYTE * *data*, IN int *dataLen*, OUT BYTE * *response*, IN_OUT int * *respLen*)

Send a Command to NGA device

Sends a command to the device .

Parameters

cmd	buffer of command to execute.
cmdLen,the	length of the buffer cmd.
data	buffer of IDG command data.
dataLen,the	length of the buffer data.
response	Response data
respLen,the	length of Response data

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.9.4.24 int device_SendDataCommandITP (IN BYTE * *cmd*, IN int *cmdLen*, IN BYTE * *data*, IN int *dataLen*, OUT BYTE * *response*, IN_OUT int * *respLen*)

Send a Command to ITP device

Sends a command to the device .

Parameters

cmd	buffer of command to execute.
cmdLen,the	length of the buffer cmd.
data	buffer of ITP command data.
dataLen,the	length of the buffer data.
response	Response data
respLen,the	length of Response data

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.9.4.25 int device_SendDataCommandNEO (IN int *cmd*, IN int *subCmd*, IN BYTE * *data*, IN int *dataLen*, OUT BYTE * *response*, IN_OUT int * *respLen*)

Send a Command to NEO device

Sends a command to the NEO device .

Parameters

cmd	command to execute.
subCmd,sub	command to execute.
data	buffer of NEO command data.
dataLen,the	length of the buffer data.
response	Response data
respLen,the	length of Response data

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.9.4.26 int device_setConfigPath (const char * *path*)

Set the path to the config xml file(s) if any

Parameters

path	The path to the config xml files (such as "NEO2_Devices.xml" which contains the information of NEO2 devices). Only need to specify the path to the folder which contains the config files. File names are not needed. The maximum length of path is 200 characters including the '\0' at the end.

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.9.4.27 int device_setCurrentDevice (int *deviceType*)

Sets the current device to talk to

The connect status can be checked by device_isConnected().

Parameters

deviceType	Device to connect to
	
	`enum DEVICE_TYPE`
	`{`
	`IDT_DEVICE_UNKNOWN=0,`
	`IDT_DEVICE_AUGUSTA_HID,`
	`IDT_DEVICE_AUGUSTA_KB,`
	`IDT_DEVICE_AUGUSTA_S_HID,`
	`IDT_DEVICE_AUGUSTA_S_KB,`
	`IDT_DEVICE_AUGUSTA_S_TTK_HID,`
	`IDT_DEVICE_SPECTRUM_PRO,`
	`IDT_DEVICE_MINISMART_II,`
	`IDT_DEVICE_L100,`
	`IDT_DEVICE_UNIPAY,`
	`IDT_DEVICE_UNIPAY_I_V,`
	`IDT_DEVICE_VP3300_AJ,`
	`IDT_DEVICE_KIOSK_III,`
	`IDT_DEVICE_KIOSK_III_S,`
	`IDT_DEVICE_PIP_READER,`
	`IDT_DEVICE_VENDI,`
	`IDT_DEVICE_VP3300_USB,`
	`IDT_DEVICE_UNIPAY_I_V_TTK,`
	`IDT_DEVICE_VP3300_BT,`
	`IDT_DEVICE_VP8800,`
	`IDT_DEVICE_SREDKEY2_HID,`
	`IDT_DEVICE_SREDKEY2_KB,`
	`IDT_DEVICE_NEO2,`
	`IDT_DEVICE_MINISMART_II_COM = IDT_DEVICE_NEO2+5,`
	`IDT_DEVICE_SPECTRUM_PRO_COM,`
	`IDT_DEVICE_KIOSK_III_COM,`
	`IDT_DEVICE_KIOSK_III_S_COM,`
	`IDT_DEVICE_VP3300_COM,`
	`IDT_DEVICE_NEO2_COM,`
	`IDT_DEVICE_MAX_DEVICES = IDT_DEVICE_NEO2_COM+5`
	`};`

Returns

RETURN_CODE: 1: success, 0: failed

12.9.4.28 int device_setNEO2DevicesConfigs (IN const char * *configs*, IN int *len*)

Pass the content of the config xml file ("NEO2_Devices.xml") as a string to the SDK instead of reading the config xml file by the SDK It needs to be called before device_init(), otherwise the SDK will try to read the config xml file.

Parameters

configs	The content read from the config xml file ("NEO2_Devices.xml" which contains the information of NEO2 devices).
len	The length of the string configs. The maximum length is 5000 bytes.

12.9.4.29 int device_setSystemLanguage (char * *language*)

Set System Language Sets the language for the message displayed in the LCD screen

Parameters

language	2-byte ASCII code, can be "EN" or "JP"

Returns

success or error code. Values can be parsed with device_getIDGStatusCodeString

See Also

ErrorCode

12.9.4.30 void device_setTransactionExponent (int *exponent*)

Sets the transaction exponent to be used with device_startTransaction. Default value is 2

Parameters

exponent,The	exponent to use when calling device_startTransaction

12.9.4.31 int device_updateFirmware (IN BYTE * *firmwareData,* IN int *firmwareDataLen,* IN char * *firmwareName,* IN int *encryptionType,* IN BYTE * *keyBlob,* IN int *keyBlobLen*)

Update Firmware Updates the firmware of NEO 2 devices.

Parameters

firmwareData	Signed binary data of a firmware file provided by IDTech
firmwareDataLen	Length of firmwareData
firmwareName	Firmware name. • For example "VP5300_v1.00.023.0167.S_Test.fm"
encryptionType	Encryption type • 0 : Plaintext • 1 : TDES ECB, PKCS#5 padding • 2 : TDES CBC, PKCS#5, IV is all 0
keyBlob	Encrypted firmware session key blob, TR-31 Rev B, wrapped by FW Key (Optional, none if firmware is plaintext)
keyBlobLen	Length of keyBlob

Returns

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

Firmware update status is returned in the callback with the following values: sender = device type state = DEVICE_FIRMWARE_UPDATE current block total blocks ResultCode:

- RETURN_CODE_DO_SUCCESS = Firmware Update Completed Successfully
- RETURN_CODE_BLOCK_TRANSFER_SUCCESS = Current block transferred successfully
- Any other return code represents an error condition

12.9.4.32 void emv_registerCallBk (pEMV_callBack *pEMVf*)

To register the emv callback function to get the EMV processing response. (Pass NULL to disable the callback.)

12.9.4.33 void lcd_registerCallBk (pLCD_callBack *pLCDf*)

To register the lcd callback function to get the LCDItem. (Pass NULL to disable the callback.)

12.9.4.34 int msr_disable ( )

Disable MSR Disable MSR functions.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.9.4.35 int msr_getClearPANID (BYTE * *value*)

Get Clear PAN ID.

Returns the number of digits that begin the PAN that will be in the clear

Parameters

value	4901 <Setting value>="">. setting Value: Number of digits in clear. Values are char '0' - '6'

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.9.4.36 int msr_getExpirationMask (BYTE * *value*)

Get MSR expiration date mask.

Parameters

value	5001 <Setting value>="">. setting Value: '0' = masked, '1' = not-masked

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.9.4.37 int msr_getFunctionStatus (int * *enable*, int * *isBufferMode*, int * *withNotification*)

Get MSR Function Status.

Gets the MSR function status

Parameters

enable	1 = MSR enabled, 0 = MSR disabled
isBufferMode	1 = buffer mode, 0 = auto mode
withNotification	1 = with notification, 0 = without notification

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.9.4.38 int msr_getSwipeForcedEncryptionOption (BYTE * *option*)

Get MSR Swipe Forced Encryption Option.

Parameters

option	8401 <Setting value>="">. Setting Value Byte using lower four bits as flags. 0 = Force Encryption Off, 1 = Force Encryption On bit0 = Track 1 bit1 = Track 2 bit2 = Track 3 bit4 = Track 3 Card Option 0

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.9.4.39 int msr_getSwipeMaskOption (BYTE * *option*)

Get MSR Swipe Mask Option.

Gets the swipe mask/clear data sending option

Parameters

option	8601 <Setting value>="">. Setting Value Byte using lower three bits as flags. 0 = Mask Option Off, 1 = Mask Option On bit0 = Track 1 bit1 = Track 2 bit2 = Track 3 Example: Response 0x03 = Track1/Track2 Masked Option ON, Track3 Masked Option Off

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.9.4.40 void msr_registerCallBk (pMSR_callBack *pMSRf*)

To register the msr callback function to get the MSR card data. (Pass NULL to disable the callback.)

12.9.4.41 void msr_registerCallBkp (pMSR_callBackp *pMSRf*)

To register the msr callback function to get the MSR card data pointer. (Pass NULL to disable the callback.)

12.9.4.42 int msr_setClearPANID (BYTE *val*)

Set Clear PAN ID.

Parameters

val	Set Clear PAN ID to value: Number of digits to show in clear. Range 0-6.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.9.4.43 int msr_setExpirationMask (int *mask*)

Set Expiration Masking

Sets the flag to mask the expiration date

Parameters

mask	TRUE = mask expiration

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.9.4.44 int msr_setSwipeForcedEncryptionOption (int *track1,* int *track2,* int *track3,* int *track3card0*)

Set MSR Swipe Forced Encryption Option.

Parameters

tarck1	Set track1 encryption to true or false.
tarck2	Set track2 encryption to true or false.
tarck3	Set track3 encryption to true or false.
tarck3card0	Set track3 card0 encryption to true or false.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.9.4.45 int msr_setSwipeMaskOption (int *track1,* int *track2,* int *track3*)

Set MSR Swipe Mask Option.

Sets the swipe mask/clear data sending option

Parameters

tarck1	Set track1 mask to true or false.
tarck2	Set track2 mask to true or false.
tarck3	Set track3 mask to true or false.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.9.4.46 void pin_registerCallBk (pPIN_callBack *pPINf*)

To register the pin callback function to get the PINPad data. (Pass NULL to disable the callback.)

12.9.4.47 void registerHotplugCallBk (pMessageHotplug *pMsgHotplug*)

To register the USB HID hot-plug callback function which implemented in the application to monitor the hotplug message from the SDK.

12.9.4.48 void registerLogCallBk (pSendDataLog *pFSend,* pReadDataLog *pFRead*)

To register the log callback function which implemented in the application to monitor sending/reading data between application and reader.

12.9.4.49 int rs232_device_init (int *deviceType,* int *port_number,* int *brate*)

Initial the device by RS232

It will try to connect to the device with provided deviceType, port_number, and brate.

Parameters

deviceType	Device to connect to
port_number	Port number of the device

Port nr. | **Linux** | **Windows**

| 0 | ttyS0 | COM1 | | 1 | ttyS1 | COM2 | | 2 | ttyS2 | COM3 | | 3 | ttyS3 | COM4 | | 4 | ttyS4 | COM5 | | 5 | ttyS5 | COM6 | | 6 | ttyS6 | COM7 | | 7 | ttyS7 | COM8 | | 8 | ttyS8 | COM9 | | 9 | ttyS9 | COM10 | | 10 | ttyS10 | COM11 | | 11 | ttyS11 | COM12 | | 12 | ttyS12 | COM13 | | 13 | ttyS13 | COM14 | | 14 | ttyS14 | COM15 | | 15 | ttyS15 | COM16 | | 16 | ttyUSB0 | n.a. | | 17 | ttyUSB1 | n.a. | | 18 | ttyUSB2 | n.a. | | 19 | ttyUSB3 | n.a. | | 20 | ttyUSB4 | n.a. | | 21 | ttyUSB5 | n.a. | | 22 | ttyAMA0 | n.a. | | 23 | ttyAMA1 | n.a. | | 24 | ttyACM0 | n.a. | | 25 | ttyACM1 | n.a. | | 26 | rfcomm0 | n.a. | | 27 | rfcomm1 | n.a. | | 28 | ircomm0 | n.a. | | 29 | ircomm1 | n.a. | | 30 | cuau0 | n.a. | | 31 | cuau1 | n.a. | | 32 | cuau2 | n.a. | | 33 | cuau3 | n.a. | | 34 | cuaU0 | n.a. | | 35 | cuaU1 | n.a. | | 36 | cuaU2 | n.a. | | 37 | cuaU3 | n.a. |

Parameters

brate	Bitrate of the device

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.9.4.50 char* SDK_Version ( )

To Get SDK version

Returns

return the SDK version string

12.9.4.51 int setAbsoluteLibraryPath (const char * *absoluteLibraryPath*)

Set the path to use when searching for ID TECH's libraries. If this is not set, the libraries will be searched for with the system's default procedures.

Parameters

absoluteLibrary-Path	The absolute path to ID TECH's libraries.

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.10 Source_C/libIDT_UniPayI_V.h File Reference

UniPay 1.5 API.

```
#include "IDTDef.h"
```

Macros

- #define IN
- #define OUT
- #define IN_OUT

Typedefs

- typedef void(∗ pMessageHotplug)(int, int)
- typedef void(∗ pSendDataLog)(unsigned char ∗, int)
- typedef void(∗ pReadDataLog)(unsigned char ∗, int)
- typedef void(∗ pEMV_callBack)(int, int, unsigned char ∗, int, IDTTransactionData ∗, EMV_Callback ∗, int)
- typedef void(∗ pMSR_callBack)(int, IDTMSRData)
- typedef void(∗ pMSR_callBackp)(int, IDTMSRData ∗)
- typedef void(∗ pPIN_callBack)(int, IDTPINData ∗)
- typedef void(∗ pCMR_callBack)(int, IDTCMRData ∗)
- typedef void(∗ pCSFS_callBack)(BYTE status)
- typedef void(∗ ftpComm_callBack)(int, int, int)
- typedef void(∗ httpComm_callBack)(BYTE ∗, int)
- typedef void(∗ v4Comm_callBack)(BYTE, BYTE, BYTE ∗, int)

Functions

- void registerHotplugCallBk (pMessageHotplug pMsgHotplug)
- void registerLogCallBk (pSendDataLog pFSend, pReadDataLog pFRead)
- void emv_registerCallBk (pEMV_callBack pEMVf)
- void msr_registerCallBk (pMSR_callBack pMSRf)
- void msr_registerCallBkp (pMSR_callBackp pMSRf)
- void pin_registerCallBk (pPIN_callBack pPINf)
- void device_registerCameraCallBk (pCMR_callBack pCMRf)
- void device_registerCardStatusFrontSwitchCallBk (pCSFS_callBack pCSFSf)
- void comm_registerHTTPCallback (httpComm_callBack cBack)
- void comm_registerV4Callback (v4Comm_callBack cBack)
- char ∗ SDK_Version ()
- int setAbsoluteLibraryPath (const char ∗absoluteLibraryPath)
- int device_init ()
- int device_setCurrentDevice (int deviceType)
- int device_close ()
- void device_getIDGStatusCodeString (IN int returnCode, OUT char ∗despcrition)
- int device_isConnected ()
- int device_isAttached (int deviceType)
- int device_getFirmwareVersion (OUT char ∗firmwareVersion)
- int device_getFirmwareVersion_Len (OUT char ∗firmwareVersion, IN_OUT int ∗firmwareVersionLen)
- int device_pingDevice ()
- int device_getCurrentDeviceType ()
- int device_SendDataCommandNEO (IN int cmd, IN int subCmd, IN BYTE ∗data, IN int dataLen, OUT BYTE ∗response, IN_OUT int ∗respLen)
- int device_enablePassThrough (int enablePassThrough)
- int device_setMerchantRecord (int index, int enabled, char ∗merchantID, char ∗merchantURL)
- int device_getMerchantRecord (IN int index, OUT BYTE ∗record)
- int device_getMerchantRecord_Len (IN int index, OUT BYTE ∗record, IN_OUT int ∗recordLen)
- int device_getSDKWaitTime ()
- void device_setSDKWaitTime (int waitTime)
- int device_getThreadStackSize ()

- void device_setThreadStackSize (int threadSize)
- int config_getSerialNumber (OUT char *sNumber)
- int config_getSerialNumber_Len (OUT char *sNumber, IN_OUT int *sNumberLen)
- void emv_allowFallback (IN int allow)
- void emv_setAutoAuthenticateTransaction (IN int authenticate)
- void emv_setAutoCompleteTransaction (IN int complete)
- int emv_getAutoAuthenticateTransaction ()
- int emv_getAutoCompleteTransaction ()
- int emv_startTransaction (IN double amount, IN double amtOther, IN int exponent, IN int type, IN int timeout, IN BYTE *tags, IN int tagsLen, IN int forceOnline)
- int emv_activateTransaction (IN int timeout, IN BYTE *tags, IN int tagsLen, IN int forceOnline)
- int emv_authenticateTransaction (IN BYTE *updatedTLV, IN int updatedTLVLen)
- int emv_authenticateTransactionWithTimeout (IN int timeout, IN BYTE *updatedTLV, IN int updatedTLVLen)
- int emv_completeTransaction (IN int commError, IN BYTE *authCode, IN int authCodeLen, IN BYTE *iad, IN int iadLen, IN BYTE *tlvScripts, IN int tlvScriptsLen, IN BYTE *tlv, IN int tlvLen)
- int emv_cancelTransaction ()
- int emv_retrieveApplicationData (IN BYTE *AID, IN int AIDLen, OUT BYTE *tlv, IN_OUT int *tlvLen)
- int emv_setApplicationData (IN BYTE *name, IN int nameLen, IN BYTE *tlv, IN int tlvLen)
- int emv_setApplicationDataTLV (IN BYTE *tlv, IN int tlvLen)
- int emv_removeApplicationData (IN BYTE *AID, IN int AIDLen)
- int emv_removeAllApplicationData ()
- int emv_retrieveAIDList (OUT BYTE *AIDList, IN_OUT int *AIDListLen)
- int emv_retrieveTerminalData (OUT BYTE *tlv, IN_OUT int *tlvLen)
- int emv_setTerminalData (IN BYTE *tlv, IN int tlvLen)
- int emv_setTerminalMajorConfiguration (IN int configuration)
- int emv_retrieveCAPK (IN BYTE *capk, IN int capkLen, OUT BYTE *key, IN_OUT int *keyLen)
- int emv_setCAPK (IN BYTE *capk, IN int capkLen)
- int emv_removeCAPK (IN BYTE *capk, IN int capkLen)
- int emv_removeAllCAPK ()
- int emv_retrieveCAPKList (OUT BYTE *keys, IN_OUT int *keysLen)
- int emv_retrieveCRL (OUT BYTE *list, IN_OUT int *lssLen)
- int emv_setCRL (IN BYTE *list, IN int lsLen)
- int emv_removeCRL (IN BYTE *list, IN int lsLen)
- int emv_removeAllCRL ()
- int icc_getICCReaderStatus (OUT BYTE *status)
- int icc_powerOnICC (OUT BYTE *ATR, IN_OUT int *inLen)
- int icc_powerOffICC ()
- int icc_exchangeAPDU (IN BYTE *c_APDU, IN int cLen, OUT BYTE *reData, IN_OUT int *reLen)
- int msr_cancelMSRSwipe ()
- int msr_startMSRSwipe (IN int _timeout)
- void parseMSRData (IN BYTE *resData, IN int resLen, IN_OUT IDTMSRData *cardData)

12.10.1 Detailed Description

UniPay 1.5 API. UniPay 1.5 Global API methods.

12.10.2 Macro Definition Documentation

12.10.2.1 #define IN

INPUT parameter.

12.10.2.2 #define IN_OUT

INPUT / OUTPUT PARAMETER.

12.10.2.3 #define OUT

OUTPUT parameter.

12.10.3 Typedef Documentation

12.10.3.1 typedef void(* ftpComm_callBack)(int, int, int)

Define the comm callback function to get FTP file transfer status

It should be passed as a parameter in a FTP request, Signature (int, int, int) = response code, current block, total blocks RESPONSE CODES: 100 = FILE DOWNLOAD STARTED 101 = FILE BLOCK XX OF XX RECEIVED 102 = FILE DOWNLOAD COMPLETED 103 = FILE DOWNLOAD TERMINATED PREMATURELY

12.10.3.2 typedef void(* httpComm_callBack)(BYTE *, int)

Define the comm callback function to get the async url data

It should be registered using the comm_registerHTTPCallback

12.10.3.3 typedef void(* pCMR_callBack)(int, IDTCMRData *)

Define the camera callback function to get the image data

It should be registered using the device_registerCameraCallBk,

12.10.3.4 typedef void(* pCSFS_callBack)(BYTE status)

Define the card status and front switch callback function to get card and front switch status

It should be registered using the device_registerCardStatusFrontSwitchCallBk,

12.10.3.5 typedef void(* pEMV_callBack)(int, int, unsigned char *, int, IDTTransactionData *, EMV_Callback *, int)

Define the EMV callback function to get the transaction message/data/result.

It should be registered using the emv_registerCallBk,

12.10.3.6 typedef void(* pMessageHotplug)(int, int)

Define the USB hot-plug callback function to monitor the info when plug in/out the reader.

It should be registered using the registerHotplugCallBk, The first integer parameter is device type, and the second integer parameter is either 0: Device Plugged Out or 1: Device Plugged In

12.10.3.7 typedef void(* pMSR_callBack)(int, IDTMSRData)

Define the MSR callback function to get the MSR card data

It should be registered using the msr_registerCallBk, this callback function is for backward compatibility

12.10.3.8 typedef void(* pMSR_callBackp)(int, IDTMSRData *)

Define the MSR callback function to get pointer to the MSR card data

It should be registered using the msr_registerCallBk, this callback function is recommended instead of pMSR_callBack

12.10.3.9 typedef void(* pPIN_callBack)(int, IDTPINData *)

Define the PINPad callback function to get the input PIN Pad data

It should be registered using the pin_registerCallBk,

12.10.3.10 typedef void(* pReadDataLog)(unsigned char *, int)

Define the read response callback function to monitor the reading response from the reader.

It should be registered using the registerLogCallBk,

12.10.3.11 typedef void(* pSendDataLog)(unsigned char *, int)

Define the send command callback function to monitor the sending command into the reader.

It should be registered using the registerLogCallBk,

12.10.3.12 typedef void(* v4Comm_callBack)(BYTE, BYTE, BYTE *, int)

Define the comm callback function to receive the V4 Protocol packets received by the device from an external source (IP/USB/RS-232) It should be registered using the comm_registerV4Callback, Data callback will contain command, sub-command, and data from V4 packet

12.10.4 Function Documentation

12.10.4.1 void comm_registerHTTPCallback (httpComm_callBack *cBack*)

Register Comm HTTP Async Callback

Parameters

cBack	- HTTP Comm callback

12.10.4.2 void comm_registerV4Callback (v4Comm_callBack *cBack*)

Register External V4 Protocol commands Callback

Parameters

cBack	- V4 Protocol Comm callback

12.10.4.3 int config_getSerialNumber (OUT char * *sNumber*)

DEPRECATED : please use config_getSerialNumber_Len(OUT char* sNumber, IN_OUT int *sNumberLen)

Polls device for Serial Number

Parameters

sNumber	Returns Serial Number; needs to have at least 64 bytes of memory

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString

12.10.4.4 int config_getSerialNumber_Len (OUT char * *sNumber,* IN_OUT int * *sNumberLen*)

Polls device for Serial Number

Parameters

sNumber	Returns Serial Number
sNumberLen	Length of Serial Number

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString

12.10.4.5 int device_close ( )

Close the device

Returns

RETURN_CODE: 0: success, 0x0A: failed

12.10.4.6 int device_enablePassThrough (int *enablePassThrough*)

Start Remote Key Injection

Starts a remote key injection request with IDTech RKI servers. This function is reserved and not implemented.

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

Enable Pass Through

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

Parameters

enablePass-Through	1 = pass through ON, 0 = pass through OFF

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.10.4.7 int device_getCurrentDeviceType ( )

Get current active device type

Returns

: return the device type defined as DEVICE_TYPE in the IDTDef.h

12.10.4.8 int device_getFirmwareVersion (OUT char * *firmwareVersion*)

DEPRECATED : please use device_getFirmwareVersion_Len(OUT char* firmwareVersion, IN_OUT int *firmwareVersionLen)

Polls device for Firmware Version

Parameters

firmwareVersion	Response returned of Firmware Version; needs to have at least 128 bytes of memory

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.10.4.9 int device_getFirmwareVersion_Len (OUT char * *firmwareVersion,* IN_OUT int * *firmwareVersionLen*)

Polls device for Firmware Version

Parameters

firmwareVersion	Response returned of Firmware Version
firmwareVersion-Len	Length of Firmware Version

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.10.4.10 void device_getIDGStatusCodeString (IN int *returnCode,* OUT char * *despcrition*)

Review the return code description.

Parameters

returnCode	the response result.
description	

Return values

the	string for description of response result

- 0: "no error, beginning task";
- 1: "no response from reader";
- 2: "invalid response data";
  - 01: " Incorrect Header Tag";
  - 02: " Unknown Command";
  - 03: " Unknown Sub-Command";
  - 04: " CRC Error in Frame";
  - 05: " Incorrect Parameter";
  - 06: " Parameter Not Supported";
  - 07: " Mal-formatted Data";
  - 08: " Timeout";
  - 0A: " Failed / NACK";

- 0B: " Command not Allowed";
- 0C: " Sub-Command not Allowed";
- 0D: " Buffer Overflow (Data Length too large for reader buffer)";
- 0E: " User Interface Event";
- 10: " Need clear firmware(apply in boot loader only)";
- 11: " Communication type not supported, VT-1, burst, etc. Need encrypted firmware (apply in boot loader only)";
- 12: " Secure interface is not functional or is in an intermediate state.";
- 13: " Data field is not mod 8";
- 14: " Pad 0x80 not found where expected";
- 15: " Specified key type is invalid";
- 16: " Could not retrieve key from the SAM (InitSecureComm)";
- 17: " Hash code problem";
- 18: " Could not store the key into the SAM (InstallKey)";
- 19: " Frame is too large";
- 1A: " Unit powered up in authentication state but POS must resend the InitSecureComm command";
- 1B: " The EEPROM may not be initialized because SecCommInterface does not make sense";
- 1C: " Problem encoding APDU Module-Specific Status Codes ";
- 20: " Unsupported Index (ILM) SAM Transceiver error - problem communicating with the SAM (Key Mgr)";
- 21: " Unexpected Sequence Counter in multiple frames for single bitmap (ILM)Length error in data returned from the SAM (Key Mgr) - 22: " Improper bit map (ILM)"; - 23: " Request Online Authorization"; - 24: " ViVOCard3 raw data read successful"; - 25: " Message index not available (ILM) ViVOcomm activate transaction card type (ViVOcomm)"; - 26: " Version Information Mismatch (ILM)"; - 27: " Not sending commands in correct index message index (ILM)"; - 28: " Time out or next expected message not received (ILM)"; - 29: " ILM languages not available for viewing (ILM)"; - 2A: " Other language not supported (ILM)"; - 41: " from 41 to 4F, Module-specific errors for Key Manager";
- 50: " Auto-Switch OK";
- 51: " Auto-Switch failed";
- 70: " Antenna Error 80h Use another card";
- 81: " Insert or swipe card";
- 90: " Data encryption Key does not exist";
- 91: " Data encryption Key KSN exhausted";

12.10.4.11 int device_getMerchantRecord (IN int *index,* OUT BYTE * *record*)

DEPRECATED : please use device_getMerchantRecord_Len(IN int index, OUT BYTE * record, IN_OUT int *recordLen)

Get Merchant Record

Gets the merchant record for the device.

Parameters

index	Merchant Record index, valid values 1-6
record;	needs to have at least 99 bytes of memory response data from reader. Merchant Record Index: 1 byte enabled: 1 byte Merchant ID: 32 bytes Length of Merchant URL: 1 byte Merchant URL: 64 bytes

Returns

success or error code. Values can be parsed with device_getIDGStatusCodeString()

See Also

ErrorCode

12.10.4.12 int device_getMerchantRecord_Len (IN int *index*, OUT BYTE * *record*, IN_OUT int * *recordLen*)

Get Merchant Record

Gets the merchant record for the device.

Parameters

index	Merchant Record index, valid values 1-6
record	response data from reader. Merchant Record Index: 1 byte enabled: 1 byte Merchant ID: 32 bytes Length of Merchant URL: 1 byte Merchant URL: 64 bytes
recordLen	Length of record

Returns

success or error code. Values can be parsed with device_getIDGStatusCodeString()

See Also

ErrorCode

12.10.4.13 int device_getSDKWaitTime ( )

Get SDK Wait Time

Get the SDK wait time for transactions

Returns

SDK wait time in seconds

12.10.4.14 int device_getThreadStackSize ( )

Get Thread Stack Size

Get the stack size setting for newly created threads

Returns

Thread Stack Size

12.10.4.15 int device_init ( )

Initial the device by USB

It will detect the device and trying connect.

The connect status can be checked by device_isConnected().

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.10.4.16 int device_isAttached (int *deviceType*)

Check if the device is attached to the USB port The function device_init() must be called before this function.

Parameters

deviceType,the	device type of the USB device

Returns

1 if the device is attached, or 0 if the device is not attached

12.10.4.17 int device_isConnected ( )

Check the device conntected status

Returns

DEVICE_DISCONNECT=0, or DEVICE_CONNECTED = 1

12.10.4.18 int device_pingDevice ( )

Ping Device

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

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.10.4.19 void device_registerCameraCallBk (pCMR_callBack *pCMRf*)

To register the camera callback function to get the image data. (Pass NULL to disable the callback.)

12.10.4.20 void device_registerCardStatusFrontSwitchCallBk (pCSFS_callBack *pCSFSf*)

To register the card status and front switch callback function to get status. (Pass NULL to disable the callback.)

12.10.4.21 int device_SendDataCommandNEO (IN int *cmd,* IN int *subCmd,* IN BYTE * *data,* IN int *dataLen,* OUT BYTE * *response,* IN_OUT int * *respLen*)

Send a Command to device

Sends a command to the device .

Parameters

cmd	buffer of command to execute.
cmdLen,the	length of the buffer cmd.
data	buffer of IDG command data.
dataLen,the	length of the buffer data.
response	Response data
respLen,the	length of Response data

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString() Send a Command to NEO device

Sends a command to the NEO device .

Parameters

cmd	command to execute.
subCmd,sub	command to execute.
data	buffer of NEO command data.
dataLen,the	length of the buffer data.
response	Response data
respLen,the	length of Response data

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.10.4.22 int device_setCurrentDevice (int *deviceType*)

Sets the current device to talk to

The connect status can be checked by device_isConnected().

Parameters

deviceType	Device to connect to
	`enum DEVICE_TYPE`
	`{`
	`IDT_DEVICE_UNKNOWN=0,`
	`IDT_DEVICE_AUGUSTA_HID,`
	`IDT_DEVICE_AUGUSTA_KB,`
	`IDT_DEVICE_AUGUSTA_S_HID,`
	`IDT_DEVICE_AUGUSTA_S_KB,`
	`IDT_DEVICE_AUGUSTA_S_TTK_HID,`
	`IDT_DEVICE_SPECTRUM_PRO,`
	`IDT_DEVICE_MINISMART_II,`
	`IDT_DEVICE_L100,`
	`IDT_DEVICE_UNIPAY,`
	`IDT_DEVICE_UNIPAY_I_V,`
	`IDT_DEVICE_VP3300_AJ,`
	`IDT_DEVICE_KIOSK_III,`
	`IDT_DEVICE_KIOSK_III_S,`
	`IDT_DEVICE_PIP_READER,`
	`IDT_DEVICE_VENDI,`
	`IDT_DEVICE_VP3300_USB,`
	`IDT_DEVICE_UNIPAY_I_V_TTK,`
	`IDT_DEVICE_VP3300_BT,`
	`IDT_DEVICE_VP8800,`
	`IDT_DEVICE_SREDKEY2_HID,`
	`IDT_DEVICE_SREDKEY2_KB,`
	`IDT_DEVICE_NEO2,`
	`IDT_DEVICE_MINISMART_II_COM = IDT_DEVICE_NEO2+5,`
	`IDT_DEVICE_SPECTRUM_PRO_COM,`
	`IDT_DEVICE_KIOSK_III_COM,`
	`IDT_DEVICE_KIOSK_III_S_COM,`
	`IDT_DEVICE_VP3300_COM,`
	`IDT_DEVICE_NEO2_COM,`
	`IDT_DEVICE_MAX_DEVICES = IDT_DEVICE_NEO2_COM+5`
	`};`

Returns

RETURN_CODE: 1: success, 0: failed

12.10.4.23 int device_setMerchantRecord (int *index,* int *enabled,* char * *merchantID,* char * *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 device_getIDGStatusCodeString()

12.10.4.24 void device_setSDKWaitTime (int *waitTime*)

Set SDK Wait Time

Set the SDK wait time for transactions

Parameters

waitTime	The SDK wait time for transaction in seconds

12.10.4.25 void device_setThreadStackSize (int *threadSize*)

Set Thread Stack Size

Set the stack size setting for newly created threads

12.10.4.26 int emv_activateTransaction (IN int *timeout,* IN BYTE * *tags,* IN int *tagsLen,* IN int *forceOnline*)

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	Tags to be included in the request. Passed as a TLV stream. Example, tag 9F0C with amount 0x000000000100 would be 0x9F0C06000000000100
tagsLen	Length of tags
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 = DFEE1A079F029-F36959F37

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString >>>>>IMPORTANT: parameters for amount and amtOther MUST BE PASSED AS A DOUBLE VALUE WITH DECIMAL POINT. Example, do not pass 1, but instead pass 1.0 or 1.00. Otherwise, results will be unpredictable

12.10.4.27 void emv_allowFallback (IN int *allow*)

Allow fallback for EMV transactions. Default is TRUE

Parameters

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

12.10.4.28 int emv_authenticateTransaction (IN BYTE * *updatedTLV,* IN int *updatedTLVLen*)

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.

Parameters

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, 9F37 to be included in response = DFEE1-A079F029F36959F37
updatedTLVLen	

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.10.4.29 int emv_authenticateTransactionWithTimeout (IN int *timeout,* IN BYTE * *updatedTLV,* IN int *updatedTLVLen*)

Authenticate EMV Transaction Request with Timeout

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.

Parameters

timeout	Timeout value in seconds.
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, 9F37 to be included in response = DFEE1-A079F029F36959F37
updatedTLVLen	

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.10.4.30 int emv_cancelTransaction ( )

Cancel EMV Transaction

Cancels the currently executing EMV transaction.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.10.4.31 int emv_completeTransaction (IN int *commError,* IN BYTE * *authCode,* IN int *authCodeLen,* IN BYTE * *iad,* IN int *iadLen,* IN BYTE * *tlvScripts,* IN int *tlvScriptsLen,* IN BYTE * *tlv,* IN int *tlvLen*)

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(1) if host was unreachable, or FALSE(0) 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
authCodeLen	the length of authCode
iad	Issuer Authentication Data, if any. Example 0x11223344556677883030 (tag value 91).
iadLen	the length of iadLen
tlvScripts	71/72 scripts, if any
tlvScriptsLen	the length of tlvScriptsLen
tlv	Additional TLV data to return with transaction results (if any)
tlvLen	the length of tlv

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.10.4.32 int emv_getAutoAuthenticateTransaction ( )

Gets auto authenticate value for EMV transactions.

Returns

RETURN_CODE: TRUE = auto authenticate, FALSE = manually authenticate

12.10.4.33 int emv_getAutoCompleteTransaction ( )

Gets auto complete value for EMV transactions.

Returns

RETURN_CODE: TRUE = auto complete, FALSE = manually complete

12.10.4.34 void emv_registerCallBk (pEMV_callBack *pEMVf*)

To register the emv callback function to get the EMV processing response. (Pass NULL to disable the callback.)

12.10.4.35 int emv_removeAllApplicationData ( )

Remove All Application Data

Removes all the Application Data

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.10.4.36 int emv_removeAllCAPK ( )

Remove All Certificate Authority Public Key

Removes all the CAPK

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.10.4.37 int emv_removeAllCRL ( )

Remove All Certificate Revocation List Entries

Removes all CRLEntry entries

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.10.4.38 int emv_removeApplicationData (IN BYTE * *AID*, IN int *AIDLen*)

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
AIDLen	the length of AID data buffer

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.10.4.39 int emv_removeCAPK (IN BYTE * *capk*, IN int *capkLen*)

Remove Certificate Authority Public Key

Removes the CAPK as specified by the RID/Index

Parameters

capk	6 byte CAPK = 5 bytes RID + 1 byte INDEX
capkLen	the length of capk data buffer

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.10.4.40 int emv_removeCRL (IN BYTE * *list*, IN int *lsLen*)

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
lssLen	the length of list data buffer

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.10.4.41 int emv_retrieveAIDList (OUT BYTE * *AIDList*, IN_OUT int * *AIDListLen*)

Retrieve AID list

Returns all the AID names installed on the terminal for CTLS. .

Parameters

AIDList	array of AID name byte arrays
AIDListLen	the length of AIDList array buffer

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.10.4.42 int emv_retrieveApplicationData (IN BYTE * *AID*, IN int *AIDLen*, OUT BYTE * *tlv*, IN_OUT int * *tlvLen*)

Retrieve Application Data by AID

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

Parameters

AID	Name of ApplicationID. Must be between 5 and 16 bytes
AIDLen	the length of AID data buffer.
tlv	The TLV elements of the requested AID
tlvLen	the length of tlv data buffer.

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.10.4.43 int emv_retrieveCAPK (IN BYTE * *capk*, IN int *capkLen*, OUT BYTE * *key*, IN_OUT int * *keyLen*)

Retrieve Certificate Authority Public Key

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

Parameters

capk	6 bytes CAPK = 5 bytes RID + 1 byte Index
capkLen	the length of capk data buffer
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.
keyLen	the length of key data buffer •

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.10.4.44 int emv_retrieveCAPKList (OUT BYTE * *keys*, IN_OUT int * *keysLen*)

Retrieve the Certificate Authority Public Key list

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

Parameters

keys	[key1][key2]...[keyn], each key 6 bytes where key = 5 bytes RID + 1 byte index
keysLen	the length of keys data buffer

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.10.4.45 int emv_retrieveCRL (OUT BYTE * *list*, IN_OUT int * *lssLen*)

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
lssLen	the length of list data buffer

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.10.4.46 int emv_retrieveTerminalData (OUT BYTE * *tlv,* IN_OUT int * *tlvLen*)

Retrieve Terminal Data

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

Parameters

tlv	Response returned as a TLV
tlvLen	the length of tlv data buffer

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.10.4.47 int emv_setApplicationData (IN BYTE * *name,* IN int *nameLen,* IN BYTE * *tlv,* IN int *tlvLen*)

Set Application Data by AID

Sets the Application Data as specified by the application name and TLV data

Parameters

name	Application name, 10-32 ASCII hex characters representing 5-16 bytes Example "a0000000031010"
nameLen	the length of name data buffer of Application name,
tlv	Application data in TLV format
tlvLen	the length of tlv data buffer

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.10.4.48 int emv_setApplicationDataTLV (IN BYTE * *tlv,* IN int *tlvLen*)

Set Application Data by TLV

Sets the Application Data as specified by the 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) Example valid TLV, for Group #2, AID a0000000035010- : "dfee2d01029f0607a0000000051010ffe10101ffe50110ffe30114ffe20106"
tlvLen	the length of tlv data buffer

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.10.4.49 void emv_setAutoAuthenticateTransaction (IN int *authenticate*)

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

Parameters

authenticate	TRUE = auto authenticate, FALSE = manually authenticate

12.10.4.50 void emv_setAutoCompleteTransaction (IN int *complete*)

Enables complete for EMV transactions. If a emv_authenticateTranaction results in code 0x0004 (go online), then emv_completeTransaction can automatically execute if parameter is set to TRUE

Parameters

complete	TRUE = auto complete, FALSE = manually complete

12.10.4.51 int emv_setCAPK (IN BYTE * *capk,* IN int *capkLen*)

Set Certificate Authority Public Key

Sets the CAPK as specified by the CAKey structure

Parameters

capk	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.

capkLen	the length of capk data buffer

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.10.4.52 int emv_setCRL (IN BYTE * *list*, IN int *lsLen*)

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]
lsLen	the length of list data buffer

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.10.4.53 int emv_setTerminalData (IN BYTE * *tlv*, IN int *tlvLen*)

Set Terminal Data

Sets the Terminal Data for CTLS as specified by the TLV. The first TLV must be Configuration Group Number (Tag FFE4). The terminal global data is group 0, so the first TLV would be FFE40100. Other groups can be defined using this method (1 or greater), and those can be retrieved with emv_getConfigurationGroup(int group), and deleted with emv_removeConfigurationGroup(int group). You cannot delete group 0.

Parameters

tlv	TerminalData configuration file
tlvLen	the length of tlv data buffer

Return values

RETURN_CODE	Values can be parsed with device_getIDGStatusCodeString()

12.10.4.54 int emv_setTerminalMajorConfiguration (IN int *configuration*)

Sets the terminal major configuration in ICS .

Parameters

configuration	A configuration value, range 1-23 • 1 = 1C • 2 = 2C • 3 = 3C • 4 = 4C • 5 = 5C ... • 23 = 23C

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.10.4.55 int emv_startTransaction (IN double *amount,* IN double *amtOther,* IN int *exponent,* IN int *type,* IN int *timeout,* IN BYTE * *tags,* IN int *tagsLen,* IN int *forceOnline*)

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) - SEE IMPORTANT NOTE BELOW
amtOther	Other amount value, if any (tag value 9F03) - SEE IMPORTANT NOTE BELOW
exponent	Number of characters after decimal point
type	Transaction type (tag value 9C).
timeout	Timeout value in seconds.
tags	Any other tags to be included in the request. Passed as a TLV stream. Example, tag 9F0C with amount 0x000000000100 would be 0x9F0C06000000000100 If tags 9F02 (amount),9-F03 (other amount), or 9C (transaction type) are included, they will take priority over these values supplied as individual parameters to this method.
tagsLen	Length of tags
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 = DFEE1A079F029-F36959F37

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString >>>>>IMPORTANT: parameters for amount and amtOther MUST BE PASSED AS A DOUBLE VALUE WITH DECIMAL POINT. Example, do not pass 1, but instead pass 1.0 or 1.00. Otherwise, results will be unpredictable

12.10.4.56 int icc_exchangeAPDU (IN BYTE * *c_APDU,* IN int *cLen,* OUT BYTE * *reData,* IN_OUT int * *reLen*)

Exchange APDU with plain text For Non-SRED Augusta Only

Sends an APDU packet to the ICC. If successful, response is the APDU data in response parameter.

Parameters

c_APDU	APDU data packet
cLen	APDU data packet length
reData	Unencrypted APDU response
reLen	Unencrypted APDU response data length

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString

12.10.4.57 int icc_getICCReaderStatus (OUT 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_getIDGStatusCodeString

12.10.4.58 int icc_powerOffICC ( )

Power Off ICC

Powers down the ICC

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

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

12.10.4.59 int icc_powerOnICC (OUT BYTE * *ATR,* IN_OUT int * *inLen*)

Power On ICC

Power up the currently selected microprocessor card in the ICC reader

Parameters

ATR,the	ATR data response when succeeded power on ICC,
inLen,the	length of ATR data

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.10.4.60 int msr_cancelMSRSwipe ( )

Disable MSR Swipe Cancels MSR swipe request.

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.10.4.61 void msr_registerCallBk (pMSR_callBack *pMSRf*)

To register the msr callback function to get the MSR card data. (Pass NULL to disable the callback.)

12.10.4.62 void msr_registerCallBkp (pMSR_callBackp *pMSRf*)

To register the msr callback function to get the MSR card data pointer. (Pass NULL to disable the callback.)

12.10.4.63 int msr_startMSRSwipe (IN int *_timeout*)

Start MSR Swipe Enables MSR, waiting for swipe to occur. Allows track selection. Returns IDTMSRData instance to deviceDelegate::swipeMSRData:()

Parameters

timeout	Swipe Timeout Value

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString() Note: if auto poll mode is on, it will return command not allowed error

12.10.4.64 void parseMSRData (IN BYTE * *resData*, IN int *resLen*, IN_OUT IDTMSRData * *cardData*)

Parser the MSR data from the buffer into IDTMSTData structure

Parameters

resData	MSR card data buffer
resLen	the length of resData
cardData	the parser result with IDTMSTData structure

12.10.4.65 void pin_registerCallBk (pPIN_callBack *pPINf*)

To register the pin callback function to get the PINPad data. (Pass NULL to disable the callback.)

12.10.4.66 void registerHotplugCallBk (pMessageHotplug *pMsgHotplug*)

To register the USB HID hot-plug callback function which implemented in the application to monitor the hotplug message from the SDK.

12.10.4.67 void registerLogCallBk (pSendDataLog *pFSend*, pReadDataLog *pFRead*)

To register the log callback function which implemented in the application to monitor sending/reading data between application and reader.

12.10.4.68 char* SDK_Version ( )

To Get SDK version

Returns

return the SDK version string

12.10.4.69 int setAbsoluteLibraryPath (const char * *absoluteLibraryPath*)

Set the path to use when searching for ID TECH's libraries. If this is not set, the libraries will be searched for with the system's default procedures.

Parameters

absoluteLibrary-Path	The absolute path to ID TECH's libraries.

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.11 Source_C/libIDT_Vendi.h File Reference

Vendi API.

```
#include "IDTDef.h"
```

Macros

- #define IN
- #define OUT
- #define IN_OUT

Typedefs

- typedef void(∗ pMessageHotplug)(int, int)
- typedef void(∗ pSendDataLog)(unsigned char ∗, int)
- typedef void(∗ pReadDataLog)(unsigned char ∗, int)
- typedef void(∗ pEMV_callBack)(int, int, unsigned char ∗, int, IDTTransactionData ∗, EMV_Callback ∗, int)
- typedef void(∗ pMSR_callBack)(int, IDTMSRData)
- typedef void(∗ pMSR_callBackp)(int, IDTMSRData ∗)
- typedef void(∗ pPIN_callBack)(int, IDTPINData ∗)
- typedef void(∗ pCMR_callBack)(int, IDTCMRData ∗)
- typedef void(∗ pCSFS_callBack)(BYTE status)
- typedef void(∗ ftpComm_callBack)(int, int, int)
- typedef void(∗ httpComm_callBack)(BYTE ∗, int)
- typedef void(∗ v4Comm_callBack)(BYTE, BYTE, BYTE ∗, int)

Functions

- void registerHotplugCallBk (pMessageHotplug pMsgHotplug)
- void registerLogCallBk (pSendDataLog pFSend, pReadDataLog pFRead)
- void emv_registerCallBk (pEMV_callBack pEMVf)
- void msr_registerCallBk (pMSR_callBack pMSRf)
- void msr_registerCallBkp (pMSR_callBackp pMSRf)
- void ctls_registerCallBk (pMSR_callBack pCTLSf)
- void ctls_registerCallBkp (pMSR_callBackp pCTLSf)
- void pin_registerCallBk (pPIN_callBack pPINf)
- void device_registerCameraCallBk (pCMR_callBack pCMRf)
- void device_registerCardStatusFrontSwitchCallBk (pCSFS_callBack pCSFSf)
- void comm_registerHTTPCallback (httpComm_callBack cBack)
- void comm_registerV4Callback (v4Comm_callBack cBack)
- char ∗ SDK_Version ()
- int setAbsoluteLibraryPath (const char ∗absoluteLibraryPath)
- int device_init ()
- int device_setCurrentDevice (int deviceType)
- int device_isAttached (int deviceType)
- int device_close ()
- void device_getIDGStatusCodeString (IN int returnCode, OUT char ∗despcrition)
- int device_isConnected ()
- int device_getFirmwareVersion (OUT char ∗firmwareVersion)
- int device_getFirmwareVersion_Len (OUT char ∗firmwareVersion, IN_OUT int ∗firmwareVersionLen)
- int device_pingDevice ()

- int device_controlUserInterface (IN BYTE *values)
- int device_getCurrentDeviceType ()
- int device_SendDataCommandNEO (IN int cmd, IN int subCmd, IN BYTE *data, IN int dataLen, OUT BYTE *response, IN_OUT int *respLen)
- int device_enablePassThrough (int enablePassThrough)
- int device_setBurstMode (IN BYTE mode)
- int device_setPollMode (IN BYTE mode)
- int device_getSDKWaitTime ()
- void device_setSDKWaitTime (int waitTime)
- int device_getThreadStackSize ()
- void device_setThreadStackSize (int threadSize)
- int device_setMerchantRecord (int index, int enabled, char *merchantID, char *merchantURL)
- int device_getMerchantRecord (IN int index, OUT BYTE *record)
- int device_getMerchantRecord_Len (IN int index, OUT BYTE *record, IN_OUT int *recordLen)
- int device_getTransactionResults (IDTMSRData *cardData)
- int config_getSerialNumber (OUT char *sNumber)
- int config_getSerialNumber_Len (OUT char *sNumber, IN_OUT int *sNumberLen)
- int ctls_startTransaction (IN double amount, IN double amtOther, IN int type, IN const int _timeout, IN BYTE *tags, IN int tagsLen)
- int ctls_activateTransaction (IN const int _timeout, IN BYTE *tags, IN int tagsLen)
- int ctls_cancelTransaction ()
- int ctls_retrieveApplicationData (IN BYTE *AID, IN int AIDLen, OUT BYTE *tlv, IN_OUT int *tlvLen)
- int ctls_setApplicationData (IN BYTE *tlv, IN int tlvLen)
- int ctls_removeApplicationData (IN BYTE *AID, IN int AIDLen)
- int ctls_removeAllApplicationData ()
- int ctls_retrieveAIDList (OUT BYTE *AIDList, IN_OUT int *AIDListLen)
- int ctls_retrieveTerminalData (OUT BYTE *tlv, IN_OUT int *tlvLen)
- int ctls_setTerminalData (IN BYTE *tlv, IN int tlvLen)
- int ctls_retrieveCAPK (IN BYTE *capk, IN int capkLen, OUT BYTE *key, IN_OUT int *keyLen)
- int ctls_setCAPK (IN BYTE *capk, IN int capkLen)
- int ctls_removeCAPK (IN BYTE *capk, IN int capkLen)
- int ctls_removeAllCAPK ()
- int ctls_retrieveCAPKList (OUT BYTE *keys, IN_OUT int *keysLen)
- int ctls_setConfigurationGroup (IN BYTE *tlv, IN int tlvLen)
- int ctls_getConfigurationGroup (IN int group, OUT BYTE *tlv, OUT int *tlvLen)
- int ctls_getAllConfigurationGroups (OUT BYTE *tlv, IN_OUT int *tlvLen)
- int ctls_removeConfigurationGroup (int group)
- int msr_cancelMSRSwipe ()
- int msr_startMSRSwipe (IN int _timeout)
- void parseMSRData (IN BYTE *resData, IN int resLen, IN_OUT IDTMSRData *cardData)

12.11.1 Detailed Description

Vendi API. Vendi Global API methods.

12.11.2 Macro Definition Documentation

12.11.2.1 #define IN

INPUT parameter.

12.11.2.2 #define IN_OUT

INPUT / OUTPUT PARAMETER.

12.11.2.3 #define OUT

OUTPUT parameter.

12.11.3 Typedef Documentation

12.11.3.1 typedef void(∗ ftpComm_callBack)(int, int, int)

Define the comm callback function to get FTP file transfer status

It should be passed as a parameter in a FTP request, Signature (int, int, int) = response code, current block, total blocks RESPONSE CODES: 100 = FILE DOWNLOAD STARTED 101 = FILE BLOCK XX OF XX RECEIVED 102 = FILE DOWNLOAD COMPLETED 103 = FILE DOWNLOAD TERMINATED PREMATURELY

12.11.3.2 typedef void(∗ httpComm_callBack)(BYTE ∗, int)

Define the comm callback function to get the async url data

It should be registered using the comm_registerHTTPCallback

12.11.3.3 typedef void(∗ pCMR_callBack)(int, IDTCMRData ∗)

Define the camera callback function to get the image data

It should be registered using the device_registerCameraCallBk,

12.11.3.4 typedef void(∗ pCSFS_callBack)(BYTE status)

Define the card status and front switch callback function to get card and front switch status

It should be registered using the device_registerCardStatusFrontSwitchCallBk,

12.11.3.5 typedef void(∗ pEMV_callBack)(int, int, unsigned char ∗, int, IDTTransactionData ∗, EMV_Callback ∗, int)

Define the EMV callback function to get the transaction message/data/result.

It should be registered using the emv_registerCallBk,

12.11.3.6 typedef void(∗ pMessageHotplug)(int, int)

Define the USB hot-plug callback function to monitor the info when plug in/out the reader.

It should be registered using the registerHotplugCallBk, The first integer parameter is device type, and the second integer parameter is either 0: Device Plugged Out or 1: Device Plugged In

12.11.3.7 typedef void(∗ pMSR_callBack)(int, IDTMSRData)

Define the MSR callback function to get the MSR card data

It should be registered using the msr_registerCallBk, this callback function is for backward compatibility

12.11.3.8 typedef void(∗ pMSR_callBackp)(int, IDTMSRData ∗)

Define the MSR callback function to get pointer to the MSR card data

It should be registered using the msr_registerCallBk, this callback function is recommended instead of pMSR_callBack

12.11.3.9 typedef void(* pPIN_callBack)(int, IDTPINData *)

Define the PINPad callback function to get the input PIN Pad data

It should be registered using the pin_registerCallBk,

12.11.3.10 typedef void(* pReadDataLog)(unsigned char *, int)

Define the read response callback function to monitor the reading response from the reader.

It should be registered using the registerLogCallBk,

12.11.3.11 typedef void(* pSendDataLog)(unsigned char *, int)

Define the send command callback function to monitor the sending command into the reader.

It should be registered using the registerLogCallBk,

12.11.3.12 typedef void(* v4Comm_callBack)(BYTE, BYTE, BYTE *, int)

Define the comm callback function to receive the V4 Protocol packets received by the device from an external source (IP/USB/RS-232) It should be registered using the comm_registerV4Callback, Data callback will contain command, sub-command, and data from V4 packet

12.11.4 Function Documentation

12.11.4.1 void comm_registerHTTPCallback (httpComm_callBack *cBack*)

Register Comm HTTP Async Callback

Parameters

cBack	HTTP Comm callback

12.11.4.2 void comm_registerV4Callback (v4Comm_callBack *cBack*)

Register External V4 Protocol commands Callback

Parameters

cBack	V4 Protocol Comm callback

12.11.4.3 int config_getSerialNumber (OUT char * *sNumber*)

DEPRECATED : please use config_getSerialNumber_Len(OUT char* sNumber, IN_OUT int *sNumberLen)

Polls device for Serial Number

Parameters

sNumber	Returns Serial Number; needs to have at least 64 bytes of memory

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString

12.11.4.4 int config_getSerialNumber_Len (OUT char * *sNumber,* IN_OUT int * *sNumberLen*)

Polls device for Serial Number

Parameters

sNumber	Returns Serial Number
sNumberLen	Length of Serial Number

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString

12.11.4.5 int ctls_activateTransaction (IN const int *_timeout,* IN BYTE * *tags,* IN int *tagsLen*)

Start CTLS Transaction Request

Authorizes the CTLS 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 stream. Example, tag 9F0C with amount 0x000000000100 would be 0x9F0C06000000000100
tagsLen	The length of tags data buffer.

>>>>>IMPORTANT: parameters for amount and amtOther MUST BE PASSED AS A DOUBLE VALUE WITH DECIMAL POINT. Example, do not pass 1, but instead pass 1.0 or 1.00. Otherwise, results will be unpredictable

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString() Note: if auto poll is on, it will returm the error IDG_P2_STATUS_CODE_COMMAND_NOT_ALLOWED

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

12.11.4.6 int ctls_cancelTransaction ( )

Cancel EMV Transaction

Cancels the currently executing EMV transaction.

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.11.4.7 int ctls_getAllConfigurationGroups (OUT BYTE * *tlv*, IN_OUT int * *tlvLen*)

Retrieve All Configuration Groups

Returns all the Configuration Groups installed on the terminal for CTLS

Parameters

tlv	The TLV elements data
tlvLen	the length of tlv data buffer.

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.11.4.8 int ctls_getConfigurationGroup (IN int *group*, OUT BYTE * *tlv*, OUT int * *tlvLen*)

Get Configuration Group

Retrieves the Configuration for the specified Group.

Parameters

group	Configuration Group
tlv	return data
tlvLen	the length of tlv data buffer

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.11.4.9 void ctls_registerCallBk (pMSR_callBack *pCTLSf*)

To register the ctls callback function to get the MSR card data. (Pass NULL to disable the callback.)

12.11.4.10 void ctls_registerCallBkp (pMSR_callBackp *pCTLSf*)

To register the ctls callback function to get the MSR card data pointer. (Pass NULL to disable the callback.)

12.11.4.11 int ctls_removeAllApplicationData ( )

Remove All Application Data

Removes all the Application Data

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.11.4.12 int ctls_removeAllCAPK ( )

Remove All Certificate Authority Public Key

Removes all the CAPK

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.11.4.13 int ctls_removeApplicationData (IN BYTE * *AID,* IN int *AIDLen*)

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
AIDLen	the length of AID data buffer

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.11.4.14 int ctls_removeCAPK (IN BYTE * *capk,* IN int *capkLen*)

Remove Certificate Authority Public Key

Removes the CAPK as specified by the RID/Index

Parameters

capk	6 byte CAPK = 5 bytes RID + 1 byte INDEX
capkLen	the length of capk data buffer

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.11.4.15 int 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	Values can be parsed with device_getIDGStatusCodeString()

12.11.4.16 int ctls_retrieveAIDList (OUT BYTE * *AIDList*, IN_OUT int * *AIDListLen*)

Retrieve AID list

Returns all the AID names installed on the terminal for CTLS. .

Parameters

AIDList	array of AID name byte arrays
AIDListLen	the length of AIDList array buffer

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.11.4.17 int ctls_retrieveApplicationData (IN BYTE * *AID*, IN int *AIDLen*, OUT BYTE * *tlv*, IN_OUT int * *tlvLen*)

Retrieve Application Data by AID

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

Parameters

AID	Name of ApplicationID. Must be between 5 and 16 bytes
AIDLen	the length of AID data buffer.
tlv	The TLV elements of the requested AID
tlvLen	the length of tlv data buffer.

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.11.4.18 int ctls_retrieveCAPK (IN BYTE * *capk*, IN int *capkLen*, OUT BYTE * *key*, IN_OUT int * *keyLen*)

Retrieve Certificate Authority Public Key

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

Parameters

capk	6 bytes CAPK = 5 bytes RID + 1 byte Index
capkLen	the length of capk data buffer
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.
keyLen	the length of key data buffer •

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.11.4.19 int ctls_retrieveCAPKList (OUT BYTE * *keys*, IN_OUT int * *keysLen*)

Retrieve the Certificate Authority Public Key list

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

Parameters

keys	[key1][key2]...[keyn], each key 6 bytes where key = 5 bytes RID + 1 byte index
keysLen	the length of keys data buffer

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.11.4.20 int ctls_retrieveTerminalData (OUT BYTE * *tlv*, IN_OUT int * *tlvLen*)

Retrieve Terminal Data

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

Parameters

tlv	Response returned as a TLV
tlvLen	the length of tlv data buffer

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.11.4.21 int ctls_setApplicationData (IN BYTE * *tlv*, IN int *tlvLen*)

Set Application Data by AID

Sets the Application Data for CTLS as specified by TLV data

Parameters

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

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

Parameters

tlvLen	the length of tlv data buffer

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.11.4.22 int ctls_setCAPK (IN BYTE * *capk*, IN int *capkLen*)

Set Certificate Authority Public Key

Sets the CAPK as specified by the CAKey structure

Parameters

capk	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.
capkLen	the length of capk data buffer

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.11.4.23 int ctls_setConfigurationGroup (IN BYTE * *tlv,* IN int *tlvLen*)

Set Configuration Group

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

Parameters

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

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.11.4.24 int ctls_setTerminalData (IN BYTE * *tlv,* IN int *tlvLen*)

Set Terminal Data

Sets the Terminal Data for CTLS as specified by the TLV. The first TLV must be Configuration Group Number (Tag FFE4). The terminal global data is group 0, so the first TLV would be FFE40100. Other groups can be defined using this method (1 or greater), and those can be retrieved with ctls_getConfigurationGroup(int group), and deleted with ctls_removeConfigurationGroup(int group). You cannot delete group 0.

Parameters

tlv	TerminalData configuration file
tlvLen	the length of tlv data buffer

Return values

RETURN_CODE	Values can be parsed with device_getIDGStatusCodeString()

12.11.4.25 int ctls_startTransaction (IN double *amount,* IN double *amtOther,* IN int *type,* IN const int *_timeout,* IN BYTE * *tags,* IN int *tagsLen*)

Start CTLS Transaction Request

Authorizes the CTLS transaction for an ICC card

The tags will be returned in the callback routine.

Parameters

amount	Transaction amount value (tag value 9F02) - SEE IMPORTANT NOTE BELOW
amtOther	Other amount value, if any (tag value 9F03) - SEE IMPORTANT NOTE BELOW
type	Transaction type (tag value 9C).
timeout	Timeout value in seconds.
tags	Any other tags to be included in the request. Passed as TLV stream. Example, tag 9F0C with amount 0x000000000100 would be 0x9F0C06000000000100 If tags 9F02 (amount),9-F03 (other amount), or 9C (transaction type) are included, they will take priority over these values supplied as individual parameters to this method.
tagsLen	The length of tags data buffer.

>>>>>IMPORTANT: parameters for amount and amtOther MUST BE PASSED AS A DOUBLE VALUE WITH DECIMAL POINT. Example, do not pass 1, but instead pass 1.0 or 1.00. Otherwise, results will be unpredictable

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString() Note: if auto poll is on, it will returm the error IDG_P2_STATUS_CODE_COMMAND_NOT_ALLOWED

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

12.11.4.26 int device_close ( )

Close the device

Returns

RETURN_CODE: 0: success, 0x0A: failed

12.11.4.27 int device_controlUserInterface (IN BYTE * *values*)

Control User Interface

Controls the User Interface: Display, Beep, LED

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

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.11.4.28 int device_enablePassThrough (int *enablePassThrough*)

Start Remote Key Injection

Starts a remote key injection request with IDTech RKI servers. This function is reserved and not implemented.

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

Enable Pass Through

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

Parameters

enablePass-Through	1 = pass through ON, 0 = pass through OFF

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.11.4.29 int device_getCurrentDeviceType ( )

Get current active device type

Returns

: return the device type defined as DEVICE_TYPE in the IDTDef.h

12.11.4.30 int device_getFirmwareVersion (OUT char * *firmwareVersion*)

DEPRECATED : please use device_getFirmwareVersion_Len(OUT char* firmwareVersion, IN_OUT int *firmwareVersionLen)

Polls device for Firmware Version

Parameters

firmwareVersion	Response returned of Firmware Version; needs to have at least 128 bytes of memory

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.11.4.31 int device_getFirmwareVersion_Len (OUT char * *firmwareVersion,* IN_OUT int * *firmwareVersionLen*)

Polls device for Firmware Version

Parameters

firmwareVersion	Response returned of Firmware Version
firmwareVersion-Len	Length of Firmware Version

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.11.4.32 void device_getIDGStatusCodeString (IN int *returnCode,* OUT char * *despcrition*)

Review the return code description.

Parameters

returnCode	the response result.
description	

Return values

the	string for description of response result

- 0: "no error, beginning task";
- 1: "no response from reader";
- 2: "invalid response data";
  - 01: " Incorrect Header Tag";
  - 02: " Unknown Command";

- 03: " Unknown Sub-Command";
- 04: " CRC Error in Frame";
- 05: " Incorrect Parameter";
- 06: " Parameter Not Supported";
- 07: " Mal-formatted Data";
- 08: " Timeout";
- 0A: " Failed / NACK";
- 0B: " Command not Allowed";
- 0C: " Sub-Command not Allowed";
- 0D: " Buffer Overflow (Data Length too large for reader buffer)";
- 0E: " User Interface Event";
- 10: " Need clear firmware(apply in boot loader only)";
- 11: " Communication type not supported, VT-1, burst, etc. Need encrypted firmware (apply in boot loader only)";
- 12: " Secure interface is not functional or is in an intermediate state.";
- 13: " Data field is not mod 8";
- 14: " Pad 0x80 not found where expected";
- 15: " Specified key type is invalid";
- 16: " Could not retrieve key from the SAM (InitSecureComm)";
- 17: " Hash code problem";
- 18: " Could not store the key into the SAM (InstallKey)";
- 19: " Frame is too large";
- 1A: " Unit powered up in authentication state but POS must resend the InitSecureComm command";
- 1B: " The EEPROM may not be initialized because SecCommInterface does not make sense";
- 1C: " Problem encoding APDU Module-Specific Status Codes ";
- 20: " Unsupported Index (ILM) SAM Transceiver error - problem communicating with the SAM (Key Mgr)";
- 21: " Unexpected Sequence Counter in multiple frames for single bitmap (ILM)Length error in data returned from the SAM (Key Mgr) - 22: " Improper bit map (ILM)"; - 23: " Request Online Authorization"; - 24: " ViVOCard3 raw data read successful"; - 25: " Message index not available (ILM) ViVOcomm activate transaction card type (ViVOcomm)"; - 26: " Version Information Mismatch (ILM)"; - 27: " Not sending commands in correct index message index (ILM)"; - 28: " Time out or next expected message not received (ILM)"; - 29: " ILM languages not available for viewing (ILM)"; - 2A: " Other language not supported (ILM)"; - 41: " from 41 to 4F, Module-specific errors for Key Manager";
- 50: " Auto-Switch OK";
- 51: " Auto-Switch failed";
- 70: " Antenna Error 80h Use another card";
- 81: " Insert or swipe card";
- 90: " Data encryption Key does not exist";
- 91: " Data encryption Key KSN exhausted";

12.11.4.33 int device_getMerchantRecord (IN int *index,* OUT BYTE * *record*)

DEPRECATED : please use device_getMerchantRecord_Len(IN int index, OUT BYTE * record, IN_OUT int *recordLen)

Get Merchant Record

Gets the merchant record for the device.

Parameters

index	Merchant Record index, valid values 1-6
record;	needs to have at least 99 bytes of memory response data from reader. Merchant Record Index: 1 byte enabled: 1 byte Merchant ID: 32 bytes Length of Merchant URL: 1 byte Merchant URL: 64 bytes

Returns

success or error code. Values can be parsed with device_getIDGStatusCodeString()

See Also

ErrorCode

12.11.4.34 int device_getMerchantRecord_Len (IN int *index*, OUT BYTE * *record*, IN_OUT int * *recordLen*)

Get Merchant Record

Gets the merchant record for the device.

Parameters

index	Merchant Record index, valid values 1-6
record	response data from reader. Merchant Record Index: 1 byte enabled: 1 byte Merchant ID: 32 bytes Length of Merchant URL: 1 byte Merchant URL: 64 bytes
recordLen	Length of record

Returns

success or error code. Values can be parsed with device_getIDGStatusCodeString()

See Also

ErrorCode

12.11.4.35 int device_getSDKWaitTime ( )

Get SDK Wait Time

Get the SDK wait time for transactions

Returns

SDK wait time in seconds

12.11.4.36 int device_getThreadStackSize ( )

Get Thread Stack Size

Get the stack size setting for newly created threads

Returns

Thread Stack Size

12.11.4.37 int device_getTransactionResults (IDTMSRData * *cardData*)

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

Parameters

cardData	The transaction results

Returns

success or error code. Values can be parsed with device_getResponseCodeString

See Also

ErrorCode

12.11.4.38 int device_init ( )

Initial the device by USB

It will detect the device and trying connect.

The connect status can be checked by device_isConnected().

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.11.4.39 int device_isAttached (int *deviceType*)

Check if the device is attached to the USB port The function device_init() must be called before this function.

Parameters

deviceType,the	device type of the USB device

Returns

1 if the device is attached, or 0 if the device is not attached

12.11.4.40 int device_isConnected ( )

Check the device conntected status

Returns

DEVICE_DISCONNECT=0, or DEVICE_CONNECTED = 1

12.11.4.41 int device_pingDevice ( )

Ping Device

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

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.11.4.42 void device_registerCameraCallBk (pCMR_callBack *pCMRf*)

To register the camera callback function to get the image data. (Pass NULL to disable the callback.)

12.11.4.43 void device_registerCardStatusFrontSwitchCallBk (pCSFS_callBack *pCSFSf*)

To register the card status and front switch callback function to get status. (Pass NULL to disable the callback.)

12.11.4.44 int device_SendDataCommandNEO (IN int *cmd,* IN int *subCmd,* IN BYTE * *data,* IN int *dataLen,* OUT BYTE * *response,* IN_OUT int * *respLen*)

Send a Command to device

Sends a command to the device .

Parameters

cmd	buffer of command to execute.
cmdLen,the	length of the buffer cmd.
data	buffer of IDG command data.
dataLen,the	length of the buffer data.
response	Response data
respLen,the	length of Response data

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString() Send a Command to NEO device

Sends a command to the NEO device .

Parameters

cmd	command to execute.
subCmd,sub	command to execute.
data	buffer of NEO command data.
dataLen,the	length of the buffer data.
response	Response data
respLen,the	length of Response data

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.11.4.45 int device_setBurstMode (IN BYTE *mode*)

Send Burst Mode

Sets the burst mode for the device.

Parameters

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

Returns

success or error code. Values can be parsed with device_getIDGStatusCodeString

See Also

ErrorCode

12.11.4.46 int device_setCurrentDevice (int *deviceType*)

Sets the current device to talk to

The connect status can be checked by device_isConnected().

Parameters

deviceType	Device to connect to
	
	`enum DEVICE_TYPE`
	`{`
	`IDT_DEVICE_UNKNOWN=0,`
	`IDT_DEVICE_AUGUSTA_HID,`
	`IDT_DEVICE_AUGUSTA_KB,`
	`IDT_DEVICE_AUGUSTA_S_HID,`
	`IDT_DEVICE_AUGUSTA_S_KB,`
	`IDT_DEVICE_AUGUSTA_S_TTK_HID,`
	`IDT_DEVICE_SPECTRUM_PRO,`
	`IDT_DEVICE_MINISMART_II,`
	`IDT_DEVICE_L100,`
	`IDT_DEVICE_UNIPAY,`
	`IDT_DEVICE_UNIPAY_I_V,`
	`IDT_DEVICE_VP3300_AJ,`
	`IDT_DEVICE_KIOSK_III,`
	`IDT_DEVICE_KIOSK_III_S,`
	`IDT_DEVICE_PIP_READER,`
	`IDT_DEVICE_VENDI,`
	`IDT_DEVICE_VP3300_USB,`
	`IDT_DEVICE_UNIPAY_I_V_TTK,`
	`IDT_DEVICE_VP3300_BT,`
	`IDT_DEVICE_VP8800,`
	`IDT_DEVICE_SREDKEY2_HID,`
	`IDT_DEVICE_SREDKEY2_KB,`
	`IDT_DEVICE_NEO2,`
	`IDT_DEVICE_MINISMART_II_COM = IDT_DEVICE_NEO2+5,`
	`IDT_DEVICE_SPECTRUM_PRO_COM,`
	`IDT_DEVICE_KIOSK_III_COM,`
	`IDT_DEVICE_KIOSK_III_S_COM,`
	`IDT_DEVICE_VP3300_COM,`
	`IDT_DEVICE_NEO2_COM,`
	`IDT_DEVICE_MAX_DEVICES = IDT_DEVICE_NEO2_COM+5`
	`};`

Returns

RETURN_CODE: 1: success, 0: failed

12.11.4.47 int device_setMerchantRecord (int *index,* int *enabled,* char * *merchantID,* char * *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 device_getIDGStatusCodeString()

12.11.4.48 int device_setPollMode (IN BYTE *mode*)

Set 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 device_getIDGStatusCodeString()

12.11.4.49 void device_setSDKWaitTime (int *waitTime*)

Set SDK Wait Time

Set the SDK wait time for transactions

Parameters

waitTime	The SDK wait time for transaction in seconds

12.11.4.50 void device_setThreadStackSize (int *threadSize*)

Set Thread Stack Size

Set the stack size setting for newly created threads

12.11.4.51 void emv_registerCallBk (pEMV_callBack *pEMVf*)

To register the emv callback function to get the EMV processing response. (Pass NULL to disable the callback.)

12.11.4.52 int msr_cancelMSRSwipe ( )

Disable MSR Swipe Cancels MSR swipe request.

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.11.4.53 void msr_registerCallBk (pMSR_callBack *pMSRf*)

To register the msr callback function to get the MSR card data. (Pass NULL to disable the callback.)

12.11.4.54 void msr_registerCallBkp (pMSR_callBackp *pMSRf*)

To register the msr callback function to get the MSR card data pointer. (Pass NULL to disable the callback.)

12.11.4.55 int msr_startMSRSwipe (IN int *_timeout*)

Start MSR Swipe Enables MSR, waiting for swipe to occur. Allows track selection. Returns IDTMSRData instance to deviceDelegate::swipeMSRData:()

Parameters

timeout	Swipe Timeout Value

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString() Note: if auto poll mode is on, it will return command not allowed error

12.11.4.56 void parseMSRData (IN BYTE * *resData*, IN int *resLen*, IN_OUT IDTMSRData * *cardData*)

Parser the MSR data from the buffer into IDTMSTData structure

Parameters

resData	MSR card data buffer
resLen	the length of resData
cardData	the parser result with IDTMSTData structure

12.11.4.57 void pin_registerCallBk (pPIN_callBack *pPINf*)

To register the pin callback function to get the PINPad data. (Pass NULL to disable the callback.)

12.11.4.58 void registerHotplugCallBk (pMessageHotplug *pMsgHotplug*)

To register the USB HID hot-plug callback function which implemented in the application to monitor the hotplug message from the SDK.

12.11.4.59 void registerLogCallBk (pSendDataLog *pFSend*, pReadDataLog *pFRead*)

To register the log callback function which implemented in the application to monitor sending/reading data between application and reader.

12.11.4.60 char* SDK_Version ( )

To Get SDK version

Returns

return the SDK version string

12.11.4.61 int setAbsoluteLibraryPath (const char * *absoluteLibraryPath*)

Set the path to use when searching for ID TECH's libraries. If this is not set, the libraries will be searched for with the system's default procedures.

Parameters

absoluteLibrary-Path	The absolute path to ID TECH's libraries.

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.12 Source_C/libIDT_VP3300_AJ.h File Reference

VP3300 AJ API.

```
#include "IDTDef.h"
```

Macros

- #define IN
- #define OUT
- #define IN_OUT

Typedefs

- typedef void(∗ pMessageHotplug)(int, int)
- typedef void(∗ pSendDataLog)(unsigned char ∗, int)
- typedef void(∗ pReadDataLog)(unsigned char ∗, int)
- typedef void(∗ pEMV_callBack)(int, int, unsigned char ∗, int, IDTTransactionData ∗, EMV_Callback ∗, int)
- typedef void(∗ pMSR_callBack)(int, IDTMSRData)
- typedef void(∗ pMSR_callBackp)(int, IDTMSRData ∗)
- typedef void(∗ pPIN_callBack)(int, IDTPINData ∗)
- typedef void(∗ pCMR_callBack)(int, IDTCMRData ∗)
- typedef void(∗ pCSFS_callBack)(BYTE status)
- typedef void(∗ ftpComm_callBack)(int, int, int)
- typedef void(∗ httpComm_callBack)(BYTE ∗, int)
- typedef void(∗ v4Comm_callBack)(BYTE, BYTE, BYTE ∗, int)

Functions

- void registerHotplugCallBk (pMessageHotplug pMsgHotplug)
- void registerLogCallBk (pSendDataLog pFSend, pReadDataLog pFRead)
- void device_registerRKICallBk (pRKI_callBack pRKIf)
- void emv_registerCallBk (pEMV_callBack pEMVf)
- void msr_registerCallBk (pMSR_callBack pMSRf)
- void msr_registerCallBkp (pMSR_callBackp pMSRf)
- void ctls_registerCallBk (pMSR_callBack pCTLSf)
- void ctls_registerCallBkp (pMSR_callBackp pCTLSf)
- void pin_registerCallBk (pPIN_callBack pPINf)
- void device_registerCameraCallBk (pCMR_callBack pCMRf)
- void device_registerCardStatusFrontSwitchCallBk (pCSFS_callBack pCSFSf)
- int device_getSDKWaitTime ()
- void device_setSDKWaitTime (int waitTime)
- int device_getThreadStackSize ()
- void device_setThreadStackSize (int threadSize)
- void comm_registerHTTPCallback (httpComm_callBack cBack)
- void comm_registerV4Callback (v4Comm_callBack cBack)
- char ∗ SDK_Version ()
- int setAbsoluteLibraryPath (const char ∗absoluteLibraryPath)
- int device_init ()
- int device_setCurrentDevice (int deviceType)
- int device_close ()
- void device_getIDGStatusCodeString (IN int returnCode, OUT char ∗despcrition)

- int device_isConnected ()
- int device_isAttached (int deviceType)
- int device_getFirmwareVersion (OUT char *firmwareVersion)
- int device_getFirmwareVersion_Len (OUT char *firmwareVersion, IN_OUT int *firmwareVersionLen)
- int device_pingDevice ()
- int device_controlUserInterface (IN BYTE *values)
- int device_getCurrentDeviceType ()
- int device_SendDataCommandNEO (IN int cmd, IN int subCmd, IN BYTE *data, IN int dataLen, OUT BYTE *response, IN_OUT int *respLen)
- int device_enablePassThrough (int enablePassThrough)
- int device_setBurstMode (IN BYTE mode)
- int device_setPollMode (IN BYTE mode)
- int device_setMerchantRecord (int index, int enabled, char *merchantID, char *merchantURL)
- int device_getMerchantRecord (IN int index, OUT BYTE *record)
- int device_getMerchantRecord_Len (IN int index, OUT BYTE *record, IN_OUT int *recordLen)
- int device_getTransactionResults (IDTMSRData *cardData)
- int device_startTransaction (IN double amount, IN double amtOther, IN int type, IN const int _timeout, IN BYTE *tags, IN int tagsLen)
- void device_setTransactionExponent (int exponent)
- int device_activateTransaction (IN const int _timeout, IN BYTE *tags, IN int tagsLen)
- int device_cancelTransaction ()
- int device_getRTCDateTime (IN BYTE *dateTime, IN_OUT int *dateTimeLen)
- int device_setRTCDateTime (IN BYTE *dateTime, IN int dateTimeLen)
- int device_startRKI (const char *caPath)
- int config_getSerialNumber (OUT char *sNumber)
- int config_getSerialNumber_Len (OUT char *sNumber, IN_OUT int *sNumberLen)
- int ctls_startTransaction (IN double amount, IN double amtOther, IN int type, IN const int _timeout, IN BYTE *tags, IN int tagsLen)
- int ctls_activateTransaction (IN const int _timeout, IN BYTE *tags, IN int tagsLen)
- int ctls_cancelTransaction ()
- int ctls_retrieveApplicationData (IN BYTE *AID, IN int AIDLen, OUT BYTE *tlv, IN_OUT int *tlvLen)
- int ctls_setApplicationData (IN BYTE *tlv, IN int tlvLen)
- int ctls_removeApplicationData (IN BYTE *AID, IN int AIDLen)
- int ctls_removeAllApplicationData ()
- int ctls_retrieveAIDList (OUT BYTE *AIDList, IN_OUT int *AIDListLen)
- int ctls_retrieveTerminalData (OUT BYTE *tlv, IN_OUT int *tlvLen)
- int ctls_setTerminalData (IN BYTE *tlv, IN int tlvLen)
- int ctls_retrieveCAPK (IN BYTE *capk, IN int capkLen, OUT BYTE *key, IN_OUT int *keyLen)
- int ctls_setCAPK (IN BYTE *capk, IN int capkLen)
- int ctls_removeCAPK (IN BYTE *capk, IN int capkLen)
- int ctls_removeAllCAPK ()
- int ctls_retrieveCAPKList (OUT BYTE *keys, IN_OUT int *keysLen)
- int ctls_setConfigurationGroup (IN BYTE *tlv, IN int tlvLen)
- int ctls_getConfigurationGroup (IN int group, OUT BYTE *tlv, OUT int *tlvLen)
- int ctls_getAllConfigurationGroups (OUT BYTE *tlv, IN_OUT int *tlvLen)
- int ctls_removeConfigurationGroup (int group)
- void emv_allowFallback (IN int allow)
- void emv_setAutoAuthenticateTransaction (IN int authenticate)
- void emv_setAutoCompleteTransaction (IN int complete)
- int emv_getAutoAuthenticateTransaction ()
- int emv_getAutoCompleteTransaction ()
- void emv_setTransactionParameters (IN double amount, IN double amtOther, IN int type, IN int timeout, IN BYTE *tags, IN int tagsLen)
- int emv_startTransaction (IN double amount, IN double amtOther, IN int exponent, IN int type, IN int timeout, IN BYTE *tags, IN int tagsLen, IN int forceOnline)

- int emv_activateTransaction (IN int timeout, IN BYTE *tags, IN int tagsLen, IN int forceOnline)
- int emv_authenticateTransaction (IN BYTE *updatedTLV, IN int updatedTLVLen)
- int emv_authenticateTransactionWithTimeout (IN int timeout, IN BYTE *updatedTLV, IN int updatedTLVLen)
- int emv_completeTransaction (IN int commError, IN BYTE *authCode, IN int authCodeLen, IN BYTE *iad, IN int iadLen, IN BYTE *tlvScripts, IN int tlvScriptsLen, IN BYTE *tlv, IN int tlvLen)
- int emv_cancelTransaction ()
- int emv_retrieveApplicationData (IN BYTE *AID, IN int AIDLen, OUT BYTE *tlv, IN_OUT int *tlvLen)
- int emv_setApplicationData (IN BYTE *name, IN int nameLen, IN BYTE *tlv, IN int tlvLen)
- int emv_setApplicationDataTLV (IN BYTE *tlv, IN int tlvLen)
- int emv_removeApplicationData (IN BYTE *AID, IN int AIDLen)
- int emv_removeAllApplicationData ()
- int emv_retrieveAIDList (OUT BYTE *AIDList, IN_OUT int *AIDListLen)
- int emv_retrieveTerminalData (OUT BYTE *tlv, IN_OUT int *tlvLen)
- int emv_setTerminalData (IN BYTE *tlv, IN int tlvLen)
- int emv_setTerminalMajorConfiguration (IN int configuration)
- int emv_retrieveCAPK (IN BYTE *capk, IN int capkLen, OUT BYTE *key, IN_OUT int *keyLen)
- int emv_setCAPK (IN BYTE *capk, IN int capkLen)
- int emv_removeCAPK (IN BYTE *capk, IN int capkLen)
- int emv_removeAllCAPK ()
- int emv_retrieveCAPKList (OUT BYTE *keys, IN_OUT int *keysLen)
- int emv_retrieveCRL (OUT BYTE *list, IN_OUT int *lssLen)
- int emv_setCRL (IN BYTE *list, IN int lsLen)
- int emv_removeCRL (IN BYTE *list, IN int lsLen)
- int emv_removeAllCRL ()
- int icc_getICCReaderStatus (OUT BYTE *status)
- int icc_powerOnICC (OUT BYTE *ATR, IN_OUT int *inLen)
- int icc_powerOffICC ()
- int icc_exchangeAPDU (IN BYTE *c_APDU, IN int cLen, OUT BYTE *reData, IN_OUT int *reLen)
- int msr_cancelMSRSwipe ()
- int msr_startMSRSwipe (IN int _timeout)
- void parseMSRData (IN BYTE *resData, IN int resLen, IN_OUT IDTMSRData *cardData)

12.12.1 Detailed Description

VP3300 AJ API. VP3300 AJ Global API methods.

12.12.2 Macro Definition Documentation

12.12.2.1 #define IN

INPUT parameter.

12.12.2.2 #define IN_OUT

INPUT / OUTPUT PARAMETER.

12.12.2.3 #define OUT

OUTPUT parameter.

12.12.3 Typedef Documentation

12.12.3.1 typedef void(* ftpComm_callBack)(int, int, int)

Define the comm callback function to get FTP file transfer status

It should be passed as a parameter in a FTP request, Signature (int, int, int) = response code, current block, total blocks RESPONSE CODES: 100 = FILE DOWNLOAD STARTED 101 = FILE BLOCK XX OF XX RECEIVED 102 = FILE DOWNLOAD COMPLETED 103 = FILE DOWNLOAD TERMINATED PREMATURELY

12.12.3.2 typedef void(* httpComm_callBack)(BYTE *, int)

Define the comm callback function to get the async url data

It should be registered using the comm_registerHTTPCallback

12.12.3.3 typedef void(* pCMR_callBack)(int, IDTCMRData *)

Define the camera callback function to get the image data

It should be registered using the device_registerCameraCallBk,

12.12.3.4 typedef void(* pCSFS_callBack)(BYTE status)

Define the card status and front switch callback function to get card and front switch status

It should be registered using the device_registerCardStatusFrontSwitchCallBk,

12.12.3.5 typedef void(* pEMV_callBack)(int, int, unsigned char *, int, IDTTransactionData *, EMV_Callback *, int)

Define the EMV callback function to get the transaction message/data/result.

It should be registered using the emv_registerCallBk,

12.12.3.6 typedef void(* pMessageHotplug)(int, int)

Define the USB hot-plug callback function to monitor the info when plug in/out the reader.

It should be registered using the registerHotplugCallBk, The first integer parameter is device type, and the second integer parameter is either 0: Device Plugged Out or 1: Device Plugged In

12.12.3.7 typedef void(* pMSR_callBack)(int, IDTMSRData)

Define the MSR callback function to get the MSR card data

It should be registered using the msr_registerCallBk, this callback function is for backward compatibility

12.12.3.8 typedef void(* pMSR_callBackp)(int, IDTMSRData *)

Define the MSR callback function to get pointer to the MSR card data

It should be registered using the msr_registerCallBk, this callback function is recommended instead of pMSR_call-Back

12.12.3.9 typedef void(* pPIN_callBack)(int, IDTPINData *)

Define the PINPad callback function to get the input PIN Pad data

It should be registered using the pin_registerCallBk,

12.12.3.10 typedef void(* pReadDataLog)(unsigned char *, int)

Define the read response callback function to monitor the reading response from the reader.

It should be registered using the registerLogCallBk,

12.12.3.11 typedef void(* pSendDataLog)(unsigned char *, int)

Define the send command callback function to monitor the sending command into the reader.

It should be registered using the registerLogCallBk,

12.12.3.12 typedef void(* v4Comm_callBack)(BYTE, BYTE, BYTE *, int)

Define the comm callback function to receive the V4 Protocol packets received by the device from an external source (IP/USB/RS-232) It should be registered using the comm_registerV4Callback, Data callback will contain command, sub-command, and data from V4 packet

12.12.4 Function Documentation

12.12.4.1 void comm_registerHTTPCallback (httpComm_callBack *cBack*)

Register Comm HTTP Async Callback

Parameters

cBack	- HTTP Comm callback

12.12.4.2 void comm_registerV4Callback (v4Comm_callBack *cBack*)

Register External V4 Protocol commands Callback

Parameters

cBack	- V4 Protocol Comm callback

12.12.4.3 int config_getSerialNumber (OUT char * *sNumber*)

DEPRECATED : please use config_getSerialNumber_Len(OUT char* sNumber, IN_OUT int *sNumberLen)

Polls device for Serial Number

Parameters

sNumber	Returns Serial Number; needs to have at least 64 bytes of memory

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString

12.12.4.4 int config_getSerialNumber_Len (OUT char * *sNumber,* IN_OUT int * *sNumberLen*)

Polls device for Serial Number

Parameters

sNumber	Returns Serial Number
sNumberLen	Length of Serial Number

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString

12.12.4.5 int ctls_activateTransaction (IN const int *_timeout,* IN BYTE * *tags,* IN int *tagsLen*)

Start CTLS Transaction Request

Authorizes the CTLS 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 stream. Example, tag 9F0C with amount 0x000000000100 would be 0x9F0C06000000000100
tagsLen	The length of tags data buffer.

>>>>>IMPORTANT: parameters for amount and amtOther MUST BE PASSED AS A DOUBLE VALUE WITH DECIMAL POINT. Example, do not pass 1, but instead pass 1.0 or 1.00. Otherwise, results will be unpredictable

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString() Note: if auto poll is on, it will returm the error IDG_P2_STATUS_CODE_COMMAND_NOT_ALLOWED

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

12.12.4.6 int ctls_cancelTransaction ( )

Cancel EMV Transaction

Cancels the currently executing EMV transaction.

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.12.4.7 int ctls_getAllConfigurationGroups (OUT BYTE * *tlv*, IN_OUT int * *tlvLen*)

Retrieve All Configuration Groups

Returns all the Configuration Groups installed on the terminal for CTLS

Parameters

tlv	The TLV elements data
tlvLen	the length of tlv data buffer.

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.12.4.8 int ctls_getConfigurationGroup (IN int *group*, OUT BYTE * *tlv*, OUT int * *tlvLen*)

Get Configuration Group

Retrieves the Configuration for the specified Group.

Parameters

group	Configuration Group
tlv	return data
tlvLen	the length of tlv data buffer

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.12.4.9 void ctls_registerCallBk (pMSR_callBack *pCTLSf*)

To register the ctls callback function to get the MSR card data. (Pass NULL to disable the callback.)

12.12.4.10 void ctls_registerCallBkp (pMSR_callBackp *pCTLSf*)

To register the ctls callback function to get the MSR card data pointer. (Pass NULL to disable the callback.)

12.12.4.11 int ctls_removeAllApplicationData ( )

Remove All Application Data

Removes all the Application Data

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.12.4.12 int ctls_removeAllCAPK ( )

Remove All Certificate Authority Public Key

Removes all the CAPK

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.12.4.13 int ctls_removeApplicationData (IN BYTE * *AID*, IN int *AIDLen*)

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
AIDLen	the length of AID data buffer

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.12.4.14 int ctls_removeCAPK (IN BYTE * *capk*, IN int *capkLen*)

Remove Certificate Authority Public Key

Removes the CAPK as specified by the RID/Index

Parameters

capk	6 byte CAPK = 5 bytes RID + 1 byte INDEX
capkLen	the length of capk data buffer

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.12.4.15 int 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	Values can be parsed with device_getIDGStatusCodeString()

12.12.4.16 int ctls_retrieveAIDList (OUT BYTE * *AIDList*, IN_OUT int * *AIDListLen*)

Retrieve AID list

Returns all the AID names installed on the terminal for CTLS. .

Parameters

AIDList	array of AID name byte arrays
AIDListLen	the length of AIDList array buffer

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.12.4.17 int ctls_retrieveApplicationData (IN BYTE * *AID*, IN int *AIDLen*, OUT BYTE * *tlv*, IN_OUT int * *tlvLen*)

Retrieve Application Data by AID

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

Parameters

AID	Name of ApplicationID. Must be between 5 and 16 bytes
AIDLen	the length of AID data buffer.
tlv	The TLV elements of the requested AID
tlvLen	the length of tlv data buffer.

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.12.4.18 int ctls_retrieveCAPK (IN BYTE * *capk*, IN int *capkLen*, OUT BYTE * *key*, IN_OUT int * *keyLen*)

Retrieve Certificate Authority Public Key

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

Parameters

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

capkLen	the length of capk data buffer
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.
keyLen	the length of key data buffer •

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.12.4.19 int ctls_retrieveCAPKList (OUT BYTE * *keys*, IN_OUT int * *keysLen*)

Retrieve the Certificate Authority Public Key list

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

Parameters

keys	[key1][key2]...[keyn], each key 6 bytes where key = 5 bytes RID + 1 byte index
keysLen	the length of keys data buffer

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.12.4.20 int ctls_retrieveTerminalData (OUT BYTE * *tlv*, IN_OUT int * *tlvLen*)

Retrieve Terminal Data

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

Parameters

tlv	Response returned as a TLV
tlvLen	the length of tlv data buffer

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.12.4.21 int ctls_setApplicationData (IN BYTE * *tlv*, IN int *tlvLen*)

Set Application Data by AID

Sets the Application Data for CTLS as specified by TLV data

Parameters

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

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

Parameters

tlvLen	the length of tlv data buffer

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.12.4.22 int ctls_setCAPK (IN BYTE * *capk*, IN int *capkLen*)

Set Certificate Authority Public Key

Sets the CAPK as specified by the CAKey structure

Parameters

capk	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.

capkLen	the length of capk data buffer

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.12.4.23 int ctls_setConfigurationGroup (IN BYTE * *tlv,* IN int *tlvLen*)

Set Configuration Group

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

Parameters

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

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.12.4.24 int ctls_setTerminalData (IN BYTE * *tlv,* IN int *tlvLen*)

Set Terminal Data

Sets the Terminal Data for CTLS as specified by the TLV. The first TLV must be Configuration Group Number (Tag FFE4). The terminal global data is group 0, so the first TLV would be FFE40100. Other groups can be defined using this method (1 or greater), and those can be retrieved with ctls_getConfigurationGroup(int group), and deleted with ctls_removeConfigurationGroup(int group). You cannot delete group 0.

Parameters

tlv	TerminalData configuration file
tlvLen	the length of tlv data buffer

Return values

RETURN_CODE	Values can be parsed with device_getIDGStatusCodeString()

12.12.4.25 int ctls_startTransaction (IN double *amount,* IN double *amtOther,* IN int *type,* IN const int *_timeout,* IN BYTE * *tags,* IN int *tagsLen*)

Start CTLS Transaction Request

Authorizes the CTLS transaction for an ICC card

The tags will be returned in the callback routine.

Parameters

amount	Transaction amount value (tag value 9F02) - SEE IMPORTANT NOTE BELOW
amtOther	Other amount value, if any (tag value 9F03) - SEE IMPORTANT NOTE BELOW
type	Transaction type (tag value 9C).
timeout	Timeout value in seconds.
tags	Any other tags to be included in the request. Passed as TLV stream. Example, tag 9F0C with amount 0x000000000100 would be 0x9F0C06000000000100 If tags 9F02 (amount),9-F03 (other amount), or 9C (transaction type) are included, they will take priority over these values supplied as individual parameters to this method.
tagsLen	The length of tags data buffer.

>>>>>IMPORTANT: parameters for amount and amtOther MUST BE PASSED AS A DOUBLE VALUE WITH DECIMAL POINT. Example, do not pass 1, but instead pass 1.0 or 1.00. Otherwise, results will be unpredictable

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString() Note: if auto poll is on, it will returm the error IDG_P2_STATUS_CODE_COMMAND_NOT_ALLOWED

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

12.12.4.26 int device_activateTransaction (IN const int *_timeout,* IN BYTE * *tags,* IN int *tagsLen*)

Start Transaction Request

Authorizes the transaction for an MSR/CTLS/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 a TLV. Example, tag 9F0C with amount 0x000000000100 would be 0x9F0C06000000000100 Be sure to include 9F02 (amount)and9-C (transaction type).
tagsLen	The length of tags data buffer.

>>>>>IMPORTANT: parameters for amount and amtOther MUST BE PASSED AS A DOUBLE VALUE WITH DECIMAL POINT. Example, do not pass 1, but instead pass 1.0 or 1.00. Otherwise, results will be unpredictable

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString() Note: if auto poll is on, it will returm the error IDG_P2_STATUS_CODE_COMMAND_NOT_ALLOWED

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 device_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

12.12.4.27 int device_cancelTransaction ( )

Disable Transaction Cancel Transaction request.

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.12.4.28 int device_close ( )

Close the device

Returns

RETURN_CODE: 0: success, 0x0A: failed

12.12.4.29 int device_controlUserInterface (IN BYTE * *values*)

Control User Interface

Controls the User Interface: Display, Beep, LED

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

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.12.4.30 int device_enablePassThrough (int *enablePassThrough*)

Start Remote Key Injection

Starts a remote key injection request with IDTech RKI servers. This function is reserved and not implemented.

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

Enable Pass Through

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

Parameters

enablePass-Through	1 = pass through ON, 0 = pass through OFF

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.12.4.31 int device_getCurrentDeviceType ( )

Get current active device type

Returns

: return the device type defined as DEVICE_TYPE in the IDTDef.h

12.12.4.32 int device_getFirmwareVersion (OUT char * *firmwareVersion*)

DEPRECATED : please use device_getFirmwareVersion_Len(OUT char* firmwareVersion, IN_OUT int *firmware-VersionLen)

Polls device for Firmware Version

Parameters

firmwareVersion	Response returned of Firmware Version; needs to have at least 128 bytes of memory

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.12.4.33 int device_getFirmwareVersion_Len (OUT char * *firmwareVersion,* IN_OUT int * *firmwareVersionLen*)

Polls device for Firmware Version

Parameters

firmwareVersion	Response returned of Firmware Version
firmwareVersion-Len	Length of Firmware Version

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.12.4.34 void device_getIDGStatusCodeString (IN int *returnCode,* OUT char * *despcrition*)

Review the return code description.

Parameters

returnCode	the response result.
description	

Return values

the	string for description of response result

- 0: "no error, beginning task";
- 1: "no response from reader";
- 2: "invalid response data";
  - 01: " Incorrect Header Tag";
  - 02: " Unknown Command";
  - 03: " Unknown Sub-Command";
  - 04: " CRC Error in Frame";
  - 05: " Incorrect Parameter";
  - 06: " Parameter Not Supported";
  - 07: " Mal-formatted Data";
  - 08: " Timeout";
  - 0A: " Failed / NACK";
  - 0B: " Command not Allowed";
  - 0C: " Sub-Command not Allowed";
  - 0D: " Buffer Overflow (Data Length too large for reader buffer)";
  - 0E: " User Interface Event";
  - 10: " Need clear firmware(apply in boot loader only)";
  - 11: " Communication type not supported, VT-1, burst, etc. Need encrypted firmware (apply in boot loader only)";
  - 12: " Secure interface is not functional or is in an intermediate state.";
  - 13: " Data field is not mod 8";
  - 14: " Pad 0x80 not found where expected";
  - 15: " Specified key type is invalid";
  - 16: " Could not retrieve key from the SAM (InitSecureComm)";
  - 17: " Hash code problem";
  - 18: " Could not store the key into the SAM (InstallKey)";
  - 19: " Frame is too large";
  - 1A: " Unit powered up in authentication state but POS must resend the InitSecureComm command";
  - 1B: " The EEPROM may not be initialized because SecCommInterface does not make sense";
  - 1C: " Problem encoding APDU Module-Specific Status Codes ";
  - 20: " Unsupported Index (ILM) SAM Transceiver error - problem communicating with the SAM (Key Mgr)";
  - 21: " Unexpected Sequence Counter in multiple frames for single bitmap (ILM)Length error in data returned from the SAM (Key Mgr) - 22: " Improper bit map (ILM)"; - 23: " Request Online Authorization"; - 24: " ViVOCard3 raw data read successful"; - 25: " Message index not available (ILM) ViVOcomm activate transaction card type (ViVOcomm)"; - 26: " Version Information Mismatch (ILM)"; - 27: " Not sending commands in correct index message index (ILM)"; - 28: " Time out or next expected message not received (ILM)"; - 29: " ILM languages not available for viewing (ILM)"; - 2A: " Other language not supported (ILM)"; - 41: " from 41 to 4F, Module-specific errors for Key Manager";

- 50: " Auto-Switch OK";
- 51: " Auto-Switch failed";
- 70: " Antenna Error 80h Use another card";
- 81: " Insert or swipe card";
- 90: " Data encryption Key does not exist";
- 91: " Data encryption Key KSN exhausted";

12.12.4.35 int device_getMerchantRecord (IN int *index,* OUT BYTE * *record*)

DEPRECATED : please use device_getMerchantRecord_Len(IN int index, OUT BYTE * record, IN_OUT int *recordLen)

Get Merchant Record

Gets the merchant record for the device.

Parameters

index	Merchant Record index, valid values 1-6
record;	needs to have at least 99 bytes of memory response data from reader. Merchant Record Index: 1 byte enabled: 1 byte Merchant ID: 32 bytes Length of Merchant URL: 1 byte Merchant URL: 64 bytes

Returns

success or error code. Values can be parsed with device_getIDGStatusCodeString()

See Also

ErrorCode

12.12.4.36 int device_getMerchantRecord_Len (IN int *index,* OUT BYTE * *record,* IN_OUT int * *recordLen*)

Get Merchant Record

Gets the merchant record for the device.

Parameters

index	Merchant Record index, valid values 1-6
record	response data from reader. Merchant Record Index: 1 byte enabled: 1 byte Merchant ID: 32 bytes Length of Merchant URL: 1 byte Merchant URL: 64 bytes
recordLen	Length of record

Returns

success or error code. Values can be parsed with device_getIDGStatusCodeString()

See Also

ErrorCode

12.12.4.37 int device_getRTCDateTime (IN BYTE * *dateTime,* IN_OUT int * *dateTimeLen*)

get RTC date and time of the device

Parameters

dateTime	<dateTime data=""> is: 6 byte data for YYMMDDHHMMSS in hex. For example 0x171003102547 stands for 2017 Oct 3rd 10:25:47
dateTimeLen	return 6 bytes if successful

Returns

success or error code. Values can be parsed with device_getResponseCodeString

See Also

ErrorCode

12.12.4.38 int device_getSDKWaitTime ( )

Get SDK Wait Time

Get the SDK wait time for transactions

Returns

SDK wait time in seconds

12.12.4.39 int device_getThreadStackSize ( )

Get Thread Stack Size

Get the stack size setting for newly created threads

Returns

Thread Stack Size

12.12.4.40 int device_getTransactionResults (IDTMSRData * *cardData*)

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

Parameters

cardData	The transaction results

Returns

success or error code. Values can be parsed with device_getResponseCodeString

See Also

ErrorCode

12.12.4.41 int device_init ( )

Initial the device by USB

It will detect the device and trying connect.

The connect status can be checked by device_isConnected().

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.12.4.42 int device_isAttached (int *deviceType*)

Check if the device is attached to the USB port The function device_init() must be called before this function.

Parameters

deviceType,the	device type of the USB device

Returns

1 if the device is attached, or 0 if the device is not attached

12.12.4.43 int device_isConnected ( )

Check the device conntected status

Returns

DEVICE_DISCONNECT=0, or DEVICE_CONNECTED = 1

12.12.4.44 int device_pingDevice ( )

Ping Device

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

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.12.4.45 void device_registerCameraCallBk (pCMR_callBack *pCMRf*)

To register the camera callback function to get the image data. (Pass NULL to disable the callback.)

12.12.4.46 void device_registerCardStatusFrontSwitchCallBk (pCSFS_callBack *pCSFSf*)

To register the card status and front switch callback function to get status. (Pass NULL to disable the callback.)

12.12.4.47 void device_registerRKICallBk (pRKI_callBack *pRKIf*)

To register the RKI callback function to get the RKI status. (Pass NULL to disable the callback.)

12.12.4.48 int device_SendDataCommandNEO (IN int *cmd*, IN int *subCmd*, IN BYTE * *data*, IN int *dataLen*, OUT BYTE * *response*, IN_OUT int * *respLen*)

Send a Command to device

Sends a command to the device .

Parameters

cmd	buffer of command to execute.
cmdLen,the	length of the buffer cmd.
data	buffer of IDG command data.
dataLen,the	length of the buffer data.
response	Response data
respLen,the	length of Response data

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString() Send a Command to NEO device

Sends a command to the NEO device .

Parameters

cmd	command to execute.
subCmd,sub	command to execute.
data	buffer of NEO command data.
dataLen,the	length of the buffer data.
response	Response data
respLen,the	length of Response data

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.12.4.49 int device_setBurstMode (IN BYTE *mode*)

Send Burst Mode

Sets the burst mode for the device.

Parameters

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

Returns

success or error code. Values can be parsed with device_getIDGStatusCodeString

See Also

ErrorCode

12.12.4.50 int device_setCurrentDevice (int *deviceType*)

Sets the current device to talk to

The connect status can be checked by device_isConnected().

Parameters

deviceType	Device to connect to
	
	`enum DEVICE_TYPE`
	`{`
	`IDT_DEVICE_UNKNOWN=0,`
	`IDT_DEVICE_AUGUSTA_HID,`
	`IDT_DEVICE_AUGUSTA_KB,`
	`IDT_DEVICE_AUGUSTA_S_HID,`
	`IDT_DEVICE_AUGUSTA_S_KB,`
	`IDT_DEVICE_AUGUSTA_S_TTK_HID,`
	`IDT_DEVICE_SPECTRUM_PRO,`
	`IDT_DEVICE_MINISMART_II,`
	`IDT_DEVICE_L100,`
	`IDT_DEVICE_UNIPAY,`
	`IDT_DEVICE_UNIPAY_I_V,`
	`IDT_DEVICE_VP3300_AJ,`
	`IDT_DEVICE_KIOSK_III,`
	`IDT_DEVICE_KIOSK_III_S,`
	`IDT_DEVICE_PIP_READER,`
	`IDT_DEVICE_VENDI,`
	`IDT_DEVICE_VP3300_USB,`
	`IDT_DEVICE_UNIPAY_I_V_TTK,`
	`IDT_DEVICE_VP3300_BT,`
	`IDT_DEVICE_VP8800,`
	`IDT_DEVICE_SREDKEY2_HID,`
	`IDT_DEVICE_SREDKEY2_KB,`
	`IDT_DEVICE_NEO2,`
	`IDT_DEVICE_MINISMART_II_COM = IDT_DEVICE_NEO2+5,`
	`IDT_DEVICE_SPECTRUM_PRO_COM,`
	`IDT_DEVICE_KIOSK_III_COM,`
	`IDT_DEVICE_KIOSK_III_S_COM,`
	`IDT_DEVICE_VP3300_COM,`
	`IDT_DEVICE_NEO2_COM,`
	`IDT_DEVICE_MAX_DEVICES = IDT_DEVICE_NEO2_COM+5`
	`};`

Returns

RETURN_CODE: 1: success, 0: failed

12.12.4.51 int device_setMerchantRecord (int *index,* int *enabled,* char * *merchantID,* char * *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 device_getIDGStatusCodeString()

12.12.4.52 int device_setPollMode (IN BYTE *mode*)

Set 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 device_getIDGStatusCodeString()

12.12.4.53 int device_setRTCDateTime (IN BYTE * *dateTime*, IN int *dateTimeLen*)

set RTC date and time of the device

Parameters

dateTime	<dateTime data=""> is: 6 byte data for YYMMDDHHMMSS in hex. For example 0x171003102547 stands for 2017 Oct 3rd 10:25:47
dateTimeLen	should be always 6 bytes

Returns

success or error code. Values can be parsed with device_getResponseCodeString

See Also

ErrorCode

12.12.4.54 void device_setSDKWaitTime (int *waitTime*)

Set SDK Wait Time

Set the SDK wait time for transactions

Parameters

waitTime	The SDK wait time for transaction in seconds

12.12.4.55 void device_setThreadStackSize (int *threadSize*)

Set Thread Stack Size

Set the stack size setting for newly created threads

12.12.4.56 void device_setTransactionExponent (int *exponent*)

Sets the transaction exponent to be used with device_startTransaction. Default value is 2

Parameters

exponent,The	exponent to use when calling device_startTransaction

12.12.4.57 int device_startRKI (const char * *caPath*)

Start remote key injection.

Parameters

caPath	The path to ca-certificates.crt

Returns

success or error code.

See Also

ErrorCode

12.12.4.58 int device_startTransaction (IN double *amount,* IN double *amtOther,* IN int *type,* IN const int *_timeout,* IN BYTE * *tags,* IN int *tagsLen*)

Start Transaction Request

Authorizes the transaction for an MSR/CTLS/ICC card

The tags will be returned in the callback routine.

Parameters

amount	Transaction amount value (tag value 9F02) - SEE IMPORTANT NOTE BELOW
amtOther	Other amount value, if any (tag value 9F03) - SEE IMPORTANT NOTE BELOW
type	Transaction type (tag value 9C).
timeout	Timeout value in seconds.
tags	Any other tags to be included in the request. Passed as TLV. Example, tag 9F0C with amount 0x000000000100 would be 0x9F0C06000000000100 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.
tagsLen	The length of tags data buffer.

>>>>>IMPORTANT: parameters for amount and amtOther MUST BE PASSED AS A DOUBLE VALUE WITH DECIMAL POINT. Example, do not pass 1, but instead pass 1.0 or 1.00. Otherwise, results will be unpredictable

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString() Note: if auto poll is on, it will returm the error IDG_P2_STATUS_CODE_COMMAND_NOT_ALLOWED

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 device_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

12.12.4.59 int emv_activateTransaction (IN int *timeout,* IN BYTE * *tags,* IN int *tagsLen,* IN int *forceOnline*)

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	Tags to be included in the request. Passed as a TLV stream. Example, tag 9F0C with amount 0x000000000100 would be 0x9F0C06000000000100
tagsLen	Length of tags
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 = DFEE1A079F029-F369f9F37

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString >>>>>IMPORTANT: parameters for amount and amtOther MUST BE PASSED AS A DOUBLE VALUE WITH DECIMAL POINT. Example, do not pass 1, but instead pass 1.0 or 1.00. Otherwise, results will be unpredictable

12.12.4.60 void emv_allowFallback (IN int *allow*)

Allow fallback for EMV transactions. Default is TRUE

Parameters

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

12.12.4.61 int emv_authenticateTransaction (IN BYTE * *updatedTLV,* IN int *updatedTLVLen*)

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.

Parameters

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, 9F37 to be included in response = DFEE1-A079F029F36959F37
updatedTLVLen	

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.12.4.62 int emv_authenticateTransactionWithTimeout (IN int *timeout,* IN BYTE * *updatedTLV,* IN int *updatedTLVLen*)

Authenticate EMV Transaction Request with Timeout

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.

Parameters

timeout	Timeout value in seconds.
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, 9F37 to be included in response = DFEE1-A079F029F36959F37

updatedTLVLen	

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.12.4.63 int emv_cancelTransaction ( )

Cancel EMV Transaction

Cancels the currently executing EMV transaction.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.12.4.64 int emv_completeTransaction (IN int *commError*, IN BYTE * *authCode*, IN int *authCodeLen*, IN BYTE * *iad*, IN int *iadLen*, IN BYTE * *tlvScripts*, IN int *tlvScriptsLen*, IN BYTE * *tlv*, IN int *tlvLen*)

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(1) if host was unreachable, or FALSE(0) 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
authCodeLen	the length of authCode
iad	Issuer Authentication Data, if any. Example 0x11223344556677883030 (tag value 91).
iadLen	the length of iadLen
tlvScripts	71/72 scripts, if any
tlvScriptsLen	the length of tlvScriptsLen
tlv	Additional TLV data to return with transaction results (if any)
tlvLen	the length of tlv

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.12.4.65 int emv_getAutoAuthenticateTransaction ( )

Gets auto authenticate value for EMV transactions.

Returns

RETURN_CODE: TRUE = auto authenticate, FALSE = manually authenticate

12.12.4.66 int emv_getAutoCompleteTransaction ( )

Gets auto complete value for EMV transactions.

Returns

RETURN_CODE: TRUE = auto complete, FALSE = manually complete

12.12.4.67 void emv_registerCallBk (pEMV_callBack *pEMVf*)

To register the emv callback function to get the EMV processing response. (Pass NULL to disable the callback.)

12.12.4.68 int emv_removeAllApplicationData ( )

Remove All Application Data

Removes all the Application Data

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.12.4.69 int emv_removeAllCAPK ( )

Remove All Certificate Authority Public Key

Removes all the CAPK

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.12.4.70 int emv_removeAllCRL ( )

Remove All Certificate Revocation List Entries

Removes all CRLEntry entries

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.12.4.71 int emv_removeApplicationData (IN BYTE * *AID,* IN int *AIDLen*)

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
AIDLen	the length of AID data buffer

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.12.4.72 int emv_removeCAPK (IN BYTE * *capk,* IN int *capkLen*)

Remove Certificate Authority Public Key

Removes the CAPK as specified by the RID/Index

Parameters

capk	6 byte CAPK = 5 bytes RID + 1 byte INDEX
capkLen	the length of capk data buffer

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.12.4.73 int emv_removeCRL (IN BYTE * *list*, IN int *lsLen*)

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
lssLen	the length of list data buffer

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.12.4.74 int emv_retrieveAIDList (OUT BYTE * *AIDList*, IN_OUT int * *AIDListLen*)

Retrieve AID list

Returns all the AID names installed on the terminal for CTLS. .

Parameters

AIDList	array of AID name byte arrays
AIDListLen	the length of AIDList array buffer

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.12.4.75 int emv_retrieveApplicationData (IN BYTE * *AID*, IN int *AIDLen*, OUT BYTE * *tlv*, IN_OUT int * *tlvLen*)

Retrieve Application Data by AID

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

Parameters

AID	Name of ApplicationID. Must be between 5 and 16 bytes
AIDLen	the length of AID data buffer.
tlv	The TLV elements of the requested AID
tlvLen	the length of tlv data buffer.

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.12.4.76 int emv_retrieveCAPK (IN BYTE * *capk,* IN int *capkLen,* OUT BYTE * *key,* IN_OUT int * *keyLen*)

Retrieve Certificate Authority Public Key

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

Parameters

capk	6 bytes CAPK = 5 bytes RID + 1 byte Index
capkLen	the length of capk data buffer
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.
keyLen	the length of key data buffer •

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.12.4.77 int emv_retrieveCAPKList (OUT BYTE * *keys*, IN_OUT int * *keysLen*)

Retrieve the Certificate Authority Public Key list

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

Parameters

keys	[key1][key2]...[keyn], each key 6 bytes where key = 5 bytes RID + 1 byte index
keysLen	the length of keys data buffer

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.12.4.78 int emv_retrieveCRL (OUT BYTE * *list*, IN_OUT int * *lssLen*)

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
lssLen	the length of list data buffer

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.12.4.79 int emv_retrieveTerminalData (OUT BYTE * *tlv*, IN_OUT int * *tlvLen*)

Retrieve Terminal Data

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

Parameters

tlv	Response returned as a TLV
tlvLen	the length of tlv data buffer

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.12.4.80 int emv_setApplicationData (IN BYTE * *name*, IN int *nameLen*, IN BYTE * *tlv*, IN int *tlvLen*)

Set Application Data by AID

Sets the Application Data as specified by the application name and TLV data

Parameters

name	Application name, 10-32 ASCII hex characters representing 5-16 bytes Example "a0000000031010"
nameLen	the length of name data buffer of Application name,
tlv	Application data in TLV format
tlvLen	the length of tlv data buffer

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.12.4.81 int emv_setApplicationDataTLV (IN BYTE * *tlv*, IN int *tlvLen*)

Set Application Data by TLV

Sets the Application Data as specified by the 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) Example valid TLV, for Group #2, AID a0000000035010- : "dfee2d01029f0607a0000000051010ffe10101ffe50110ffe30114ffe20106"
tlvLen	the length of tlv data buffer

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.12.4.82 void emv_setAutoAuthenticateTransaction (IN int *authenticate*)

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

Parameters

authenticate	TRUE = auto authenticate, FALSE = manually authenticate

12.12.4.83 void emv_setAutoCompleteTransaction (IN int *complete*)

Enables complete for EMV transactions. If a emv_authenticateTranaction results in code 0x0004 (go online), then emv_completeTransaction can automatically execute if parameter is set to TRUE

Parameters

complete	TRUE = auto complete, FALSE = manually complete

12.12.4.84 int emv_setCAPK (IN BYTE * *capk,* IN int *capkLen*)

Set Certificate Authority Public Key

Sets the CAPK as specified by the CAKey structure

Parameters

capk	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.

capkLen	the length of capk data buffer

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.12.4.85 int emv_setCRL (IN BYTE * *list*, IN int *lsLen*)

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]
lsLen	the length of list data buffer

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.12.4.86 int emv_setTerminalData (IN BYTE * *tlv*, IN int *tlvLen*)

Set Terminal Data

Sets the Terminal Data for CTLS as specified by the TLV. The first TLV must be Configuration Group Number (Tag FFE4). The terminal global data is group 0, so the first TLV would be FFE40100. Other groups can be defined using this method (1 or greater), and those can be retrieved with emv_getConfigurationGroup(int group), and deleted with emv_removeConfigurationGroup(int group). You cannot delete group 0.

Parameters

tlv	TerminalData configuration file
tlvLen	the length of tlv data buffer

Return values

RETURN_CODE	Values can be parsed with device_getIDGStatusCodeString()

12.12.4.87 int emv_setTerminalMajorConfiguration (IN int *configuration*)

Sets the terminal major configuration in ICS .

Parameters

configuration	A configuration value, range 1-23 • 1 = 1C • 2 = 2C • 3 = 3C • 4 = 4C • 5 = 5C ... • 23 = 23C

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.12.4.88 void emv_setTransactionParameters (IN double *amount,* IN double *amtOther,* IN int *type,* IN int *timeout,* IN BYTE * *tags,* IN int *tagsLen*)

Set EMV Transaction Parameters

Set the parameters to be used on EMV transactions for an ICC card when Auto Poll is on

The tags will be returned in the callback routine.

Parameters

amount	Transaction amount value (tag value 9F02)
amtOther	Other amount value, if any (tag value 9F03)
type	Transaction type (tag value 9C).
timeout	Timeout value in seconds.
tags	Any other tags to be included in the request (Maximum Length = 500 bytes). Passed as a string. Example, tag 9F0C with amount 0x000000000100 would be "9F0C06000000000100" 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. 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 = DFEE1A079F029F36959F37
tagsLen	the length of tags

12.12.4.89 int emv_startTransaction (IN double *amount,* IN double *amtOther,* IN int *exponent,* IN int *type,* IN int *timeout,* IN BYTE * *tags,* IN int *tagsLen,* IN int *forceOnline*)

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) - SEE IMPORTANT NOTE BELOW
amtOther	Other amount value, if any (tag value 9F03) - SEE IMPORTANT NOTE BELOW
exponent	Number of characters after decimal point
type	Transaction type (tag value 9C).
timeout	Timeout value in seconds.
tags	Any other tags to be included in the request. Passed as a TLV stream. Example, tag 9F0C with amount 0x000000000100 would be 0x9F0C06000000000100 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.
tagsLen	Length of tags
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 = DFEE1A079F029F36959F37

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString >>>>>IMPORTANT: parameters for amount and amtOther MUST BE PASSED AS A DOUBLE VALUE WITH DECIMAL POINT. Example, do not pass 1, but instead pass 1.0 or 1.00. Otherwise, results will be unpredictable

12.12.4.90 int icc_exchangeAPDU (IN BYTE * *c_APDU,* IN int *cLen,* OUT BYTE * *reData,* IN_OUT int * *reLen*)

Exchange APDU with plain text For Non-SRED Augusta Only

Sends an APDU packet to the ICC. If successful, response is the APDU data in response parameter.

Parameters

c_APDU	APDU data packet
cLen	APDU data packet length
reData	Unencrypted APDU response
reLen	Unencrypted APDU response data length

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString

12.12.4.91 int icc_getICCReaderStatus (OUT 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_getIDGStatusCodeString

12.12.4.92 int icc_powerOffICC ( )

Power Off ICC

Powers down the ICC

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

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

12.12.4.93 int icc_powerOnICC (OUT BYTE * *ATR,* IN_OUT int * *inLen*)

Power On ICC

Power up the currently selected microprocessor card in the ICC reader

Parameters

ATR,the	ATR data response when succeeded power on ICC,
inLen,the	length of ATR data

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.12.4.94 int msr_cancelMSRSwipe ( )

Disable MSR Swipe Cancels MSR swipe request.

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.12.4.95 void msr_registerCallBk (pMSR_callBack *pMSRf*)

To register the msr callback function to get the MSR card data. (Pass NULL to disable the callback.)

12.12.4.96 void msr_registerCallBkp (pMSR_callBackp *pMSRf*)

To register the msr callback function to get the MSR card data pointer. (Pass NULL to disable the callback.)

12.12.4.97 int msr_startMSRSwipe (IN int *_timeout*)

Start MSR Swipe Enables MSR, waiting for swipe to occur. Allows track selection. Returns IDTMSRData instance to deviceDelegate::swipeMSRData:()

Parameters

timeout	Swipe Timeout Value

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString() Note: if auto poll mode is on, it will return command not allowed error

12.12.4.98 void parseMSRData (IN BYTE * *resData,* IN int *resLen,* IN_OUT IDTMSRData * *cardData*)

Parser the MSR data from the buffer into IDTMSTData structure

Parameters

resData	MSR card data buffer
resLen	the length of resData
cardData	the parser result with IDTMSTData structure

12.12.4.99 void pin_registerCallBk (pPIN_callBack *pPINf*)

To register the pin callback function to get the PINPad data. (Pass NULL to disable the callback.)

12.12.4.100 void registerHotplugCallBk (pMessageHotplug *pMsgHotplug*)

To register the USB HID hot-plug callback function which implemented in the application to monitor the hotplug message from the SDK.

12.12.4.101 void registerLogCallBk (pSendDataLog *pFSend,* pReadDataLog *pFRead*)

To register the log callback function which implemented in the application to monitor sending/reading data between application and reader.

12.12.4.102 char* SDK_Version ( )

To Get SDK version

Returns

return the SDK version string

12.12.4.103 int setAbsoluteLibraryPath (const char * *absoluteLibraryPath*)

Set the path to use when searching for ID TECH's libraries. If this is not set, the libraries will be searched for with the system's default procedures.

Parameters

absoluteLibrary-Path	The absolute path to ID TECH's libraries.

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.13 Source_C/libIDT_VP3300_BT.h File Reference

VP3300 BT API.

```
#include "IDTDef.h"
```

Macros

- #define IN
- #define OUT
- #define IN_OUT

Typedefs

- typedef void(* pMessageHotplug)(int, int)
- typedef void(* pSendDataLog)(unsigned char *, int)
- typedef void(* pReadDataLog)(unsigned char *, int)
- typedef void(* pEMV_callBack)(int, int, unsigned char *, int, IDTTransactionData *, EMV_Callback *, int)
- typedef void(* pMSR_callBack)(int, IDTMSRData)
- typedef void(* pMSR_callBackp)(int, IDTMSRData *)
- typedef void(* pPIN_callBack)(int, IDTPINData *)
- typedef void(* pCMR_callBack)(int, IDTCMRData *)
- typedef void(* pCSFS_callBack)(BYTE status)
- typedef void(* ftpComm_callBack)(int, int, int)
- typedef void(* httpComm_callBack)(BYTE *, int)
- typedef void(* v4Comm_callBack)(BYTE, BYTE, BYTE *, int)

Functions

- void registerHotplugCallBk (pMessageHotplug pMsgHotplug)
- void registerLogCallBk (pSendDataLog pFSend, pReadDataLog pFRead)
- void device_registerRKICallBk (pRKI_callBack pRKIf)
- void emv_registerCallBk (pEMV_callBack pEMVf)
- void msr_registerCallBk (pMSR_callBack pMSRf)
- void msr_registerCallBkp (pMSR_callBackp pMSRf)
- void ctls_registerCallBk (pMSR_callBack pCTLSf)
- void ctls_registerCallBkp (pMSR_callBackp pCTLSf)
- void pin_registerCallBk (pPIN_callBack pPINf)
- void device_registerCameraCallBk (pCMR_callBack pCMRf)
- void device_registerCardStatusFrontSwitchCallBk (pCSFS_callBack pCSFSf)
- void comm_registerHTTPCallback (httpComm_callBack cBack)
- void comm_registerV4Callback (v4Comm_callBack cBack)
- char * SDK_Version ()
- int setAbsoluteLibraryPath (const char *absoluteLibraryPath)
- int device_init ()
- int device_setCurrentDevice (int deviceType)
- int device_close ()
- void device_getIDGStatusCodeString (IN int returnCode, OUT char *despcrition)
- int device_isConnected ()
- int device_isAttached (int deviceType)
- int device_getFirmwareVersion (OUT char *firmwareVersion)
- int device_getFirmwareVersion_Len (OUT char *firmwareVersion, IN_OUT int *firmwareVersionLen)
- int device_pingDevice ()
- int device_controlUserInterface (IN BYTE *values)
- int device_getCurrentDeviceType ()
- int device_SendDataCommandNEO (IN int cmd, IN int subCmd, IN BYTE *data, IN int dataLen, OUT BYTE *response, IN_OUT int *respLen)
- int device_enablePassThrough (int enablePassThrough)
- int device_setBurstMode (IN BYTE mode)
- int device_setPollMode (IN BYTE mode)
- int device_setMerchantRecord (int index, int enabled, char *merchantID, char *merchantURL)
- int device_getMerchantRecord (IN int index, OUT BYTE *record)
- int device_getMerchantRecord_Len (IN int index, OUT BYTE *record, IN_OUT int *recordLen)
- int device_getTransactionResults (IDTMSRData *cardData)
- int device_startTransaction (IN double amount, IN double amtOther, IN int type, IN const int _timeout, IN BYTE *tags, IN int tagsLen)
- void device_setTransactionExponent (int exponent)
- int device_activateTransaction (IN const int _timeout, IN BYTE *tags, IN int tagsLen)
- int device_cancelTransaction ()
- int device_getRTCDateTime (IN BYTE *dateTime, IN_OUT int *dateTimeLen)
- int device_setRTCDateTime (IN BYTE *dateTime, IN int dateTimeLen)
- int device_startRKI (const char *caPath)
- int device_getSDKWaitTime ()
- void device_setSDKWaitTime (int waitTime)
- int device_getThreadStackSize ()
- void device_setThreadStackSize (int threadSize)
- int config_getSerialNumber (OUT char *sNumber)
- int config_getSerialNumber_Len (OUT char *sNumber, IN_OUT int *sNumberLen)
- int ctls_startTransaction (IN double amount, IN double amtOther, IN int type, IN const int _timeout, IN BYTE *tags, IN int tagsLen)
- int ctls_activateTransaction (IN const int _timeout, IN BYTE *tags, IN int tagsLen)
- int ctls_cancelTransaction ()

- int ctls_retrieveApplicationData (IN BYTE *AID, IN int AIDLen, OUT BYTE *tlv, IN_OUT int *tlvLen)
- int ctls_setApplicationData (IN BYTE *tlv, IN int tlvLen)
- int ctls_removeApplicationData (IN BYTE *AID, IN int AIDLen)
- int ctls_removeAllApplicationData ()
- int ctls_retrieveAIDList (OUT BYTE *AIDList, IN_OUT int *AIDListLen)
- int ctls_retrieveTerminalData (OUT BYTE *tlv, IN_OUT int *tlvLen)
- int ctls_setTerminalData (IN BYTE *tlv, IN int tlvLen)
- int ctls_retrieveCAPK (IN BYTE *capk, IN int capkLen, OUT BYTE *key, IN_OUT int *keyLen)
- int ctls_setCAPK (IN BYTE *capk, IN int capkLen)
- int ctls_removeCAPK (IN BYTE *capk, IN int capkLen)
- int ctls_removeAllCAPK ()
- int ctls_retrieveCAPKList (OUT BYTE *keys, IN_OUT int *keysLen)
- int ctls_setConfigurationGroup (IN BYTE *tlv, IN int tlvLen)
- int ctls_getConfigurationGroup (IN int group, OUT BYTE *tlv, OUT int *tlvLen)
- int ctls_getAllConfigurationGroups (OUT BYTE *tlv, IN_OUT int *tlvLen)
- int ctls_removeConfigurationGroup (int group)
- void emv_allowFallback (IN int allow)
- void emv_setAutoAuthenticateTransaction (IN int authenticate)
- void emv_setAutoCompleteTransaction (IN int complete)
- int emv_getAutoAuthenticateTransaction ()
- int emv_getAutoCompleteTransaction ()
- void emv_setTransactionParameters (IN double amount, IN double amtOther, IN int type, IN int timeout, IN BYTE *tags, IN int tagsLen)
- int emv_startTransaction (IN double amount, IN double amtOther, IN int exponent, IN int type, IN int timeout, IN BYTE *tags, IN int tagsLen, IN int forceOnline)
- int emv_activateTransaction (IN int timeout, IN BYTE *tags, IN int tagsLen, IN int forceOnline)
- int emv_authenticateTransaction (IN BYTE *updatedTLV, IN int updatedTLVLen)
- int emv_authenticateTransactionWithTimeout (IN int timeout, IN BYTE *updatedTLV, IN int updatedTLVLen)
- int emv_completeTransaction (IN int commError, IN BYTE *authCode, IN int authCodeLen, IN BYTE *iad, IN int iadLen, IN BYTE *tlvScripts, IN int tlvScriptsLen, IN BYTE *tlv, IN int tlvLen)
- int emv_cancelTransaction ()
- int emv_retrieveApplicationData (IN BYTE *AID, IN int AIDLen, OUT BYTE *tlv, IN_OUT int *tlvLen)
- int emv_setApplicationData (IN BYTE *name, IN int nameLen, IN BYTE *tlv, IN int tlvLen)
- int emv_setApplicationDataTLV (IN BYTE *tlv, IN int tlvLen)
- int emv_removeApplicationData (IN BYTE *AID, IN int AIDLen)
- int emv_removeAllApplicationData ()
- int emv_retrieveAIDList (OUT BYTE *AIDList, IN_OUT int *AIDListLen)
- int emv_retrieveTerminalData (OUT BYTE *tlv, IN_OUT int *tlvLen)
- int emv_setTerminalData (IN BYTE *tlv, IN int tlvLen)
- int emv_setTerminalMajorConfiguration (IN int configuration)
- int emv_retrieveCAPK (IN BYTE *capk, IN int capkLen, OUT BYTE *key, IN_OUT int *keyLen)
- int emv_setCAPK (IN BYTE *capk, IN int capkLen)
- int emv_removeCAPK (IN BYTE *capk, IN int capkLen)
- int emv_removeAllCAPK ()
- int emv_retrieveCAPKList (OUT BYTE *keys, IN_OUT int *keysLen)
- int emv_retrieveCRL (OUT BYTE *list, IN_OUT int *lssLen)
- int emv_setCRL (IN BYTE *list, IN int lsLen)
- int emv_removeCRL (IN BYTE *list, IN int lsLen)
- int emv_removeAllCRL ()
- int icc_getICCReaderStatus (OUT BYTE *status)
- int icc_powerOnICC (OUT BYTE *ATR, IN_OUT int *inLen)
- int icc_powerOffICC ()
- int icc_exchangeAPDU (IN BYTE *c_APDU, IN int cLen, OUT BYTE *reData, IN_OUT int *reLen)
- int msr_cancelMSRSwipe ()
- int msr_startMSRSwipe (IN int _timeout)
- void parseMSRData (IN BYTE *resData, IN int resLen, IN_OUT IDTMSRData *cardData)

12.13.1 Detailed Description

VP3300 BT API. VP3300 BT Global API methods.

12.13.2 Macro Definition Documentation

12.13.2.1 #define IN

INPUT parameter.

12.13.2.2 #define IN_OUT

INPUT / OUTPUT PARAMETER.

12.13.2.3 #define OUT

OUTPUT parameter.

12.13.3 Typedef Documentation

12.13.3.1 typedef void(∗ ftpComm_callBack)(int, int, int)

Define the comm callback function to get FTP file transfer status

It should be passed as a parameter in a FTP request, Signature (int, int, int) = response code, current block, total blocks RESPONSE CODES: 100 = FILE DOWNLOAD STARTED 101 = FILE BLOCK XX OF XX RECEIVED 102 = FILE DOWNLOAD COMPLETED 103 = FILE DOWNLOAD TERMINATED PREMATURELY

12.13.3.2 typedef void(∗ httpComm_callBack)(BYTE ∗, int)

Define the comm callback function to get the async url data

It should be registered using the comm_registerHTTPCallback

12.13.3.3 typedef void(∗ pCMR_callBack)(int, IDTCMRData ∗)

Define the camera callback function to get the image data

It should be registered using the device_registerCameraCallBk,

12.13.3.4 typedef void(∗ pCSFS_callBack)(BYTE status)

Define the card status and front switch callback function to get card and front switch status

It should be registered using the device_registerCardStatusFrontSwitchCallBk,

12.13.3.5 typedef void(∗ pEMV_callBack)(int, int, unsigned char ∗, int, IDTTransactionData ∗, EMV_Callback ∗, int)

Define the EMV callback function to get the transaction message/data/result.

It should be registered using the emv_registerCallBk,

12.13.3.6 typedef void(∗ pMessageHotplug)(int, int)

Define the USB hot-plug callback function to monitor the info when plug in/out the reader.

It should be registered using the registerHotplugCallBk, The first integer parameter is device type, and the second integer parameter is either 0: Device Plugged Out or 1: Device Plugged In

12.13.3.7 typedef void(∗ pMSR_callBack)(int, IDTMSRData)

Define the MSR callback function to get the MSR card data

It should be registered using the msr_registerCallBk, this callback function is for backward compatibility

12.13.3.8 typedef void(∗ pMSR_callBackp)(int, IDTMSRData ∗)

Define the MSR callback function to get pointer to the MSR card data

It should be registered using the msr_registerCallBk, this callback function is recommended instead of pMSR_callBack

12.13.3.9 typedef void(∗ pPIN_callBack)(int, IDTPINData ∗)

Define the PINPad callback function to get the input PIN Pad data

It should be registered using the pin_registerCallBk,

12.13.3.10 typedef void(∗ pReadDataLog)(unsigned char ∗, int)

Define the read response callback function to monitor the reading response from the reader.

It should be registered using the registerLogCallBk,

12.13.3.11 typedef void(∗ pSendDataLog)(unsigned char ∗, int)

Define the send command callback function to monitor the sending command into the reader.

It should be registered using the registerLogCallBk,

12.13.3.12 typedef void(∗ v4Comm_callBack)(BYTE, BYTE, BYTE ∗, int)

Define the comm callback function to receive the V4 Protocol packets received by the device from an external source (IP/USB/RS-232) It should be registered using the comm_registerV4Callback, Data callback will contain command, sub-command, and data from V4 packet

12.13.4 Function Documentation

12.13.4.1 void comm_registerHTTPCallback (httpComm_callBack *cBack*)

Register Comm HTTP Async Callback

Parameters

cBack	- HTTP Comm callback

12.13.4.2 void comm_registerV4Callback (v4Comm_callBack *cBack*)

Register External V4 Protocol commands Callback

Parameters

cBack	- V4 Protocol Comm callback

12.13.4.3 int config_getSerialNumber (OUT char * *sNumber*)

DEPRECATED : please use config_getSerialNumber_Len(OUT char* sNumber, IN_OUT int *sNumberLen)

Polls device for Serial Number

Parameters

sNumber	Returns Serial Number; needs to have at least 64 bytes of memory

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString

12.13.4.4 int config_getSerialNumber_Len (OUT char * *sNumber,* IN_OUT int * *sNumberLen*)

Polls device for Serial Number

Parameters

sNumber	Returns Serial Number
sNumberLen	Length of Serial Number

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString

12.13.4.5 int ctls_activateTransaction (IN const int *_timeout,* IN BYTE * *tags,* IN int *tagsLen*)

Start CTLS Transaction Request

Authorizes the CTLS 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 stream. Example, tag 9F0C with amount 0x000000000100 would be 0x9F0C06000000000100
tagsLen	The length of tags data buffer.

>>>>>IMPORTANT: parameters for amount and amtOther MUST BE PASSED AS A DOUBLE VALUE WITH DECIMAL POINT. Example, do not pass 1, but instead pass 1.0 or 1.00. Otherwise, results will be unpredictable

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString() Note: if auto poll is on, it will returm the error IDG_P2_STATUS_CODE_COMMAND_NOT_ALLOWED

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

12.13.4.6 int ctls_cancelTransaction ( )

Cancel EMV Transaction

Cancels the currently executing EMV transaction.

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.13.4.7 int ctls_getAllConfigurationGroups (OUT BYTE * *tlv,* IN_OUT int * *tlvLen*)

Retrieve All Configuration Groups

Returns all the Configuration Groups installed on the terminal for CTLS

Parameters

tlv	The TLV elements data
tlvLen	the length of tlv data buffer.

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.13.4.8 int ctls_getConfigurationGroup (IN int *group*, OUT BYTE * *tlv*, OUT int * *tlvLen*)

Get Configuration Group

Retrieves the Configuration for the specified Group.

Parameters

group	Configuration Group
tlv	return data
tlvLen	the length of tlv data buffer

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.13.4.9 void ctls_registerCallBk (pMSR_callBack *pCTLSf*)

To register the ctls callback function to get the MSR card data. (Pass NULL to disable the callback.)

12.13.4.10 void ctls_registerCallBkp (pMSR_callBackp *pCTLSf*)

To register the ctls callback function to get the MSR card data pointer. (Pass NULL to disable the callback.)

12.13.4.11 int ctls_removeAllApplicationData ( )

Remove All Application Data

Removes all the Application Data

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.13.4.12 int ctls_removeAllCAPK ( )

Remove All Certificate Authority Public Key

Removes all the CAPK

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.13.4.13 int ctls_removeApplicationData (IN BYTE * *AID*, IN int *AIDLen*)

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
AIDLen	the length of AID data buffer

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.13.4.14 int ctls_removeCAPK (IN BYTE * *capk*, IN int *capkLen*)

Remove Certificate Authority Public Key

Removes the CAPK as specified by the RID/Index

Parameters

capk	6 byte CAPK = 5 bytes RID + 1 byte INDEX
capkLen	the length of capk data buffer

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.13.4.15 int 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	Values can be parsed with device_getIDGStatusCodeString()

12.13.4.16 int ctls_retrieveAIDList (OUT BYTE * *AIDList*, IN_OUT int * *AIDListLen*)

Retrieve AID list

Returns all the AID names installed on the terminal for CTLS. .

Parameters

AIDList	array of AID name byte arrays
AIDListLen	the length of AIDList array buffer

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.13.4.17 int ctls_retrieveApplicationData (IN BYTE * *AID*, IN int *AIDLen*, OUT BYTE * *tlv*, IN_OUT int * *tlvLen*)

Retrieve Application Data by AID

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

Parameters

AID	Name of ApplicationID. Must be between 5 and 16 bytes
AIDLen	the length of AID data buffer.
tlv	The TLV elements of the requested AID
tlvLen	the length of tlv data buffer.

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.13.4.18 int ctls_retrieveCAPK (IN BYTE * *capk*, IN int *capkLen*, OUT BYTE * *key*, IN_OUT int * *keyLen*)

Retrieve Certificate Authority Public Key

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

Parameters

capk	6 bytes CAPK = 5 bytes RID + 1 byte Index
capkLen	the length of capk data buffer
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.
keyLen	the length of key data buffer •

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.13.4.19 int ctls_retrieveCAPKList (OUT BYTE * *keys*, IN_OUT int * *keysLen*)

Retrieve the Certificate Authority Public Key list

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

Parameters

keys	[key1][key2]...[keyn], each key 6 bytes where key = 5 bytes RID + 1 byte index
keysLen	the length of keys data buffer

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.13.4.20 int ctls_retrieveTerminalData (OUT BYTE * *tlv*, IN_OUT int * *tlvLen*)

Retrieve Terminal Data

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

Parameters

tlv	Response returned as a TLV
tlvLen	the length of tlv data buffer

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.13.4.21 int ctls_setApplicationData (IN BYTE * *tlv*, IN int *tlvLen*)

Set Application Data by AID

Sets the Application Data for CTLS as specified by TLV data

Parameters

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

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

Parameters

tlvLen	the length of tlv data buffer

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.13.4.22 int ctls_setCAPK (IN BYTE * *capk*, IN int *capkLen*)

Set Certificate Authority Public Key

Sets the CAPK as specified by the CAKey structure

Parameters

capk	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.
capkLen	the length of capk data buffer

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.13.4.23 int ctls_setConfigurationGroup (IN BYTE * *tlv*, IN int *tlvLen*)

Set Configuration Group

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

Parameters

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

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.13.4.24 int ctls_setTerminalData (IN BYTE * *tlv*, IN int *tlvLen*)

Set Terminal Data

Sets the Terminal Data for CTLS as specified by the TLV. The first TLV must be Configuration Group Number (Tag FFE4). The terminal global data is group 0, so the first TLV would be FFE40100. Other groups can be defined using this method (1 or greater), and those can be retrieved with ctls_getConfigurationGroup(int group), and deleted with ctls_removeConfigurationGroup(int group). You cannot delete group 0.

Parameters

tlv	TerminalData configuration file
tlvLen	the length of tlv data buffer

Return values

RETURN_CODE	Values can be parsed with device_getIDGStatusCodeString()

12.13.4.25 int ctls_startTransaction (IN double *amount,* IN double *amtOther,* IN int *type,* IN const int *_timeout,* IN BYTE * *tags,* IN int *tagsLen*)

Start CTLS Transaction Request

Authorizes the CTLS transaction for an ICC card

The tags will be returned in the callback routine.

Parameters

amount	Transaction amount value (tag value 9F02) - SEE IMPORTANT NOTE BELOW
amtOther	Other amount value, if any (tag value 9F03) - SEE IMPORTANT NOTE BELOW
type	Transaction type (tag value 9C).
timeout	Timeout value in seconds.
tags	Any other tags to be included in the request. Passed as TLV stream. Example, tag 9F0C with amount 0x000000000100 would be 0x9F0C06000000000100 If tags 9F02 (amount),9-F03 (other amount), or 9C (transaction type) are included, they will take priority over these values supplied as individual parameters to this method.
tagsLen	The length of tags data buffer.

>>>>>IMPORTANT: parameters for amount and amtOther MUST BE PASSED AS A DOUBLE VALUE WITH DECIMAL POINT. Example, do not pass 1, but instead pass 1.0 or 1.00. Otherwise, results will be unpredictable

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString() Note: if auto poll is on, it will returm the error IDG_P2_STATUS_CODE_COMMAND_NOT_ALLOWED

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

12.13.4.26 int device_activateTransaction (IN const int *_timeout,* IN BYTE * *tags,* IN int *tagsLen*)

Start Transaction Request

Authorizes the transaction for an MSR/CTLS/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 a TLV. Example, tag 9F0C with amount 0x000000000100 would be 0x9F0C06000000000100 Be sure to include 9F02 (amount)and9-C (transaction type).
tagsLen	The length of tags data buffer.

>>>>>IMPORTANT: parameters for amount and amtOther MUST BE PASSED AS A DOUBLE VALUE WITH DECIMAL POINT. Example, do not pass 1, but instead pass 1.0 or 1.00. Otherwise, results will be unpredictable

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString() Note: if auto poll is on, it will returm the error IDG_P2_STATUS_CODE_COMMAND_NOT_ALLOWED

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 device_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

12.13.4.27 int device_cancelTransaction ( )

Disable Transaction Cancel Transaction request.

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.13.4.28 int device_close ( )

Close the device

Returns

RETURN_CODE: 0: success, 0x0A: failed

12.13.4.29 int device_controlUserInterface (IN BYTE * *values*)

Control User Interface

Controls the User Interface: Display, Beep, LED

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

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.13.4.30 int device_enablePassThrough (int *enablePassThrough*)

Start Remote Key Injection

Starts a remote key injection request with IDTech RKI servers. This function is reserved and not implemented.

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

Enable Pass Through

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

Parameters

enablePass-Through	1 = pass through ON, 0 = pass through OFF

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.13.4.31 int device_getCurrentDeviceType ( )

Get current active device type

Returns

: return the device type defined as DEVICE_TYPE in the IDTDef.h

12.13.4.32 int device_getFirmwareVersion (OUT char * *firmwareVersion*)

DEPRECATED : please use device_getFirmwareVersion_Len(OUT char* firmwareVersion, IN_OUT int *firmwareVersionLen)

Polls device for Firmware Version

Parameters

firmwareVersion	Response returned of Firmware Version; needs to have at least 128 bytes of memory

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.13.4.33 int device_getFirmwareVersion_Len (OUT char * *firmwareVersion,* IN_OUT int * *firmwareVersionLen*)

Polls device for Firmware Version

Parameters

firmwareVersion	Response returned of Firmware Version
firmwareVersion-Len	Length of Firmware Version

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.13.4.34 void device_getIDGStatusCodeString (IN int *returnCode,* OUT char * *despcrition*)

Review the return code description.

Parameters

returnCode	the response result.
description	

Return values

the	string for description of response result

- 0: "no error, beginning task";
- 1: "no response from reader";
- 2: "invalid response data";
  - 01: " Incorrect Header Tag";
  - 02: " Unknown Command";
  - 03: " Unknown Sub-Command";
  - 04: " CRC Error in Frame";
  - 05: " Incorrect Parameter";

- 06: " Parameter Not Supported";
- 07: " Mal-formatted Data";
- 08: " Timeout";
- 0A: " Failed / NACK";
- 0B: " Command not Allowed";
- 0C: " Sub-Command not Allowed";
- 0D: " Buffer Overflow (Data Length too large for reader buffer)";
- 0E: " User Interface Event";
- 10: " Need clear firmware(apply in boot loader only)";
- 11: " Communication type not supported, VT-1, burst, etc. Need encrypted firmware (apply in boot loader only)";
- 12: " Secure interface is not functional or is in an intermediate state.";
- 13: " Data field is not mod 8";
- 14: " Pad 0x80 not found where expected";
- 15: " Specified key type is invalid";
- 16: " Could not retrieve key from the SAM (InitSecureComm)";
- 17: " Hash code problem";
- 18: " Could not store the key into the SAM (InstallKey)";
- 19: " Frame is too large";
- 1A: " Unit powered up in authentication state but POS must resend the InitSecureComm command";
- 1B: " The EEPROM may not be initialized because SecCommInterface does not make sense";
- 1C: " Problem encoding APDU Module-Specific Status Codes ";
- 20: " Unsupported Index (ILM) SAM Transceiver error - problem communicating with the SAM (Key Mgr)";
- 21: " Unexpected Sequence Counter in multiple frames for single bitmap (ILM)Length error in data returned from the SAM (Key Mgr) - 22: " Improper bit map (ILM)"; - 23: " Request Online Authorization"; - 24: " ViVOCard3 raw data read successful"; - 25: " Message index not available (ILM) ViVOcomm activate transaction card type (ViVOcomm)"; - 26: " Version Information Mismatch (ILM)"; - 27: " Not sending commands in correct index message index (ILM)"; - 28: " Time out or next expected message not received (ILM)"; - 29: " ILM languages not available for viewing (ILM)"; - 2A: " Other language not supported (ILM)"; - 41: " from 41 to 4F, Module-specific errors for Key Manager";
- 50: " Auto-Switch OK";
- 51: " Auto-Switch failed";
- 70: " Antenna Error 80h Use another card";
- 81: " Insert or swipe card";
- 90: " Data encryption Key does not exist";
- 91: " Data encryption Key KSN exhausted";

12.13.4.35 int device_getMerchantRecord (IN int *index,* OUT BYTE * *record*)

DEPRECATED : please use device_getMerchantRecord_Len(IN int index, OUT BYTE * record, IN_OUT int *recordLen)

Get Merchant Record

Gets the merchant record for the device.

Parameters

index	Merchant Record index, valid values 1-6
record;	needs to have at least 99 bytes of memory response data from reader. Merchant Record Index: 1 byte enabled: 1 byte Merchant ID: 32 bytes Length of Merchant URL: 1 byte Merchant URL: 64 bytes

Returns

success or error code. Values can be parsed with device_getIDGStatusCodeString()

See Also

ErrorCode

12.13.4.36 int device_getMerchantRecord_Len (IN int *index*, OUT BYTE * *record*, IN_OUT int * *recordLen*)

Get Merchant Record

Gets the merchant record for the device.

Parameters

index	Merchant Record index, valid values 1-6
record	response data from reader. Merchant Record Index: 1 byte enabled: 1 byte Merchant ID: 32 bytes Length of Merchant URL: 1 byte Merchant URL: 64 bytes
recordLen	Length of record

Returns

success or error code. Values can be parsed with device_getIDGStatusCodeString()

See Also

ErrorCode

12.13.4.37 int device_getRTCDateTime (IN BYTE * *dateTime*, IN_OUT int * *dateTimeLen*)

get RTC date and time of the device

Parameters

dateTime	<dateTime data=""> is: 6 byte data for YYMMDDHHMMSS in hex. For example 0x171003102547 stands for 2017 Oct 3rd 10:25:47
dateTimeLen	return 6 bytes if successful

Returns

success or error code. Values can be parsed with device_getResponseCodeString

See Also

ErrorCode

12.13.4.38 int device_getSDKWaitTime ( )

Get SDK Wait Time

Get the SDK wait time for transactions

Returns

SDK wait time in seconds

12.13.4.39 int device_getThreadStackSize ( )

Get Thread Stack Size

Get the stack size setting for newly created threads

Returns

Thread Stack Size

12.13.4.40 int device_getTransactionResults (IDTMSRData * *cardData*)

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

Parameters

cardData	The transaction results

Returns

success or error code. Values can be parsed with device_getResponseCodeString

See Also

ErrorCode

12.13.4.41 int device_init ( )

Initial the device by USB

It will detect the device and trying connect.

The connect status can be checked by device_isConnected().

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.13.4.42 int device_isAttached (int *deviceType*)

Check if the device is attached to the USB port The function device_init() must be called before this function.

Parameters

deviceType,the	device type of the USB device

Returns

1 if the device is attached, or 0 if the device is not attached

12.13.4.43 int device_isConnected ( )

Check the device conntected status

Returns

DEVICE_DISCONNECT=0, or DEVICE_CONNECTED = 1

12.13.4.44 int device_pingDevice ( )

Ping Device

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

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.13.4.45 void device_registerCameraCallBk (pCMR_callBack *pCMRf*)

To register the camera callback function to get the image data. (Pass NULL to disable the callback.)

12.13.4.46 void device_registerCardStatusFrontSwitchCallBk (pCSFS_callBack *pCSFSf*)

To register the card status and front switch callback function to get status. (Pass NULL to disable the callback.)

12.13.4.47 void device_registerRKICallBk (pRKI_callBack *pRKIf*)

To register the RKI callback function to get the RKI status. (Pass NULL to disable the callback.)

12.13.4.48 int device_SendDataCommandNEO (IN int *cmd*, IN int *subCmd*, IN BYTE * *data*, IN int *dataLen*, OUT BYTE * *response*, IN_OUT int * *respLen*)

Send a Command to device

Sends a command to the device .

Parameters

cmd	buffer of command to execute.
cmdLen,the	length of the buffer cmd.
data	buffer of IDG command data.
dataLen,the	length of the buffer data.
response	Response data
respLen,the	length of Response data

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString() Send a Command to NEO device

Sends a command to the NEO device .

Parameters

cmd	command to execute.
subCmd,sub	command to execute.
data	buffer of NEO command data.
dataLen,the	length of the buffer data.
response	Response data
respLen,the	length of Response data

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.13.4.49 int device_setBurstMode (IN BYTE *mode*)

Send Burst Mode

Sets the burst mode for the device.

Parameters

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

Returns

success or error code. Values can be parsed with device_getIDGStatusCodeString

See Also

ErrorCode

12.13.4.50 int device_setCurrentDevice (int *deviceType*)

Sets the current device to talk to

The connect status can be checked by device_isConnected().

Parameters

deviceType	Device to connect to
	
	`enum DEVICE_TYPE`
	`{`
	`IDT_DEVICE_UNKNOWN=0,`
	`IDT_DEVICE_AUGUSTA_HID,`
	`IDT_DEVICE_AUGUSTA_KB,`
	`IDT_DEVICE_AUGUSTA_S_HID,`
	`IDT_DEVICE_AUGUSTA_S_KB,`
	`IDT_DEVICE_AUGUSTA_S_TTK_HID,`
	`IDT_DEVICE_SPECTRUM_PRO,`
	`IDT_DEVICE_MINISMART_II,`
	`IDT_DEVICE_L100,`
	`IDT_DEVICE_UNIPAY,`
	`IDT_DEVICE_UNIPAY_I_V,`
	`IDT_DEVICE_VP3300_AJ,`
	`IDT_DEVICE_KIOSK_III,`
	`IDT_DEVICE_KIOSK_III_S,`
	`IDT_DEVICE_PIP_READER,`
	`IDT_DEVICE_VENDI,`
	`IDT_DEVICE_VP3300_USB,`
	`IDT_DEVICE_UNIPAY_I_V_TTK,`
	`IDT_DEVICE_VP3300_BT,`
	`IDT_DEVICE_VP8800,`
	`IDT_DEVICE_SREDKEY2_HID,`
	`IDT_DEVICE_SREDKEY2_KB,`
	`IDT_DEVICE_NEO2,`
	`IDT_DEVICE_MINISMART_II_COM = IDT_DEVICE_NEO2+5,`
	`IDT_DEVICE_SPECTRUM_PRO_COM,`
	`IDT_DEVICE_KIOSK_III_COM,`
	`IDT_DEVICE_KIOSK_III_S_COM,`
	`IDT_DEVICE_VP3300_COM,`
	`IDT_DEVICE_NEO2_COM,`
	`IDT_DEVICE_MAX_DEVICES = IDT_DEVICE_NEO2_COM+5`
	`};`

Returns

RETURN_CODE: 1: success, 0: failed

12.13.4.51 int device_setMerchantRecord (int *index,* int *enabled,* char * *merchantID,* char * *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 device_getIDGStatusCodeString()

12.13.4.52 int device_setPollMode (IN BYTE *mode*)

Set 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 device_getIDGStatusCodeString()

12.13.4.53 int device_setRTCDateTime (IN BYTE * *dateTime*, IN int *dateTimeLen*)

set RTC date and time of the device

Parameters

dateTime	<dateTime data=""> is: 6 byte data for YYMMDDHHMMSS in hex. For example 0x171003102547 stands for 2017 Oct 3rd 10:25:47
dateTimeLen	should be always 6 bytes

Returns

success or error code. Values can be parsed with device_getResponseCodeString

See Also

ErrorCode

12.13.4.54 void device_setSDKWaitTime (int *waitTime*)

Set SDK Wait Time

Set the SDK wait time for transactions

Parameters

waitTime	The SDK wait time for transaction in seconds

12.13.4.55 void device_setThreadStackSize (int *threadSize*)

Set Thread Stack Size

Set the stack size setting for newly created threads

12.13.4.56 void device_setTransactionExponent (int *exponent*)

Sets the transaction exponent to be used with device_startTransaction. Default value is 2

Parameters

exponent,The	exponent to use when calling device_startTransaction

12.13.4.57 int device_startRKI (const char * *caPath*)

Start remote key injection.

Parameters

caPath	The path to ca-certificates.crt

Returns

success or error code.

See Also

ErrorCode

12.13.4.58 int device_startTransaction (IN double *amount,* IN double *amtOther,* IN int *type,* IN const int *_timeout,* IN BYTE * *tags,* IN int *tagsLen*)

Start Transaction Request

Authorizes the transaction for an MSR/CTLS/ICC card

The tags will be returned in the callback routine.

Parameters

amount	Transaction amount value (tag value 9F02) - SEE IMPORTANT NOTE BELOW
amtOther	Other amount value, if any (tag value 9F03) - SEE IMPORTANT NOTE BELOW
type	Transaction type (tag value 9C).
timeout	Timeout value in seconds.
tags	Any other tags to be included in the request. Passed as TLV. Example, tag 9F0C with amount 0x000000000100 would be 0x9F0C06000000000100 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.
tagsLen	The length of tags data buffer.

>>>>>IMPORTANT: parameters for amount and amtOther MUST BE PASSED AS A DOUBLE VALUE WITH DECIMAL POINT. Example, do not pass 1, but instead pass 1.0 or 1.00. Otherwise, results will be unpredictable

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString() Note: if auto poll is on, it will returm the error IDG_P2_STATUS_CODE_COMMAND_NOT_ALLOWED

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 device_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

12.13.4.59 int emv_activateTransaction (IN int *timeout,* IN BYTE * *tags,* IN int *tagsLen,* IN int *forceOnline*)

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	Tags to be included in the request. Passed as a TLV stream. Example, tag 9F0C with amount 0x000000000100 would be 0x9F0C06000000000100
tagsLen	Length of tags
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 = DFEE1A079F029-F36959F37

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString >>>>>IMPORTANT: parameters for amount and amtOther MUST BE PASSED AS A DOUBLE VALUE WITH DECIMAL POINT. Example, do not pass 1, but instead pass 1.0 or 1.00. Otherwise, results will be unpredictable

12.13.4.60 void emv_allowFallback (IN int *allow*)

Allow fallback for EMV transactions. Default is TRUE

Parameters

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

12.13.4.61 int emv_authenticateTransaction (IN BYTE * *updatedTLV,* IN int *updatedTLVLen*)

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.

Parameters

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, 9F37 to be included in response = DFEE1-A079F029F36959F37
updatedTLVLen	

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.13.4.62 int emv_authenticateTransactionWithTimeout (IN int *timeout,* IN BYTE * *updatedTLV,* IN int *updatedTLVLen*)

Authenticate EMV Transaction Request with Timeout

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.

Parameters

timeout	Timeout value in seconds.
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, 9F37 to be included in response = DFEE1-A079F029F36959F37

updatedTLVLen	

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.13.4.63 int emv_cancelTransaction ( )

Cancel EMV Transaction

Cancels the currently executing EMV transaction.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.13.4.64 int emv_completeTransaction (IN int *commError*, IN BYTE * *authCode*, IN int *authCodeLen*, IN BYTE * *iad*, IN int *iadLen*, IN BYTE * *tlvScripts*, IN int *tlvScriptsLen*, IN BYTE * *tlv*, IN int *tlvLen*)

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(1) if host was unreachable, or FALSE(0) 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
authCodeLen	the length of authCode
iad	Issuer Authentication Data, if any. Example 0x11223344556677883030 (tag value 91).
iadLen	the length of iadLen
tlvScripts	71/72 scripts, if any
tlvScriptsLen	the length of tlvScriptsLen
tlv	Additional TLV data to return with transaction results (if any)
tlvLen	the length of tlv

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.13.4.65 int emv_getAutoAuthenticateTransaction ( )

Gets auto authenticate value for EMV transactions.

Returns

RETURN_CODE: TRUE = auto authenticate, FALSE = manually authenticate

12.13.4.66 int emv_getAutoCompleteTransaction ( )

Gets auto complete value for EMV transactions.

Returns

RETURN_CODE: TRUE = auto complete, FALSE = manually complete

12.13.4.67 void emv_registerCallBk (pEMV_callBack *pEMVf*)

To register the emv callback function to get the EMV processing response. (Pass NULL to disable the callback.)

12.13.4.68 int emv_removeAllApplicationData ( )

Remove All Application Data

Removes all the Application Data

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.13.4.69 int emv_removeAllCAPK ( )

Remove All Certificate Authority Public Key

Removes all the CAPK

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.13.4.70 int emv_removeAllCRL ( )

Remove All Certificate Revocation List Entries

Removes all CRLEntry entries

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.13.4.71 int emv_removeApplicationData (IN BYTE * *AID*, IN int *AIDLen*)

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
AIDLen	the length of AID data buffer

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.13.4.72 int emv_removeCAPK (IN BYTE * *capk*, IN int *capkLen*)

Remove Certificate Authority Public Key

Removes the CAPK as specified by the RID/Index

Parameters

capk	6 byte CAPK = 5 bytes RID + 1 byte INDEX
capkLen	the length of capk data buffer

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.13.4.73 int emv_removeCRL (IN BYTE * *list*, IN int *lsLen*)

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
lssLen	the length of list data buffer

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.13.4.74 int emv_retrieveAIDList (OUT BYTE * *AIDList*, IN_OUT int * *AIDListLen*)

Retrieve AID list

Returns all the AID names installed on the terminal for CTLS. .

Parameters

AIDList	array of AID name byte arrays
AIDListLen	the length of AIDList array buffer

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.13.4.75 int emv_retrieveApplicationData (IN BYTE * *AID*, IN int *AIDLen*, OUT BYTE * *tlv*, IN_OUT int * *tlvLen*)

Retrieve Application Data by AID

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

Parameters

AID	Name of ApplicationID. Must be between 5 and 16 bytes
AIDLen	the length of AID data buffer.
tlv	The TLV elements of the requested AID
tlvLen	the length of tlv data buffer.

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.13.4.76 int emv_retrieveCAPK (IN BYTE * *capk,* IN int *capkLen,* OUT BYTE * *key,* IN_OUT int * *keyLen*)

Retrieve Certificate Authority Public Key

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

Parameters

capk	6 bytes CAPK = 5 bytes RID + 1 byte Index
capkLen	the length of capk data buffer
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.
keyLen	the length of key data buffer •

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.13.4.77 int emv_retrieveCAPKList (OUT BYTE * *keys*, IN_OUT int * *keysLen*)

Retrieve the Certificate Authority Public Key list

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

Parameters

keys	[key1][key2]...[keyn], each key 6 bytes where key = 5 bytes RID + 1 byte index
keysLen	the length of keys data buffer

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.13.4.78 int emv_retrieveCRL (OUT BYTE * *list*, IN_OUT int * *lssLen*)

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
lssLen	the length of list data buffer

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.13.4.79 int emv_retrieveTerminalData (OUT BYTE * *tlv*, IN_OUT int * *tlvLen*)

Retrieve Terminal Data

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

Parameters

tlv	Response returned as a TLV
tlvLen	the length of tlv data buffer

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.13.4.80 int emv_setApplicationData (IN BYTE * *name*, IN int *nameLen*, IN BYTE * *tlv*, IN int *tlvLen*)

Set Application Data by AID

Sets the Application Data as specified by the application name and TLV data

Parameters

name	Application name, 10-32 ASCII hex characters representing 5-16 bytes Example "a0000000031010"
nameLen	the length of name data buffer of Application name,
tlv	Application data in TLV format
tlvLen	the length of tlv data buffer

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.13.4.81 int emv_setApplicationDataTLV (IN BYTE * *tlv*, IN int *tlvLen*)

Set Application Data by TLV

Sets the Application Data as specified by the 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) Example valid TLV, for Group #2, AID a0000000035010- : "dfee2d01029f0607a0000000051010ffe10101ffe50110ffe30114ffe20106"
tlvLen	the length of tlv data buffer

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.13.4.82 void emv_setAutoAuthenticateTransaction (IN int *authenticate*)

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

Parameters

authenticate	TRUE = auto authenticate, FALSE = manually authenticate

12.13.4.83 void emv_setAutoCompleteTransaction (IN int *complete*)

Enables complete for EMV transactions. If a emv_authenticateTranaction results in code 0x0004 (go online), then emv_completeTransaction can automatically execute if parameter is set to TRUE

Parameters

complete	TRUE = auto complete, FALSE = manually complete

12.13.4.84 int emv_setCAPK (IN BYTE * *capk,* IN int *capkLen*)

Set Certificate Authority Public Key

Sets the CAPK as specified by the CAKey structure

Parameters

capk	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.

capkLen	the length of capk data buffer

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.13.4.85 int emv_setCRL (IN BYTE * *list*, IN int *lsLen*)

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]
lsLen	the length of list data buffer

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.13.4.86 int emv_setTerminalData (IN BYTE * *tlv*, IN int *tlvLen*)

Set Terminal Data

Sets the Terminal Data for CTLS as specified by the TLV. The first TLV must be Configuration Group Number (Tag FFE4). The terminal global data is group 0, so the first TLV would be FFE40100. Other groups can be defined using this method (1 or greater), and those can be retrieved with emv_getConfigurationGroup(int group), and deleted with emv_removeConfigurationGroup(int group). You cannot delete group 0.

Parameters

tlv	TerminalData configuration file
tlvLen	the length of tlv data buffer

Return values

RETURN_CODE	Values can be parsed with device_getIDGStatusCodeString()

12.13.4.87 int emv_setTerminalMajorConfiguration (IN int *configuration*)

Sets the terminal major configuration in ICS .

Parameters

configuration	A configuration value, range 1-23 • 1 = 1C • 2 = 2C • 3 = 3C • 4 = 4C • 5 = 5C ... • 23 = 23C

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.13.4.88 void emv_setTransactionParameters (IN double *amount,* IN double *amtOther,* IN int *type,* IN int *timeout,* IN BYTE * *tags,* IN int *tagsLen*)

Set EMV Transaction Parameters

Set the parameters to be used on EMV transactions for an ICC card when Auto Poll is on

The tags will be returned in the callback routine.

Parameters

amount	Transaction amount value (tag value 9F02)
amtOther	Other amount value, if any (tag value 9F03)
type	Transaction type (tag value 9C).
timeout	Timeout value in seconds.
tags	Any other tags to be included in the request (Maximum Length = 500 bytes). Passed as a string. Example, tag 9F0C with amount 0x000000000100 would be "9F0C06000000000100" 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. 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 = DFEE1A079F029F36959F37
tagsLen	the length of tags

12.13.4.89 int emv_startTransaction (IN double *amount,* IN double *amtOther,* IN int *exponent,* IN int *type,* IN int *timeout,* IN BYTE * *tags,* IN int *tagsLen,* IN int *forceOnline*)

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) - SEE IMPORTANT NOTE BELOW
amtOther	Other amount value, if any (tag value 9F03) - SEE IMPORTANT NOTE BELOW
exponent	Number of characters after decimal point
type	Transaction type (tag value 9C).
timeout	Timeout value in seconds.
tags	Any other tags to be included in the request. Passed as a TLV stream. Example, tag 9F0C with amount 0x000000000100 would be 0x9F0C06000000000100 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.
tagsLen	Length of tags
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 = DFEE1A079F029F36959F37

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString >>>>>IMPORTANT: parameters for amount and amtOther MUST BE PASSED AS A DOUBLE VALUE WITH DECIMAL POINT. Example, do not pass 1, but instead pass 1.0 or 1.00. Otherwise, results will be unpredictable

12.13.4.90 int icc_exchangeAPDU (IN BYTE * *c_APDU,* IN int *cLen,* OUT BYTE * *reData,* IN_OUT int * *reLen*)

Exchange APDU with plain text For Non-SRED Augusta Only

Sends an APDU packet to the ICC. If successful, response is the APDU data in response parameter.

Parameters

c_APDU	APDU data packet
cLen	APDU data packet length
reData	Unencrypted APDU response
reLen	Unencrypted APDU response data length

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString

12.13.4.91 int icc_getICCReaderStatus (OUT 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_getIDGStatusCodeString

12.13.4.92 int icc_powerOffICC ( )

Power Off ICC

Powers down the ICC

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

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

12.13.4.93 int icc_powerOnICC (OUT BYTE * *ATR,* IN_OUT int * *inLen*)

Power On ICC

Power up the currently selected microprocessor card in the ICC reader

Parameters

ATR,the	ATR data response when succeeded power on ICC,
inLen,the	length of ATR data

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.13.4.94 int msr_cancelMSRSwipe ( )

Disable MSR Swipe Cancels MSR swipe request.

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.13.4.95 void msr_registerCallBk (pMSR_callBack *pMSRf*)

To register the msr callback function to get the MSR card data. (Pass NULL to disable the callback.)

12.13.4.96 void msr_registerCallBkp (pMSR_callBackp *pMSRf*)

To register the msr callback function to get the MSR card data pointer. (Pass NULL to disable the callback.)

12.13.4.97 int msr_startMSRSwipe (IN int *_timeout*)

Start MSR Swipe Enables MSR, waiting for swipe to occur. Allows track selection. Returns IDTMSRData instance to deviceDelegate::swipeMSRData:()

Parameters

timeout	Swipe Timeout Value

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString() Note: if auto poll mode is on, it will return command not allowed error

12.13.4.98 void parseMSRData (IN BYTE * *resData,* IN int *resLen,* IN_OUT IDTMSRData * *cardData*)

Parser the MSR data from the buffer into IDTMSTData structure

Parameters

resData	MSR card data buffer
resLen	the length of resData
cardData	the parser result with IDTMSTData structure

12.13.4.99 void pin_registerCallBk (pPIN_callBack *pPINf*)

To register the pin callback function to get the PINPad data. (Pass NULL to disable the callback.)

12.13.4.100 void registerHotplugCallBk (pMessageHotplug *pMsgHotplug*)

To register the USB HID hot-plug callback function which implemented in the application to monitor the hotplug message from the SDK.

12.13.4.101 void registerLogCallBk (pSendDataLog *pFSend,* pReadDataLog *pFRead*)

To register the log callback function which implemented in the application to monitor sending/reading data between application and reader.

12.13.4.102 char* SDK_Version ( )

To Get SDK version

Returns

return the SDK version string

12.13.4.103 int setAbsoluteLibraryPath (const char * *absoluteLibraryPath*)

Set the path to use when searching for ID TECH's libraries. If this is not set, the libraries will be searched for with the system's default procedures.

Parameters

absoluteLibrary-Path	The absolute path to ID TECH's libraries.

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.14 Source_C/libIDT_VP3300_COM.h File Reference

VP3300 COM API.

```
#include "IDTDef.h"
```

Macros

- #define IN
- #define OUT
- #define IN_OUT

Typedefs

- typedef void(* pMessageHotplug)(int, int)
- typedef void(* pSendDataLog)(unsigned char *, int)
- typedef void(* pReadDataLog)(unsigned char *, int)
- typedef void(* pEMV_callBack)(int, int, unsigned char *, int, IDTTransactionData *, EMV_Callback *, int)
- typedef void(* pMSR_callBack)(int, IDTMSRData)
- typedef void(* pMSR_callBackp)(int, IDTMSRData *)
- typedef void(* pPIN_callBack)(int, IDTPINData *)
- typedef void(* pCMR_callBack)(int, IDTCMRData *)
- typedef void(* pCSFS_callBack)(BYTE status)
- typedef void(* ftpComm_callBack)(int, int, int)
- typedef void(* httpComm_callBack)(BYTE *, int)
- typedef void(* v4Comm_callBack)(BYTE, BYTE, BYTE *, int)

Functions

- void registerHotplugCallBk (pMessageHotplug pMsgHotplug)
- void registerLogCallBk (pSendDataLog pFSend, pReadDataLog pFRead)
- void device_registerRKICallBk (pRKI_callBack pRKIf)
- void emv_registerCallBk (pEMV_callBack pEMVf)
- void msr_registerCallBk (pMSR_callBack pMSRf)
- void msr_registerCallBkp (pMSR_callBackp pMSRf)
- void ctls_registerCallBk (pMSR_callBack pCTLSf)
- void ctls_registerCallBkp (pMSR_callBackp pCTLSf)
- void pin_registerCallBk (pPIN_callBack pPINf)
- void device_registerCameraCallBk (pCMR_callBack pCMRf)
- void device_registerCardStatusFrontSwitchCallBk (pCSFS_callBack pCSFSf)
- void comm_registerHTTPCallback (httpComm_callBack cBack)
- void comm_registerV4Callback (v4Comm_callBack cBack)
- char * SDK_Version ()
- int setAbsoluteLibraryPath (const char *absoluteLibraryPath)
- int device_init ()
- int device_setCurrentDevice (int deviceType)
- int device_isAttached (int deviceType)
- int device_close ()
- void device_getIDGStatusCodeString (IN int returnCode, OUT char *despcrition)
- int device_isConnected ()
- int device_getFirmwareVersion (OUT char *firmwareVersion)
- int device_getFirmwareVersion_Len (OUT char *firmwareVersion, IN_OUT int *firmwareVersionLen)
- int device_pingDevice ()
- int device_controlUserInterface (IN BYTE *values)
- int device_getCurrentDeviceType ()
- int device_SendDataCommandNEO (IN int cmd, IN int subCmd, IN BYTE *data, IN int dataLen, OUT BYTE *response, IN_OUT int *respLen)
- int device_enablePassThrough (int enablePassThrough)
- int device_setBurstMode (IN BYTE mode)
- int device_setPollMode (IN BYTE mode)
- int device_setMerchantRecord (int index, int enabled, char *merchantID, char *merchantURL)
- int device_getMerchantRecord (IN int index, OUT BYTE *record)
- int device_getMerchantRecord_Len (IN int index, OUT BYTE *record, IN_OUT int *recordLen)
- int device_getTransactionResults (IDTMSRData *cardData)
- int device_startTransaction (IN double amount, IN double amtOther, IN int type, IN const int _timeout, IN BYTE *tags, IN int tagsLen)
- void device_setTransactionExponent (int exponent)
- int device_activateTransaction (IN const int _timeout, IN BYTE *tags, IN int tagsLen)
- int device_cancelTransaction ()
- int device_getRTCDateTime (IN BYTE *dateTime, IN_OUT int *dateTimeLen)
- int device_setRTCDateTime (IN BYTE *dateTime, IN int dateTimeLen)
- int device_startRKI ()
- int device_getSDKWaitTime ()
- void device_setSDKWaitTime (int waitTime)
- int device_getThreadStackSize ()
- void device_setThreadStackSize (int threadSize)
- int config_getSerialNumber (OUT char *sNumber)
- int config_getSerialNumber_Len (OUT char *sNumber, IN_OUT int *sNumberLen)
- int ctls_startTransaction (IN double amount, IN double amtOther, IN int type, IN const int _timeout, IN BYTE *tags, IN int tagsLen)
- int ctls_activateTransaction (IN const int _timeout, IN BYTE *tags, IN int tagsLen)
- int ctls_cancelTransaction ()

- int ctls_retrieveApplicationData (IN BYTE *AID, IN int AIDLen, OUT BYTE *tlv, IN_OUT int *tlvLen)
- int ctls_setApplicationData (IN BYTE *tlv, IN int tlvLen)
- int ctls_removeApplicationData (IN BYTE *AID, IN int AIDLen)
- int ctls_removeAllApplicationData ()
- int ctls_retrieveAIDList (OUT BYTE *AIDList, IN_OUT int *AIDListLen)
- int ctls_retrieveTerminalData (OUT BYTE *tlv, IN_OUT int *tlvLen)
- int ctls_setTerminalData (IN BYTE *tlv, IN int tlvLen)
- int ctls_retrieveCAPK (IN BYTE *capk, IN int capkLen, OUT BYTE *key, IN_OUT int *keyLen)
- int ctls_setCAPK (IN BYTE *capk, IN int capkLen)
- int ctls_removeCAPK (IN BYTE *capk, IN int capkLen)
- int ctls_removeAllCAPK ()
- int ctls_retrieveCAPKList (OUT BYTE *keys, IN_OUT int *keysLen)
- int ctls_setConfigurationGroup (IN BYTE *tlv, IN int tlvLen)
- int ctls_getConfigurationGroup (IN int group, OUT BYTE *tlv, OUT int *tlvLen)
- int ctls_getAllConfigurationGroups (OUT BYTE *tlv, IN_OUT int *tlvLen)
- int ctls_removeConfigurationGroup (int group)
- void emv_allowFallback (IN int allow)
- void emv_setAutoAuthenticateTransaction (IN int authenticate)
- void emv_setAutoCompleteTransaction (IN int complete)
- int emv_getAutoAuthenticateTransaction ()
- int emv_getAutoCompleteTransaction ()
- void emv_setTransactionParameters (IN double amount, IN double amtOther, IN int type, IN int timeout, IN BYTE *tags, IN int tagsLen)
- int emv_startTransaction (IN double amount, IN double amtOther, IN int exponent, IN int type, IN int timeout, IN BYTE *tags, IN int tagsLen, IN int forceOnline)
- int emv_activateTransaction (IN int timeout, IN BYTE *tags, IN int tagsLen, IN int forceOnline)
- int emv_authenticateTransaction (IN BYTE *updatedTLV, IN int updatedTLVLen)
- int emv_authenticateTransactionWithTimeout (IN int timeout, IN BYTE *updatedTLV, IN int updatedTLVLen)
- int emv_completeTransaction (IN int commError, IN BYTE *authCode, IN int authCodeLen, IN BYTE *iad, IN int iadLen, IN BYTE *tlvScripts, IN int tlvScriptsLen, IN BYTE *tlv, IN int tlvLen)
- int emv_cancelTransaction ()
- int emv_retrieveApplicationData (IN BYTE *AID, IN int AIDLen, OUT BYTE *tlv, IN_OUT int *tlvLen)
- int emv_setApplicationData (IN BYTE *name, IN int nameLen, IN BYTE *tlv, IN int tlvLen)
- int emv_setApplicationDataTLV (IN BYTE *tlv, IN int tlvLen)
- int emv_removeApplicationData (IN BYTE *AID, IN int AIDLen)
- int emv_removeAllApplicationData ()
- int emv_retrieveAIDList (OUT BYTE *AIDList, IN_OUT int *AIDListLen)
- int emv_retrieveTerminalData (OUT BYTE *tlv, IN_OUT int *tlvLen)
- int emv_setTerminalData (IN BYTE *tlv, IN int tlvLen)
- int emv_setTerminalMajorConfiguration (IN int configuration)
- int emv_retrieveCAPK (IN BYTE *capk, IN int capkLen, OUT BYTE *key, IN_OUT int *keyLen)
- int emv_setCAPK (IN BYTE *capk, IN int capkLen)
- int emv_removeCAPK (IN BYTE *capk, IN int capkLen)
- int emv_removeAllCAPK ()
- int emv_retrieveCAPKList (OUT BYTE *keys, IN_OUT int *keysLen)
- int emv_retrieveCRL (OUT BYTE *list, IN_OUT int *lssLen)
- int emv_setCRL (IN BYTE *list, IN int lsLen)
- int emv_removeCRL (IN BYTE *list, IN int lsLen)
- int emv_removeAllCRL ()
- int icc_getICCReaderStatus (OUT BYTE *status)
- int icc_powerOnICC (OUT BYTE *ATR, IN_OUT int *inLen)
- int icc_powerOffICC ()
- int icc_exchangeAPDU (IN BYTE *c_APDU, IN int cLen, OUT BYTE *reData, IN_OUT int *reLen)
- int msr_cancelMSRSwipe ()
- int msr_startMSRSwipe (IN int _timeout)
- void parseMSRData (IN BYTE *resData, IN int resLen, IN_OUT IDTMSRData *cardData)

12.14.1 Detailed Description

VP3300 COM API. VP3300 COM Global API methods.

12.14.2 Macro Definition Documentation

12.14.2.1 #define IN

INPUT parameter.

12.14.2.2 #define IN_OUT

INPUT / OUTPUT PARAMETER.

12.14.2.3 #define OUT

OUTPUT parameter.

12.14.3 Typedef Documentation

12.14.3.1 typedef void(∗ ftpComm_callBack)(int, int, int)

Define the comm callback function to get FTP file transfer status

It should be passed as a parameter in a FTP request, Signature (int, int, int) = response code, current block, total blocks RESPONSE CODES: 100 = FILE DOWNLOAD STARTED 101 = FILE BLOCK XX OF XX RECEIVED 102 = FILE DOWNLOAD COMPLETED 103 = FILE DOWNLOAD TERMINATED PREMATURELY

12.14.3.2 typedef void(∗ httpComm_callBack)(BYTE ∗, int)

Define the comm callback function to get the async url data

It should be registered using the comm_registerHTTPCallback

12.14.3.3 typedef void(∗ pCMR_callBack)(int, IDTCMRData ∗)

Define the camera callback function to get the image data

It should be registered using the device_registerCameraCallBk,

12.14.3.4 typedef void(∗ pCSFS_callBack)(BYTE status)

Define the card status and front switch callback function to get card and front switch status

It should be registered using the device_registerCardStatusFrontSwitchCallBk,

12.14.3.5 typedef void(∗ pEMV_callBack)(int, int, unsigned char ∗, int, IDTTransactionData ∗, EMV_Callback ∗, int)

Define the EMV callback function to get the transaction message/data/result.

It should be registered using the emv_registerCallBk,

12.14.3.6 typedef void(∗ pMessageHotplug)(int, int)

Define the USB hot-plug callback function to monitor the info when plug in/out the reader.

It should be registered using the registerHotplugCallBk, The first integer parameter is device type, and the second integer parameter is either 0: Device Plugged Out or 1: Device Plugged In

12.14.3.7 typedef void(∗ pMSR_callBack)(int, IDTMSRData)

Define the MSR callback function to get the MSR card data

It should be registered using the msr_registerCallBk, this callback function is for backward compatibility

12.14.3.8 typedef void(∗ pMSR_callBackp)(int, IDTMSRData ∗)

Define the MSR callback function to get pointer to the MSR card data

It should be registered using the msr_registerCallBk, this callback function is recommended instead of pMSR_callBack

12.14.3.9 typedef void(∗ pPIN_callBack)(int, IDTPINData ∗)

Define the PINPad callback function to get the input PIN Pad data

It should be registered using the pin_registerCallBk,

12.14.3.10 typedef void(∗ pReadDataLog)(unsigned char ∗, int)

Define the read response callback function to monitor the reading response from the reader.

It should be registered using the registerLogCallBk,

12.14.3.11 typedef void(∗ pSendDataLog)(unsigned char ∗, int)

Define the send command callback function to monitor the sending command into the reader.

It should be registered using the registerLogCallBk,

12.14.3.12 typedef void(∗ v4Comm_callBack)(BYTE, BYTE, BYTE ∗, int)

Define the comm callback function to receive the V4 Protocol packets received by the device from an external source (IP/USB/RS-232) It should be registered using the comm_registerV4Callback, Data callback will contain command, sub-command, and data from V4 packet

12.14.4 Function Documentation

12.14.4.1 void comm_registerHTTPCallback (httpComm_callBack *cBack*)

Register Comm HTTP Async Callback

Parameters

cBack	- HTTP Comm callback

12.14.4.2 void comm_registerV4Callback (v4Comm_callBack *cBack*)

Register External V4 Protocol commands Callback

Parameters

cBack	- V4 Protocol Comm callback

12.14.4.3 int config_getSerialNumber (OUT char * *sNumber*)

DEPRECATED : please use config_getSerialNumber_Len(OUT char* sNumber, IN_OUT int *sNumberLen)

Polls device for Serial Number

Parameters

sNumber	Returns Serial Number; needs to have at least 64 bytes of memory

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString

12.14.4.4 int config_getSerialNumber_Len (OUT char * *sNumber,* IN_OUT int * *sNumberLen*)

Polls device for Serial Number

Parameters

sNumber	Returns Serial Number
sNumberLen	Length of Serial Number

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString

12.14.4.5 int ctls_activateTransaction (IN const int *_timeout,* IN BYTE * *tags,* IN int *tagsLen*)

Start CTLS Transaction Request

Authorizes the CTLS 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 stream. Example, tag 9F0C with amount 0x000000000100 would be 0x9F0C06000000000100
tagsLen	The length of tags data buffer.

>>>>>IMPORTANT: parameters for amount and amtOther MUST BE PASSED AS A DOUBLE VALUE WITH DECIMAL POINT. Example, do not pass 1, but instead pass 1.0 or 1.00. Otherwise, results will be unpredictable

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString() Note: if auto poll is on, it will returm the error IDG_P2_STATUS_CODE_COMMAND_NOT_ALLOWED

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

12.14.4.6 int ctls_cancelTransaction ( )

Cancel EMV Transaction

Cancels the currently executing EMV transaction.

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.14.4.7 int ctls_getAllConfigurationGroups (OUT BYTE * *tlv,* IN_OUT int * *tlvLen*)

Retrieve All Configuration Groups

Returns all the Configuration Groups installed on the terminal for CTLS

Parameters

tlv	The TLV elements data
tlvLen	the length of tlv data buffer.

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.14.4.8 int ctls_getConfigurationGroup (IN int *group*, OUT BYTE * *tlv*, OUT int * *tlvLen*)

Get Configuration Group

Retrieves the Configuration for the specified Group.

Parameters

group	Configuration Group
tlv	return data
tlvLen	the length of tlv data buffer

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.14.4.9 void ctls_registerCallBk (pMSR_callBack *pCTLSf*)

To register the ctls callback function to get the MSR card data. (Pass NULL to disable the callback.)

12.14.4.10 void ctls_registerCallBkp (pMSR_callBackp *pCTLSf*)

To register the ctls callback function to get the MSR card data pointer. (Pass NULL to disable the callback.)

12.14.4.11 int ctls_removeAllApplicationData ( )

Remove All Application Data

Removes all the Application Data

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.14.4.12 int ctls_removeAllCAPK ( )

Remove All Certificate Authority Public Key

Removes all the CAPK

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.14.4.13 int ctls_removeApplicationData (IN BYTE * *AID*, IN int *AIDLen*)

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
AIDLen	the length of AID data buffer

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.14.4.14 int ctls_removeCAPK (IN BYTE * *capk*, IN int *capkLen*)

Remove Certificate Authority Public Key

Removes the CAPK as specified by the RID/Index

Parameters

capk	6 byte CAPK = 5 bytes RID + 1 byte INDEX
capkLen	the length of capk data buffer

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.14.4.15 int 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	Values can be parsed with device_getIDGStatusCodeString()

12.14.4.16 int ctls_retrieveAIDList (OUT BYTE * *AIDList*, IN_OUT int * *AIDListLen*)

Retrieve AID list

Returns all the AID names installed on the terminal for CTLS. .

Parameters

AIDList	array of AID name byte arrays
AIDListLen	the length of AIDList array buffer

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.14.4.17 int ctls_retrieveApplicationData (IN BYTE * *AID*, IN int *AIDLen*, OUT BYTE * *tlv*, IN_OUT int * *tlvLen*)

Retrieve Application Data by AID

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

Parameters

AID	Name of ApplicationID. Must be between 5 and 16 bytes
AIDLen	the length of AID data buffer.
tlv	The TLV elements of the requested AID
tlvLen	the length of tlv data buffer.

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.14.4.18 int ctls_retrieveCAPK (IN BYTE * *capk*, IN int *capkLen*, OUT BYTE * *key*, IN_OUT int * *keyLen*)

Retrieve Certificate Authority Public Key

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

Parameters

capk	6 bytes CAPK = 5 bytes RID + 1 byte Index
capkLen	the length of capk data buffer
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.
keyLen	the length of key data buffer •

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.14.4.19 int ctls_retrieveCAPKList (OUT BYTE * *keys*, IN_OUT int * *keysLen*)

Retrieve the Certificate Authority Public Key list

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

Parameters

keys	[key1][key2]...[keyn], each key 6 bytes where key = 5 bytes RID + 1 byte index
keysLen	the length of keys data buffer

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.14.4.20 int ctls_retrieveTerminalData (OUT BYTE * *tlv,* IN_OUT int * *tlvLen*)

Retrieve Terminal Data

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

Parameters

tlv	Response returned as a TLV
tlvLen	the length of tlv data buffer

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.14.4.21 int ctls_setApplicationData (IN BYTE * *tlv,* IN int *tlvLen*)

Set Application Data by AID

Sets the Application Data for CTLS as specified by TLV data

Parameters

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

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

Parameters

tlvLen	the length of tlv data buffer

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.14.4.22 int ctls_setCAPK (IN BYTE * *capk,* IN int *capkLen*)

Set Certificate Authority Public Key

Sets the CAPK as specified by the CAKey structure

Parameters

capk	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.
capkLen	the length of capk data buffer

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.14.4.23 int ctls_setConfigurationGroup (IN BYTE * *tlv*, IN int *tlvLen*)

Set Configuration Group

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

Parameters

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

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.14.4.24 int ctls_setTerminalData (IN BYTE * *tlv*, IN int *tlvLen*)

Set Terminal Data

Sets the Terminal Data for CTLS as specified by the TLV. The first TLV must be Configuration Group Number (Tag FFE4). The terminal global data is group 0, so the first TLV would be FFE40100. Other groups can be defined using this method (1 or greater), and those can be retrieved with ctls_getConfigurationGroup(int group), and deleted with ctls_removeConfigurationGroup(int group). You cannot delete group 0.

Parameters

tlv	TerminalData configuration file
tlvLen	the length of tlv data buffer

Return values

RETURN_CODE	Values can be parsed with device_getIDGStatusCodeString()

12.14.4.25 int ctls_startTransaction (IN double *amount,* IN double *amtOther,* IN int *type,* IN const int *_timeout,* IN BYTE * *tags,* IN int *tagsLen*)

Start CTLS Transaction Request

Authorizes the CTLS transaction for an ICC card

The tags will be returned in the callback routine.

Parameters

amount	Transaction amount value (tag value 9F02) - SEE IMPORTANT NOTE BELOW
amtOther	Other amount value, if any (tag value 9F03) - SEE IMPORTANT NOTE BELOW
type	Transaction type (tag value 9C).
timeout	Timeout value in seconds.
tags	Any other tags to be included in the request. Passed as TLV stream. Example, tag 9F0C with amount 0x000000000100 would be 0x9F0C06000000000100 If tags 9F02 (amount),9-F03 (other amount), or 9C (transaction type) are included, they will take priority over these values supplied as individual parameters to this method.
tagsLen	The length of tags data buffer.

>>>>>IMPORTANT: parameters for amount and amtOther MUST BE PASSED AS A DOUBLE VALUE WITH DECIMAL POINT. Example, do not pass 1, but instead pass 1.0 or 1.00. Otherwise, results will be unpredictable

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString() Note: if auto poll is on, it will returm the error IDG_P2_STATUS_CODE_COMMAND_NOT_ALLOWED

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

12.14.4.26 int device_activateTransaction (IN const int *_timeout,* IN BYTE * *tags,* IN int *tagsLen*)

Start Transaction Request

Authorizes the transaction for an MSR/CTLS/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 a TLV. Example, tag 9F0C with amount 0x000000000100 would be 0x9F0C06000000000100 Be sure to include 9F02 (amount)and9-C (transaction type).
tagsLen	The length of tags data buffer.

>>>>>IMPORTANT: parameters for amount and amtOther MUST BE PASSED AS A DOUBLE VALUE WITH DECIMAL POINT. Example, do not pass 1, but instead pass 1.0 or 1.00. Otherwise, results will be unpredictable

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString() Note: if auto poll is on, it will returm the error IDG_P2_STATUS_CODE_COMMAND_NOT_ALLOWED

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 device_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

12.14.4.27 int device_cancelTransaction ( )

Disable Transaction Cancel Transaction request.

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.14.4.28 int device_close ( )

Close the device

Returns

RETURN_CODE: 0: success, 0x0A: failed

12.14.4.29 int device_controlUserInterface (IN BYTE * *values*)

Control User Interface
Controls the User Interface: Display, Beep, LED

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

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.14.4.30 int device_enablePassThrough (int *enablePassThrough*)

Start Remote Key Injection

Starts a remote key injection request with IDTech RKI servers. This function is reserved and not implemented.

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

Enable Pass Through

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

Parameters

enablePass-Through	1 = pass through ON, 0 = pass through OFF

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.14.4.31 int device_getCurrentDeviceType ( )

Get current active device type

Returns

: return the device type defined as DEVICE_TYPE in the IDTDef.h

12.14.4.32 int device_getFirmwareVersion (OUT char * *firmwareVersion*)

DEPRECATED : please use device_getFirmwareVersion_Len(OUT char* firmwareVersion, IN_OUT int *firmwareVersionLen)

Polls device for Firmware Version

Parameters

firmwareVersion	Response returned of Firmware Version; needs to have at least 128 bytes of memory

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.14.4.33 int device_getFirmwareVersion_Len (OUT char * *firmwareVersion,* IN_OUT int * *firmwareVersionLen*)

Polls device for Firmware Version

Parameters

firmwareVersion	Response returned of Firmware Version
firmwareVersion-Len	Length of Firmware Version

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.14.4.34 void device_getIDGStatusCodeString (IN int *returnCode,* OUT char * *despcrition*)

Review the return code description.

Parameters

returnCode	the response result.
description	

Return values

the	string for description of response result

- 0: "no error, beginning task";
- 1: "no response from reader";
- 2: "invalid response data";
  - 01: " Incorrect Header Tag";
  - 02: " Unknown Command";
  - 03: " Unknown Sub-Command";
  - 04: " CRC Error in Frame";
  - 05: " Incorrect Parameter";

- 06: " Parameter Not Supported";
- 07: " Mal-formatted Data";
- 08: " Timeout";
- 0A: " Failed / NACK";
- 0B: " Command not Allowed";
- 0C: " Sub-Command not Allowed";
- 0D: " Buffer Overflow (Data Length too large for reader buffer)";
- 0E: " User Interface Event";
- 10: " Need clear firmware(apply in boot loader only)";
- 11: " Communication type not supported, VT-1, burst, etc. Need encrypted firmware (apply in boot loader only)";
- 12: " Secure interface is not functional or is in an intermediate state.";
- 13: " Data field is not mod 8";
- 14: " Pad 0x80 not found where expected";
- 15: " Specified key type is invalid";
- 16: " Could not retrieve key from the SAM (InitSecureComm)";
- 17: " Hash code problem";
- 18: " Could not store the key into the SAM (InstallKey)";
- 19: " Frame is too large";
- 1A: " Unit powered up in authentication state but POS must resend the InitSecureComm command";
- 1B: " The EEPROM may not be initialized because SecCommInterface does not make sense";
- 1C: " Problem encoding APDU Module-Specific Status Codes ";
- 20: " Unsupported Index (ILM) SAM Transceiver error - problem communicating with the SAM (Key Mgr)";
- 21: " Unexpected Sequence Counter in multiple frames for single bitmap (ILM)Length error in data returned from the SAM (Key Mgr) - 22: " Improper bit map (ILM)"; - 23: " Request Online Authorization"; - 24: " ViVOCard3 raw data read successful"; - 25: " Message index not available (ILM) ViVOcomm activate transaction card type (ViVOcomm)"; - 26: " Version Information Mismatch (ILM)"; - 27: " Not sending commands in correct index message index (ILM)"; - 28: " Time out or next expected message not received (ILM)"; - 29: " ILM languages not available for viewing (ILM)"; - 2A: " Other language not supported (ILM)"; - 41: " from 41 to 4F, Module-specific errors for Key Manager";
- 50: " Auto-Switch OK";
- 51: " Auto-Switch failed";
- 70: " Antenna Error 80h Use another card";
- 81: " Insert or swipe card";
- 90: " Data encryption Key does not exist";
- 91: " Data encryption Key KSN exhausted";

12.14.4.35 int device_getMerchantRecord (IN int *index,* OUT BYTE * *record*)

DEPRECATED : please use device_getMerchantRecord_Len(IN int index, OUT BYTE * record, IN_OUT int *recordLen)

Get Merchant Record

Gets the merchant record for the device.

Parameters

index	Merchant Record index, valid values 1-6
record;	needs to have at least 99 bytes of memory response data from reader. Merchant Record Index: 1 byte enabled: 1 byte Merchant ID: 32 bytes Length of Merchant URL: 1 byte Merchant URL: 64 bytes

Returns

success or error code. Values can be parsed with device_getIDGStatusCodeString()

See Also

ErrorCode

12.14.4.36 int device_getMerchantRecord_Len (IN int *index*, OUT BYTE * *record*, IN_OUT int * *recordLen*)

Get Merchant Record

Gets the merchant record for the device.

Parameters

index	Merchant Record index, valid values 1-6
record	response data from reader. Merchant Record Index: 1 byte enabled: 1 byte Merchant ID: 32 bytes Length of Merchant URL: 1 byte Merchant URL: 64 bytes
recordLen	Length of record

Returns

success or error code. Values can be parsed with device_getIDGStatusCodeString()

See Also

ErrorCode

12.14.4.37 int device_getRTCDateTime (IN BYTE * *dateTime*, IN_OUT int * *dateTimeLen*)

get RTC date and time of the device

Parameters

dateTime	<dateTime data=""> is: 6 byte data for YYMMDDHHMMSS in hex. For example 0x171003102547 stands for 2017 Oct 3rd 10:25:47
dateTimeLen	return 6 bytes if successful

Returns

success or error code. Values can be parsed with device_getResponseCodeString

See Also

ErrorCode

12.14.4.38 int device_getSDKWaitTime ( )

Get SDK Wait Time

Get the SDK wait time for transactions

Returns

SDK wait time in seconds

12.14.4.39 int device_getThreadStackSize ( )

Get Thread Stack Size

Get the stack size setting for newly created threads

Returns

Thread Stack Size

12.14.4.40 int device_getTransactionResults (IDTMSRData * *cardData*)

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

Parameters

cardData	The transaction results

Returns

success or error code. Values can be parsed with device_getResponseCodeString

See Also

ErrorCode

12.14.4.41 int device_init ( )

Initial the device by USB

It will detect the device and trying connect.

The connect status can be checked by device_isConnected().

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.14.4.42 int device_isAttached (int *deviceType*)

Check if the device is attached to the USB port The function device_init() must be called before this function.

Parameters

deviceType,the	device type of the USB device

Returns

1 if the device is attached, or 0 if the device is not attached

12.14.4.43 int device_isConnected ( )

Check the device conntected status

Returns

DEVICE_DISCONNECT=0, or DEVICE_CONNECTED = 1

12.14.4.44 int device_pingDevice ( )

Ping Device

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

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.14.4.45 void device_registerCameraCallBk (pCMR_callBack *pCMRf*)

To register the camera callback function to get the image data. (Pass NULL to disable the callback.)

12.14.4.46 void device_registerCardStatusFrontSwitchCallBk (pCSFS_callBack *pCSFSf*)

To register the card status and front switch callback function to get status. (Pass NULL to disable the callback.)

12.14.4.47 void device_registerRKICallBk (pRKI_callBack *pRKIf*)

To register the RKI callback function to get the RKI status. (Pass NULL to disable the callback.)

12.14.4.48 int device_SendDataCommandNEO (IN int *cmd,* IN int *subCmd,* IN BYTE * *data,* IN int *dataLen,* OUT BYTE * *response,* IN_OUT int * *respLen*)

Send a Command to device

Sends a command to the device .

Parameters

cmd	buffer of command to execute.
cmdLen,the	length of the buffer cmd.
data	buffer of IDG command data.
dataLen,the	length of the buffer data.
response	Response data
respLen,the	length of Response data

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString() Send a Command to NEO device

Sends a command to the NEO device .

Parameters

cmd	command to execute.
subCmd,sub	command to execute.
data	buffer of NEO command data.
dataLen,the	length of the buffer data.
response	Response data
respLen,the	length of Response data

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.14.4.49 int device_setBurstMode (IN BYTE *mode*)

Send Burst Mode

Sets the burst mode for the device.

Parameters

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

Returns

success or error code. Values can be parsed with device_getIDGStatusCodeString

See Also

ErrorCode

12.14.4.50 int device_setCurrentDevice (int *deviceType*)

Sets the current device to talk to

The connect status can be checked by device_isConnected().

Parameters

deviceType	Device to connect to
	
	`enum DEVICE_TYPE`
	`{`
	`IDT_DEVICE_UNKNOWN=0,`
	`IDT_DEVICE_AUGUSTA_HID,`
	`IDT_DEVICE_AUGUSTA_KB,`
	`IDT_DEVICE_AUGUSTA_S_HID,`
	`IDT_DEVICE_AUGUSTA_S_KB,`
	`IDT_DEVICE_AUGUSTA_S_TTK_HID,`
	`IDT_DEVICE_SPECTRUM_PRO,`
	`IDT_DEVICE_MINISMART_II,`
	`IDT_DEVICE_L100,`
	`IDT_DEVICE_UNIPAY,`
	`IDT_DEVICE_UNIPAY_I_V,`
	`IDT_DEVICE_VP3300_AJ,`
	`IDT_DEVICE_KIOSK_III,`
	`IDT_DEVICE_KIOSK_III_S,`
	`IDT_DEVICE_PIP_READER,`
	`IDT_DEVICE_VENDI,`
	`IDT_DEVICE_VP3300_USB,`
	`IDT_DEVICE_UNIPAY_I_V_TTK,`
	`IDT_DEVICE_VP3300_BT,`
	`IDT_DEVICE_VP8800,`
	`IDT_DEVICE_SREDKEY2_HID,`
	`IDT_DEVICE_SREDKEY2_KB,`
	`IDT_DEVICE_NEO2,`
	`IDT_DEVICE_MINISMART_II_COM = IDT_DEVICE_NEO2+5,`
	`IDT_DEVICE_SPECTRUM_PRO_COM,`
	`IDT_DEVICE_KIOSK_III_COM,`
	`IDT_DEVICE_KIOSK_III_S_COM,`
	`IDT_DEVICE_VP3300_COM,`
	`IDT_DEVICE_NEO2_COM,`
	`IDT_DEVICE_MAX_DEVICES = IDT_DEVICE_NEO2_COM+5`
	`};`

Returns

RETURN_CODE: 1: success, 0: failed

12.14.4.51 int device_setMerchantRecord (int *index,* int *enabled,* char * *merchantID,* char * *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 device_getIDGStatusCodeString()

12.14.4.52 int device_setPollMode (IN BYTE *mode*)

Set 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 device_getIDGStatusCodeString()

12.14.4.53 int device_setRTCDateTime (IN BYTE * *dateTime*, IN int *dateTimeLen*)

set RTC date and time of the device

Parameters

dateTime	<dateTime data=""> is: 6 byte data for YYMMDDHHMMSS in hex. For example 0x171003102547 stands for 2017 Oct 3rd 10:25:47
dateTimeLen	should be always 6 bytes

Returns

success or error code. Values can be parsed with device_getResponseCodeString

See Also

ErrorCode

12.14.4.54 void device_setSDKWaitTime (int *waitTime*)

Set SDK Wait Time

Set the SDK wait time for transactions

Parameters

waitTime	The SDK wait time for transaction in seconds

12.14.4.55 void device_setThreadStackSize (int *threadSize*)

Set Thread Stack Size

Set the stack size setting for newly created threads

12.14.4.56 void device_setTransactionExponent (int *exponent*)

Sets the transaction exponent to be used with device_startTransaction. Default value is 2

Parameters

exponent,The	exponent to use when calling device_startTransaction

12.14.4.57 int device_startRKI ( )

Start remote key injection.

Returns

success or error code.

See Also

ErrorCode

12.14.4.58 int device_startTransaction (IN double *amount,* IN double *amtOther,* IN int *type,* IN const int *_timeout,* IN BYTE * *tags,* IN int *tagsLen*)

Start device Transaction Request

Authorizes the transaction for an MSR/CTLS/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)
type	Transaction type (tag value 9C).
timeout	Timeout value in seconds.
tags	Any other tags to be included in the request. Passed as TLV. Example, tag 9F0C with amount 0x000000000100 would be 0x9F0C06000000000100 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. 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 = DFEE1A079F029F36959F37

Returns

success or error code. Values can be parsed with device_getResponseCodeString

See Also

ErrorCode Note: if auto poll is on, it will returm the error IDG_P2_STATUS_CODE_COMMAND_NOT_ALLOWED

12.14.4.59 int emv_activateTransaction (IN int *timeout,* IN BYTE * *tags,* IN int *tagsLen,* IN int *forceOnline*)

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	Tags to be included in the request. Passed as a TLV stream. Example, tag 9F0C with amount 0x000000000100 would be 0x9F0C06000000000100
tagsLen	Length of tags
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 = DFEE1A079F029F36959F37

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString >>>>>IMPORTANT: parameters for amount and amtOther MUST BE PASSED AS A DOUBLE VALUE WITH DECIMAL POINT. Example, do not pass 1, but instead pass 1.0 or 1.00. Otherwise, results will be unpredictable

12.14.4.60 void emv_allowFallback (IN int *allow*)

Allow fallback for EMV transactions. Default is TRUE

Parameters

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

12.14.4.61 int emv_authenticateTransaction (IN BYTE * *updatedTLV,* IN int *updatedTLVLen*)

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.

Parameters

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, 9F37 to be included in response = DFEE1-A079F029F36959F37
updatedTLVLen	

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.14.4.62 int emv_authenticateTransactionWithTimeout (IN int *timeout,* IN BYTE * *updatedTLV,* IN int *updatedTLVLen*)

Authenticate EMV Transaction Request with Timeout

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.

Parameters

timeout	Timeout value in seconds.
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, 9F37 to be included in response = DFEE1-A079F029F36959F37
updatedTLVLen	

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.14.4.63 int emv_cancelTransaction ( )

Cancel EMV Transaction

Cancels the currently executing EMV transaction.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.14.4.64 int emv_completeTransaction (IN int *commError*, IN BYTE * *authCode*, IN int *authCodeLen*, IN BYTE * *iad*, IN int *iadLen*, IN BYTE * *tlvScripts*, IN int *tlvScriptsLen*, IN BYTE * *tlv*, IN int *tlvLen*)

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(1) if host was unreachable, or FALSE(0) 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
authCodeLen	the length of authCode
iad	Issuer Authentication Data, if any. Example 0x11223344556677883030 (tag value 91).
iadLen	the length of iadLen
tlvScripts	71/72 scripts, if any
tlvScriptsLen	the length of tlvScriptsLen
tlv	Additional TLV data to return with transaction results (if any)
tlvLen	the length of tlv

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.14.4.65 int emv_getAutoAuthenticateTransaction ( )

Gets auto authenticate value for EMV transactions.

Returns

RETURN_CODE: TRUE = auto authenticate, FALSE = manually authenticate

12.14.4.66 int emv_getAutoCompleteTransaction ( )

Gets auto complete value for EMV transactions.

Returns

RETURN_CODE: TRUE = auto complete, FALSE = manually complete

12.14.4.67 void emv_registerCallBk (pEMV_callBack *pEMVf*)

To register the emv callback function to get the EMV processing response. (Pass NULL to disable the callback.)

12.14.4.68 int emv_removeAllApplicationData ( )

Remove All Application Data

Removes all the Application Data

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.14.4.69 int emv_removeAllCAPK ( )

Remove All Certificate Authority Public Key

Removes all the CAPK

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.14.4.70 int emv_removeAllCRL ( )

Remove All Certificate Revocation List Entries

Removes all CRLEntry entries

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.14.4.71 int emv_removeApplicationData (IN BYTE * *AID,* IN int *AIDLen*)

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
AIDLen	the length of AID data buffer

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.14.4.72 int emv_removeCAPK (IN BYTE * *capk*, IN int *capkLen*)

Remove Certificate Authority Public Key

Removes the CAPK as specified by the RID/Index

Parameters

capk	6 byte CAPK = 5 bytes RID + 1 byte INDEX
capkLen	the length of capk data buffer

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.14.4.73 int emv_removeCRL (IN BYTE * *list*, IN int *lsLen*)

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
lssLen	the length of list data buffer

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.14.4.74 int emv_retrieveAIDList (OUT BYTE * *AIDList*, IN_OUT int * *AIDListLen*)

Retrieve AID list

Returns all the AID names installed on the terminal for CTLS. .

Parameters

AIDList	array of AID name byte arrays
AIDListLen	the length of AIDList array buffer

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.14.4.75 int emv_retrieveApplicationData (IN BYTE * *AID*, IN int *AIDLen*, OUT BYTE * *tlv*, IN_OUT int * *tlvLen*)

Retrieve Application Data by AID

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

Parameters

AID	Name of ApplicationID. Must be between 5 and 16 bytes
AIDLen	the length of AID data buffer.
tlv	The TLV elements of the requested AID
tlvLen	the length of tlv data buffer.

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.14.4.76 int emv_retrieveCAPK (IN BYTE * *capk*, IN int *capkLen*, OUT BYTE * *key*, IN_OUT int * *keyLen*)

Retrieve Certificate Authority Public Key

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

Parameters

capk	6 bytes CAPK = 5 bytes RID + 1 byte Index
capkLen	the length of capk data buffer
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.

keyLen	the length of key data buffer •

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.14.4.77 int emv_retrieveCAPKList (OUT BYTE * *keys*, IN_OUT int * *keysLen*)

Retrieve the Certificate Authority Public Key list

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

Parameters

keys	[key1][key2]...[keyn], each key 6 bytes where key = 5 bytes RID + 1 byte index
keysLen	the length of keys data buffer

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.14.4.78 int emv_retrieveCRL (OUT BYTE * *list*, IN_OUT int * *lssLen*)

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
lssLen	the length of list data buffer

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.14.4.79 int emv_retrieveTerminalData (OUT BYTE * *tlv*, IN_OUT int * *tlvLen*)

Retrieve Terminal Data

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

Parameters

tlv	Response returned as a TLV
tlvLen	the length of tlv data buffer

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.14.4.80 int emv_setApplicationData (IN BYTE * *name,* IN int *nameLen,* IN BYTE * *tlv,* IN int *tlvLen*)

Set Application Data by AID

Sets the Application Data as specified by the application name and TLV data

Parameters

name	Application name, 10-32 ASCII hex characters representing 5-16 bytes Example "a0000000031010"
nameLen	the length of name data buffer of Application name,
tlv	Application data in TLV format
tlvLen	the length of tlv data buffer

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.14.4.81 int emv_setApplicationDataTLV (IN BYTE * *tlv,* IN int *tlvLen*)

Set Application Data by TLV

Sets the Application Data as specified by the 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) Example valid TLV, for Group #2, AID a0000000035010- : "dfee2d01029f0607a0000000051010ffe10101ffe50110ffe30114ffe20106"
tlvLen	the length of tlv data buffer

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.14.4.82 void emv_setAutoAuthenticateTransaction (IN int *authenticate*)

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

Parameters

authenticate	TRUE = auto authenticate, FALSE = manually authenticate

12.14.4.83 void emv_setAutoCompleteTransaction (IN int *complete*)

Enables complete for EMV transactions. If a emv_authenticateTranaction results in code 0x0004 (go online), then emv_completeTransaction can automatically execute if parameter is set to TRUE

Parameters

complete	TRUE = auto complete, FALSE = manually complete

12.14.4.84 int emv_setCAPK (IN BYTE * *capk,* IN int *capkLen*)

Set Certificate Authority Public Key

Sets the CAPK as specified by the CAKey structure

Parameters

capk	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.
capkLen	the length of capk data buffer

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.14.4.85 int emv_setCRL (IN BYTE * *list,* IN int *lsLen*)

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]
lsLen	the length of list data buffer

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.14.4.86 int emv_setTerminalData (IN BYTE * *tlv,* IN int *tlvLen*)

Set Terminal Data

Sets the Terminal Data for CTLS as specified by the TLV. The first TLV must be Configuration Group Number (Tag FFE4). The terminal global data is group 0, so the first TLV would be FFE40100. Other groups can be defined using this method (1 or greater), and those can be retrieved with emv_getConfigurationGroup(int group), and deleted with emv_removeConfigurationGroup(int group). You cannot delete group 0.

Parameters

tlv	TerminalData configuration file
tlvLen	the length of tlv data buffer

Return values

RETURN_CODE	Values can be parsed with device_getIDGStatusCodeString()

12.14.4.87 int emv_setTerminalMajorConfiguration (IN int *configuration*)

Sets the terminal major configuration in ICS .

Parameters

configuration	A configuration value, range 1-23 • 1 = 1C • 2 = 2C • 3 = 3C • 4 = 4C • 5 = 5C ... • 23 = 23C

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.14.4.88 void emv_setTransactionParameters (IN double *amount,* IN double *amtOther,* IN int *type,* IN int *timeout,* IN BYTE * *tags,* IN int *tagsLen*)

Set EMV Transaction Parameters

Set the parameters to be used on EMV transactions for an ICC card when Auto Poll is on

The tags will be returned in the callback routine.

Parameters

amount	Transaction amount value (tag value 9F02)
amtOther	Other amount value, if any (tag value 9F03)
type	Transaction type (tag value 9C).
timeout	Timeout value in seconds.
tags	Any other tags to be included in the request (Maximum Length = 500 bytes). Passed as a string. Example, tag 9F0C with amount 0x000000000100 would be "9F0C06000000000100" 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. 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 = DFEE1A079F029F36959F37

tagsLen	the length of tags

12.14.4.89 int emv_startTransaction (IN double *amount,* IN double *amtOther,* IN int *exponent,* IN int *type,* IN int *timeout,* IN BYTE * *tags,* IN int *tagsLen,* IN int *forceOnline*)

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) - SEE IMPORTANT NOTE BELOW
amtOther	Other amount value, if any (tag value 9F03) - SEE IMPORTANT NOTE BELOW
exponent	Number of characters after decimal point
type	Transaction type (tag value 9C).
timeout	Timeout value in seconds.
tags	Any other tags to be included in the request. Passed as a TLV stream. Example, tag 9F0C with amount 0x000000000100 would be 0x9F0C06000000000100 If tags 9F02 (amount),9-F03 (other amount), or 9C (transaction type) are included, they will take priority over these values supplied as individual parameters to this method.
tagsLen	Length of tags
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 = DFEE1A079F029-F36959F37

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString >>>>>IMPORTANT: parameters for amount and amtOther MUST BE PASSED AS A DOUBLE VALUE WITH DECIMAL POINT. Example, do not pass 1, but instead pass 1.0 or 1.00. Otherwise, results will be unpredictable

12.14.4.90 int icc_exchangeAPDU (IN BYTE * *c_APDU,* IN int *cLen,* OUT BYTE * *reData,* IN_OUT int * *reLen*)

Exchange APDU with plain text For Non-SRED Augusta Only

Sends an APDU packet to the ICC. If successful, response is the APDU data in response parameter.

Parameters

c_APDU	APDU data packet
cLen	APDU data packet length
reData	Unencrypted APDU response
reLen	Unencrypted APDU response data length

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString

12.14.4.91 int icc_getICCReaderStatus (OUT 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_getIDGStatusCodeString

12.14.4.92 int icc_powerOffICC ( )

Power Off ICC

Powers down the ICC

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

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

12.14.4.93 int icc_powerOnICC (OUT BYTE * *ATR,* IN_OUT int * *inLen*)

Power On ICC

Power up the currently selected microprocessor card in the ICC reader

Parameters

ATR,the	ATR data response when succeeded power on ICC,
inLen,the	length of ATR data

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.14.4.94 int msr_cancelMSRSwipe ( )

Disable MSR Swipe Cancels MSR swipe request.

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.14.4.95 void msr_registerCallBk (pMSR_callBack *pMSRf*)

To register the msr callback function to get the MSR card data. (Pass NULL to disable the callback.)

12.14.4.96 void msr_registerCallBkp (pMSR_callBackp *pMSRf*)

To register the msr callback function to get the MSR card data pointer. (Pass NULL to disable the callback.)

12.14.4.97 int msr_startMSRSwipe (IN int *_timeout*)

Start MSR Swipe Enables MSR, waiting for swipe to occur. Allows track selection. Returns IDTMSRData instance to deviceDelegate::swipeMSRData:()

Parameters

timeout	Swipe Timeout Value

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString() Note: if auto poll mode is on, it will return command not allowed error

12.14.4.98 void parseMSRData (IN BYTE * *resData*, IN int *resLen*, IN_OUT IDTMSRData * *cardData*)

Parser the MSR data from the buffer into IDTMSTData structure

Parameters

resData	MSR card data buffer
resLen	the length of resData
cardData	the parser result with IDTMSTData structure

12.14.4.99 void pin_registerCallBk (pPIN_callBack *pPINf*)

To register the pin callback function to get the PINPad data. (Pass NULL to disable the callback.)

12.14.4.100 void registerHotplugCallBk (pMessageHotplug *pMsgHotplug*)

To register the USB HID hot-plug callback function which implemented in the application to monitor the hotplug message from the SDK.

12.14.4.101 void registerLogCallBk (pSendDataLog *pFSend*, pReadDataLog *pFRead*)

To register the log callback function which implemented in the application to monitor sending/reading data between application and reader.

12.14.4.102 char* SDK_Version ( )

To Get SDK version

Returns

return the SDK version string

12.14.4.103 int setAbsoluteLibraryPath (const char * *absoluteLibraryPath*)

Set the path to use when searching for ID TECH's libraries. If this is not set, the libraries will be searched for with the system's default procedures.

Parameters

absoluteLibrary-Path	The absolute path to ID TECH's libraries.

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.15 Source_C/libIDT_VP3300_USB.h File Reference

VP3300 USB API.

```
#include "IDTDef.h"
```

Macros

- #define IN
- #define OUT
- #define IN_OUT

Typedefs

- typedef void(∗ pMessageHotplug)(int, int)
- typedef void(∗ pSendDataLog)(unsigned char ∗, int)
- typedef void(∗ pReadDataLog)(unsigned char ∗, int)
- typedef void(∗ pEMV_callBack)(int, int, unsigned char ∗, int, IDTTransactionData ∗, EMV_Callback ∗, int)
- typedef void(∗ pMSR_callBack)(int, IDTMSRData)
- typedef void(∗ pMSR_callBackp)(int, IDTMSRData ∗)
- typedef void(∗ pPIN_callBack)(int, IDTPINData ∗)
- typedef void(∗ pCMR_callBack)(int, IDTCMRData ∗)
- typedef void(∗ pCSFS_callBack)(BYTE status)
- typedef void(∗ ftpComm_callBack)(int, int, int)
- typedef void(∗ httpComm_callBack)(BYTE ∗, int)
- typedef void(∗ v4Comm_callBack)(BYTE, BYTE, BYTE ∗, int)

Functions

- void registerHotplugCallBk (pMessageHotplug pMsgHotplug)
- void registerLogCallBk (pSendDataLog pFSend, pReadDataLog pFRead)
- void device_registerRKICallBk (pRKI_callBack pRKIf)
- void emv_registerCallBk (pEMV_callBack pEMVf)
- void msr_registerCallBk (pMSR_callBack pMSRf)
- void msr_registerCallBkp (pMSR_callBackp pMSRf)
- void ctls_registerCallBk (pMSR_callBack pCTLSf)
- void ctls_registerCallBkp (pMSR_callBackp pCTLSf)
- void pin_registerCallBk (pPIN_callBack pPINf)
- void device_registerCameraCallBk (pCMR_callBack pCMRf)
- void device_registerCardStatusFrontSwitchCallBk (pCSFS_callBack pCSFSf)
- void comm_registerHTTPCallback (httpComm_callBack cBack)
- void comm_registerV4Callback (v4Comm_callBack cBack)
- char ∗ SDK_Version ()
- int setAbsoluteLibraryPath (const char ∗absoluteLibraryPath)
- int device_init ()
- int device_setCurrentDevice (int deviceType)
- int device_isAttached (int deviceType)
- int device_close ()
- void device_getIDGStatusCodeString (IN int returnCode, OUT char ∗despcrition)
- int device_isConnected ()
- int device_getFirmwareVersion (OUT char ∗firmwareVersion)
- int device_getFirmwareVersion_Len (OUT char ∗firmwareVersion, IN_OUT int ∗firmwareVersionLen)

- int device_pingDevice ()
- int device_controlUserInterface (IN BYTE *values)
- int device_getCurrentDeviceType ()
- int device_SendDataCommandNEO (IN int cmd, IN int subCmd, IN BYTE *data, IN int dataLen, OUT BYTE *response, IN_OUT int *respLen)
- int device_enablePassThrough (int enablePassThrough)
- int device_setBurstMode (IN BYTE mode)
- int device_setPollMode (IN BYTE mode)
- int device_setMerchantRecord (int index, int enabled, char *merchantID, char *merchantURL)
- int device_getMerchantRecord (IN int index, OUT BYTE *record)
- int device_getMerchantRecord_Len (IN int index, OUT BYTE *record, IN_OUT int *recordLen)
- int device_getTransactionResults (IDTMSRData *cardData)
- int device_startTransaction (IN double amount, IN double amtOther, IN int type, IN const int _timeout, IN BYTE *tags, IN int tagsLen)
- void device_setTransactionExponent (int exponent)
- int device_activateTransaction (IN const int _timeout, IN BYTE *tags, IN int tagsLen)
- int device_cancelTransaction ()
- int device_getRTCDateTime (IN BYTE *dateTime, IN_OUT int *dateTimeLen)
- int device_setRTCDateTime (IN BYTE *dateTime, IN int dateTimeLen)
- int device_startRKI ()
- int device_getSDKWaitTime ()
- void device_setSDKWaitTime (int waitTime)
- int device_getThreadStackSize ()
- void device_setThreadStackSize (int threadSize)
- int config_getSerialNumber (OUT char *sNumber)
- int config_getSerialNumber_Len (OUT char *sNumber, IN_OUT int *sNumberLen)
- int ctls_startTransaction (IN double amount, IN double amtOther, IN int type, IN const int _timeout, IN BYTE *tags, IN int tagsLen)
- int ctls_activateTransaction (IN const int _timeout, IN BYTE *tags, IN int tagsLen)
- int ctls_cancelTransaction ()
- int ctls_retrieveApplicationData (IN BYTE *AID, IN int AIDLen, OUT BYTE *tlv, IN_OUT int *tlvLen)
- int ctls_setApplicationData (IN BYTE *tlv, IN int tlvLen)
- int ctls_removeApplicationData (IN BYTE *AID, IN int AIDLen)
- int ctls_removeAllApplicationData ()
- int ctls_retrieveAIDList (OUT BYTE *AIDList, IN_OUT int *AIDListLen)
- int ctls_retrieveTerminalData (OUT BYTE *tlv, IN_OUT int *tlvLen)
- int ctls_setTerminalData (IN BYTE *tlv, IN int tlvLen)
- int ctls_retrieveCAPK (IN BYTE *capk, IN int capkLen, OUT BYTE *key, IN_OUT int *keyLen)
- int ctls_setCAPK (IN BYTE *capk, IN int capkLen)
- int ctls_removeCAPK (IN BYTE *capk, IN int capkLen)
- int ctls_removeAllCAPK ()
- int ctls_retrieveCAPKList (OUT BYTE *keys, IN_OUT int *keysLen)
- int ctls_setConfigurationGroup (IN BYTE *tlv, IN int tlvLen)
- int ctls_getConfigurationGroup (IN int group, OUT BYTE *tlv, OUT int *tlvLen)
- int ctls_getAllConfigurationGroups (OUT BYTE *tlv, IN_OUT int *tlvLen)
- int ctls_removeConfigurationGroup (int group)
- void emv_allowFallback (IN int allow)
- void emv_setAutoAuthenticateTransaction (IN int authenticate)
- void emv_setAutoCompleteTransaction (IN int complete)
- int emv_getAutoAuthenticateTransaction ()
- int emv_getAutoCompleteTransaction ()
- void emv_setTransactionParameters (IN double amount, IN double amtOther, IN int type, IN int timeout, IN BYTE *tags, IN int tagsLen)
- int emv_startTransaction (IN double amount, IN double amtOther, IN int exponent, IN int type, IN int timeout, IN BYTE *tags, IN int tagsLen, IN int forceOnline)

- int emv_activateTransaction (IN int timeout, IN BYTE *tags, IN int tagsLen, IN int forceOnline)
- int emv_authenticateTransaction (IN BYTE *updatedTLV, IN int updatedTLVLen)
- int emv_authenticateTransactionWithTimeout (IN int timeout, IN BYTE *updatedTLV, IN int updatedTLVLen)
- int emv_completeTransaction (IN int commError, IN BYTE *authCode, IN int authCodeLen, IN BYTE *iad, IN int iadLen, IN BYTE *tlvScripts, IN int tlvScriptsLen, IN BYTE *tlv, IN int tlvLen)
- int emv_cancelTransaction ()
- int emv_retrieveApplicationData (IN BYTE *AID, IN int AIDLen, OUT BYTE *tlv, IN_OUT int *tlvLen)
- int emv_setApplicationData (IN BYTE *name, IN int nameLen, IN BYTE *tlv, IN int tlvLen)
- int emv_setApplicationDataTLV (IN BYTE *tlv, IN int tlvLen)
- int emv_removeApplicationData (IN BYTE *AID, IN int AIDLen)
- int emv_removeAllApplicationData ()
- int emv_retrieveAIDList (OUT BYTE *AIDList, IN_OUT int *AIDListLen)
- int emv_retrieveTerminalData (OUT BYTE *tlv, IN_OUT int *tlvLen)
- int emv_setTerminalData (IN BYTE *tlv, IN int tlvLen)
- int emv_setTerminalMajorConfiguration (IN int configuration)
- int emv_retrieveCAPK (IN BYTE *capk, IN int capkLen, OUT BYTE *key, IN_OUT int *keyLen)
- int emv_setCAPK (IN BYTE *capk, IN int capkLen)
- int emv_removeCAPK (IN BYTE *capk, IN int capkLen)
- int emv_removeAllCAPK ()
- int emv_retrieveCAPKList (OUT BYTE *keys, IN_OUT int *keysLen)
- int emv_retrieveCRL (OUT BYTE *list, IN_OUT int *lssLen)
- int emv_setCRL (IN BYTE *list, IN int lsLen)
- int emv_removeCRL (IN BYTE *list, IN int lsLen)
- int emv_removeAllCRL ()
- int icc_getICCReaderStatus (OUT BYTE *status)
- int icc_powerOnICC (OUT BYTE *ATR, IN_OUT int *inLen)
- int icc_powerOffICC ()
- int icc_exchangeAPDU (IN BYTE *c_APDU, IN int cLen, OUT BYTE *reData, IN_OUT int *reLen)
- int msr_cancelMSRSwipe ()
- int msr_startMSRSwipe (IN int _timeout)
- void parseMSRData (IN BYTE *resData, IN int resLen, IN_OUT IDTMSRData *cardData)

12.15.1 Detailed Description

VP3300 USB API. VP3300 USB Global API methods.

12.15.2 Macro Definition Documentation

12.15.2.1 #define IN

INPUT parameter.

12.15.2.2 #define IN_OUT

INPUT / OUTPUT PARAMETER.

12.15.2.3 #define OUT

OUTPUT parameter.

12.15.3 Typedef Documentation

12.15.3.1 typedef void(∗ ftpComm_callBack)(int, int, int)

Define the comm callback function to get FTP file transfer status

It should be passed as a parameter in a FTP request, Signature (int, int, int) = response code, current block, total blocks RESPONSE CODES: 100 = FILE DOWNLOAD STARTED 101 = FILE BLOCK XX OF XX RECEIVED 102 = FILE DOWNLOAD COMPLETED 103 = FILE DOWNLOAD TERMINATED PREMATURELY

12.15.3.2 typedef void(∗ httpComm_callBack)(BYTE ∗, int)

Define the comm callback function to get the async url data

It should be registered using the comm_registerHTTPCallback

12.15.3.3 typedef void(∗ pCMR_callBack)(int, IDTCMRData ∗)

Define the camera callback function to get the image data

It should be registered using the device_registerCameraCallBk,

12.15.3.4 typedef void(∗ pCSFS_callBack)(BYTE status)

Define the card status and front switch callback function to get card and front switch status

It should be registered using the device_registerCardStatusFrontSwitchCallBk,

12.15.3.5 typedef void(∗ pEMV_callBack)(int, int, unsigned char ∗, int, IDTTransactionData ∗, EMV_Callback ∗, int)

Define the EMV callback function to get the transaction message/data/result.

It should be registered using the emv_registerCallBk,

12.15.3.6 typedef void(∗ pMessageHotplug)(int, int)

Define the USB hot-plug callback function to monitor the info when plug in/out the reader.

It should be registered using the registerHotplugCallBk, The first integer parameter is device type, and the second integer parameter is either 0: Device Plugged Out or 1: Device Plugged In

12.15.3.7 typedef void(∗ pMSR_callBack)(int, IDTMSRData)

Define the MSR callback function to get the MSR card data

It should be registered using the msr_registerCallBk, this callback function is for backward compatibility

12.15.3.8 typedef void(∗ pMSR_callBackp)(int, IDTMSRData ∗)

Define the MSR callback function to get pointer to the MSR card data

It should be registered using the msr_registerCallBk, this callback function is recommended instead of pMSR_call-Back

12.15.3.9 typedef void(* pPIN_callBack)(int, IDTPINData *)

Define the PINPad callback function to get the input PIN Pad data

It should be registered using the pin_registerCallBk,

12.15.3.10 typedef void(* pReadDataLog)(unsigned char *, int)

Define the read response callback function to monitor the reading response from the reader.

It should be registered using the registerLogCallBk,

12.15.3.11 typedef void(* pSendDataLog)(unsigned char *, int)

Define the send command callback function to monitor the sending command into the reader.

It should be registered using the registerLogCallBk,

12.15.3.12 typedef void(* v4Comm_callBack)(BYTE, BYTE, BYTE *, int)

Define the comm callback function to receive the V4 Protocol packets received by the device from an external source (IP/USB/RS-232) It should be registered using the comm_registerV4Callback, Data callback will contain command, sub-command, and data from V4 packet

12.15.4 Function Documentation

12.15.4.1 void comm_registerHTTPCallback (httpComm_callBack *cBack*)

Register Comm HTTP Async Callback

Parameters

cBack	- HTTP Comm callback

12.15.4.2 void comm_registerV4Callback (v4Comm_callBack *cBack*)

Register External V4 Protocol commands Callback

Parameters

cBack	- V4 Protocol Comm callback

12.15.4.3 int config_getSerialNumber (OUT char * *sNumber*)

DEPRECATED : please use config_getSerialNumber_Len(OUT char* sNumber, IN_OUT int *sNumberLen)

Polls device for Serial Number

Parameters

sNumber	Returns Serial Number; needs to have at least 64 bytes of memory

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString

12.15.4.4 int config_getSerialNumber_Len (OUT char * *sNumber,* IN_OUT int * *sNumberLen*)

Polls device for Serial Number

Parameters

sNumber	Returns Serial Number
sNumberLen	Length of Serial Number

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString

12.15.4.5 int ctls_activateTransaction (IN const int *_timeout,* IN BYTE * *tags,* IN int *tagsLen*)

Start CTLS Transaction Request

Authorizes the CTLS 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 stream. Example, tag 9F0C with amount 0x000000000100 would be 0x9F0C06000000000100
tagsLen	The length of tags data buffer.

>>>>>IMPORTANT: parameters for amount and amtOther MUST BE PASSED AS A DOUBLE VALUE WITH DECIMAL POINT. Example, do not pass 1, but instead pass 1.0 or 1.00. Otherwise, results will be unpredictable

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString() Note: if auto poll is on, it will returm the error IDG_P2_STATUS_CODE_COMMAND_NOT_ALLOWED

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

12.15.4.6 int ctls_cancelTransaction ( )

Cancel EMV Transaction

Cancels the currently executing EMV transaction.

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.15.4.7 int ctls_getAllConfigurationGroups (OUT BYTE * *tlv*, IN_OUT int * *tlvLen*)

Retrieve All Configuration Groups

Returns all the Configuration Groups installed on the terminal for CTLS

Parameters

tlv	The TLV elements data
tlvLen	the length of tlv data buffer.

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.15.4.8 int ctls_getConfigurationGroup (IN int *group*, OUT BYTE * *tlv*, OUT int * *tlvLen*)

Get Configuration Group

Retrieves the Configuration for the specified Group.

Parameters

group	Configuration Group
tlv	return data
tlvLen	the length of tlv data buffer

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.15.4.9 void ctls_registerCallBk (pMSR_callBack *pCTLSf*)

To register the ctls callback function to get the MSR card data. (Pass NULL to disable the callback.)

12.15.4.10 void ctls_registerCallBkp (pMSR_callBackp *pCTLSf*)

To register the ctls callback function to get the MSR card data pointer. (Pass NULL to disable the callback.)

12.15.4.11 int ctls_removeAllApplicationData ( )

Remove All Application Data

Removes all the Application Data

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.15.4.12 int ctls_removeAllCAPK ( )

Remove All Certificate Authority Public Key

Removes all the CAPK

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.15.4.13 int ctls_removeApplicationData (IN BYTE * *AID*, IN int *AIDLen*)

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
AIDLen	the length of AID data buffer

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.15.4.14 int ctls_removeCAPK (IN BYTE * *capk*, IN int *capkLen*)

Remove Certificate Authority Public Key

Removes the CAPK as specified by the RID/Index

Parameters

capk	6 byte CAPK = 5 bytes RID + 1 byte INDEX
capkLen	the length of capk data buffer

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.15.4.15 int 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	Values can be parsed with device_getIDGStatusCodeString()

12.15.4.16 int ctls_retrieveAIDList (OUT BYTE * *AIDList,* IN_OUT int * *AIDListLen*)

Retrieve AID list

Returns all the AID names installed on the terminal for CTLS. .

Parameters

AIDList	array of AID name byte arrays
AIDListLen	the length of AIDList array buffer

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.15.4.17 int ctls_retrieveApplicationData (IN BYTE * *AID,* IN int *AIDLen,* OUT BYTE * *tlv,* IN_OUT int * *tlvLen*)

Retrieve Application Data by AID

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

Parameters

AID	Name of ApplicationID. Must be between 5 and 16 bytes
AIDLen	the length of AID data buffer.
tlv	The TLV elements of the requested AID
tlvLen	the length of tlv data buffer.

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.15.4.18 int ctls_retrieveCAPK (IN BYTE * *capk,* IN int *capkLen,* OUT BYTE * *key,* IN_OUT int * *keyLen*)

Retrieve Certificate Authority Public Key

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

Parameters

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

capkLen	the length of capk data buffer
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.
keyLen	the length of key data buffer •

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.15.4.19 int ctls_retrieveCAPKList (OUT BYTE * *keys*, IN_OUT int * *keysLen*)

Retrieve the Certificate Authority Public Key list

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

Parameters

keys	[key1][key2]...[keyn], each key 6 bytes where key = 5 bytes RID + 1 byte index
keysLen	the length of keys data buffer

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.15.4.20 int ctls_retrieveTerminalData (OUT BYTE * *tlv*, IN_OUT int * *tlvLen*)

Retrieve Terminal Data

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

Parameters

tlv	Response returned as a TLV
tlvLen	the length of tlv data buffer

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.15.4.21 int ctls_setApplicationData (IN BYTE * *tlv*, IN int *tlvLen*)

Set Application Data by AID

Sets the Application Data for CTLS as specified by TLV data

Parameters

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

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

Parameters

tlvLen	the length of tlv data buffer

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.15.4.22 int ctls_setCAPK (IN BYTE * *capk*, IN int *capkLen*)

Set Certificate Authority Public Key

Sets the CAPK as specified by the CAKey structure

Parameters

capk	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.
capkLen	the length of capk data buffer

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.15.4.23 int ctls_setConfigurationGroup (IN BYTE * *tlv*, IN int *tlvLen*)

Set Configuration Group

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

Parameters

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

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.15.4.24 int ctls_setTerminalData (IN BYTE * *tlv*, IN int *tlvLen*)

Set Terminal Data

Sets the Terminal Data for CTLS as specified by the TLV. The first TLV must be Configuration Group Number (Tag FFE4). The terminal global data is group 0, so the first TLV would be FFE40100. Other groups can be defined using this method (1 or greater), and those can be retrieved with ctls_getConfigurationGroup(int group), and deleted with ctls_removeConfigurationGroup(int group). You cannot delete group 0.

Parameters

tlv	TerminalData configuration file
tlvLen	the length of tlv data buffer

Return values

RETURN_CODE	Values can be parsed with device_getIDGStatusCodeString()

12.15.4.25 int ctls_startTransaction (IN double *amount*, IN double *amtOther*, IN int *type*, IN const int *_timeout*, IN BYTE * *tags*, IN int *tagsLen*)

Start CTLS Transaction Request

Authorizes the CTLS transaction for an ICC card

The tags will be returned in the callback routine.

Parameters

amount	Transaction amount value (tag value 9F02) - SEE IMPORTANT NOTE BELOW
amtOther	Other amount value, if any (tag value 9F03) - SEE IMPORTANT NOTE BELOW
type	Transaction type (tag value 9C).
timeout	Timeout value in seconds.
tags	Any other tags to be included in the request. Passed as TLV stream. Example, tag 9F0C with amount 0x000000000100 would be 0x9F0C06000000000100 If tags 9F02 (amount),9-F03 (other amount), or 9C (transaction type) are included, they will take priority over these values supplied as individual parameters to this method.
tagsLen	The length of tags data buffer.

>>>>>IMPORTANT: parameters for amount and amtOther MUST BE PASSED AS A DOUBLE VALUE WITH DECIMAL POINT. Example, do not pass 1, but instead pass 1.0 or 1.00. Otherwise, results will be unpredictable

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString() Note: if auto poll is on, it will returm the error IDG_P2_STATUS_CODE_COMMAND_NOT_ALLOWED

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

12.15.4.26 int device_activateTransaction (IN const int *_timeout,* IN BYTE * *tags,* IN int *tagsLen*)

Start Transaction Request

Authorizes the transaction for an MSR/CTLS/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 a TLV. Example, tag 9F0C with amount 0x000000000100 would be 0x9F0C06000000000100 Be sure to include 9F02 (amount)and9-C (transaction type).
tagsLen	The length of tags data buffer.

>>>>>IMPORTANT: parameters for amount and amtOther MUST BE PASSED AS A DOUBLE VALUE WITH DECIMAL POINT. Example, do not pass 1, but instead pass 1.0 or 1.00. Otherwise, results will be unpredictable

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString() Note: if auto poll is on, it will returm the error IDG_P2_STATUS_CODE_COMMAND_NOT_ALLOWED

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 device_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

12.15.4.27 int device_cancelTransaction ( )

Disable Transaction Cancel Transaction request.

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.15.4.28 int device_close ( )

Close the device

Returns

RETURN_CODE: 0: success, 0x0A: failed

12.15.4.29 int device_controlUserInterface (IN BYTE * *values*)

Control User Interface

Controls the User Interface: Display, Beep, LED

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

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.15.4.30 int device_enablePassThrough (int *enablePassThrough*)

Start Remote Key Injection

Starts a remote key injection request with IDTech RKI servers. This function is reserved and not implemented.

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

Enable Pass Through

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

Parameters

enablePass-Through	1 = pass through ON, 0 = pass through OFF

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.15.4.31 int device_getCurrentDeviceType ( )

Get current active device type

Returns

: return the device type defined as DEVICE_TYPE in the IDTDef.h

12.15.4.32 int device_getFirmwareVersion (OUT char * *firmwareVersion*)

DEPRECATED : please use device_getFirmwareVersion_Len(OUT char* firmwareVersion, IN_OUT int *firmware-VersionLen)

Polls device for Firmware Version

Parameters

firmwareVersion	Response returned of Firmware Version; needs to have at least 128 bytes of memory

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.15.4.33 int device_getFirmwareVersion_Len (OUT char * *firmwareVersion,* IN_OUT int * *firmwareVersionLen*)

Polls device for Firmware Version

Parameters

firmwareVersion	Response returned of Firmware Version
firmwareVersion-Len	Length of Firmware Version

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.15.4.34 void device_getIDGStatusCodeString (IN int *returnCode,* OUT char * *despcrition*)

Review the return code description.

Parameters

returnCode	the response result.
description	

Return values

the	string for description of response result

- 0: "no error, beginning task";
- 1: "no response from reader";
- 2: "invalid response data";
  - 01: " Incorrect Header Tag";
  - 02: " Unknown Command";
  - 03: " Unknown Sub-Command";
  - 04: " CRC Error in Frame";
  - 05: " Incorrect Parameter";
  - 06: " Parameter Not Supported";
  - 07: " Mal-formatted Data";
  - 08: " Timeout";
  - 0A: " Failed / NACK";
  - 0B: " Command not Allowed";
  - 0C: " Sub-Command not Allowed";
  - 0D: " Buffer Overflow (Data Length too large for reader buffer)";
  - 0E: " User Interface Event";
  - 10: " Need clear firmware(apply in boot loader only)";
  - 11: " Communication type not supported, VT-1, burst, etc. Need encrypted firmware (apply in boot loader only)";
  - 12: " Secure interface is not functional or is in an intermediate state.";
  - 13: " Data field is not mod 8";
  - 14: " Pad 0x80 not found where expected";
  - 15: " Specified key type is invalid";
  - 16: " Could not retrieve key from the SAM (InitSecureComm)";
  - 17: " Hash code problem";
  - 18: " Could not store the key into the SAM (InstallKey)";
  - 19: " Frame is too large";
  - 1A: " Unit powered up in authentication state but POS must resend the InitSecureComm command";
  - 1B: " The EEPROM may not be initialized because SecCommInterface does not make sense";
  - 1C: " Problem encoding APDU Module-Specific Status Codes ";
  - 20: " Unsupported Index (ILM) SAM Transceiver error - problem communicating with the SAM (Key Mgr)";
  - 21: " Unexpected Sequence Counter in multiple frames for single bitmap (ILM)Length error in data returned from the SAM (Key Mgr) - 22: " Improper bit map (ILM)"; - 23: " Request Online Authorization"; - 24: " ViVOCard3 raw data read successful"; - 25: " Message index not available (ILM) ViVOcomm activate transaction card type (ViVOcomm)"; - 26: " Version Information Mismatch (ILM)"; - 27: " Not sending commands in correct index message index (ILM)"; - 28: " Time out or next expected message not received (ILM)"; - 29: " ILM languages not available for viewing (ILM)"; - 2A: " Other language not supported (ILM)"; - 41: " from 41 to 4F, Module-specific errors for Key Manager";
  - 50: " Auto-Switch OK";
  - 51: " Auto-Switch failed";
  - 70: " Antenna Error 80h Use another card";
  - 81: " Insert or swipe card";
  - 90: " Data encryption Key does not exist";
  - 91: " Data encryption Key KSN exhausted";

12.15.4.35 int device_getMerchantRecord (IN int *index,* OUT BYTE * *record*)

DEPRECATED : please use device_getMerchantRecord_Len(IN int index, OUT BYTE * record, IN_OUT int *recordLen)

Get Merchant Record

Gets the merchant record for the device.

Parameters

index	Merchant Record index, valid values 1-6
record;	needs to have at least 99 bytes of memory response data from reader. Merchant Record Index: 1 byte enabled: 1 byte Merchant ID: 32 bytes Length of Merchant URL: 1 byte Merchant URL: 64 bytes

Returns

success or error code. Values can be parsed with device_getIDGStatusCodeString()

See Also

ErrorCode

12.15.4.36 int device_getMerchantRecord_Len (IN int *index,* OUT BYTE * *record,* IN_OUT int * *recordLen*)

Get Merchant Record

Gets the merchant record for the device.

Parameters

index	Merchant Record index, valid values 1-6
record	response data from reader. Merchant Record Index: 1 byte enabled: 1 byte Merchant ID: 32 bytes Length of Merchant URL: 1 byte Merchant URL: 64 bytes
recordLen	Length of record

Returns

success or error code. Values can be parsed with device_getIDGStatusCodeString()

See Also

ErrorCode

12.15.4.37 int device_getRTCDateTime (IN BYTE * *dateTime,* IN_OUT int * *dateTimeLen*)

get RTC date and time of the device

Parameters

dateTime	<dateTime data=""> is: 6 byte data for YYMMDDHHMMSS in hex. For example 0x171003102547 stands for 2017 Oct 3rd 10:25:47
dateTimeLen	return 6 bytes if successful

Returns

success or error code. Values can be parsed with device_getResponseCodeString

See Also

ErrorCode

12.15.4.38 int device_getSDKWaitTime ( )

Get SDK Wait Time

Get the SDK wait time for transactions

Returns

SDK wait time in seconds

12.15.4.39 int device_getThreadStackSize ( )

Get Thread Stack Size

Get the stack size setting for newly created threads

Returns

Thread Stack Size

12.15.4.40 int device_getTransactionResults (IDTMSRData * *cardData*)

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

Parameters

cardData	The transaction results

Returns

success or error code. Values can be parsed with device_getResponseCodeString

See Also

ErrorCode

12.15.4.41 int device_init ( )

Initial the device by USB

It will detect the device and trying connect.

The connect status can be checked by device_isConnected().

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.15.4.42 int device_isAttached (int *deviceType*)

Check if the device is attached to the USB port The function device_init() must be called before this function.

Parameters

deviceType,the	device type of the USB device

Returns

1 if the device is attached, or 0 if the device is not attached

12.15.4.43 int device_isConnected ( )

Check the device conntected status

Returns

DEVICE_DISCONNECT=0, or DEVICE_CONNECTED = 1

12.15.4.44 int device_pingDevice ( )

Ping Device

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

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.15.4.45 void device_registerCameraCallBk (pCMR_callBack *pCMRf*)

To register the camera callback function to get the image data. (Pass NULL to disable the callback.)

12.15.4.46 void device_registerCardStatusFrontSwitchCallBk (pCSFS_callBack *pCSFSf*)

To register the card status and front switch callback function to get status. (Pass NULL to disable the callback.)

12.15.4.47 void device_registerRKICallBk (pRKI_callBack *pRKIf*)

To register the RKI callback function to get the RKI status. (Pass NULL to disable the callback.)

12.15.4.48 int device_SendDataCommandNEO (IN int *cmd*, IN int *subCmd*, IN BYTE * *data*, IN int *dataLen*, OUT BYTE * *response*, IN_OUT int * *respLen*)

Send a Command to device

Sends a command to the device .

Parameters

cmd	buffer of command to execute.
cmdLen,the	length of the buffer cmd.
data	buffer of IDG command data.

dataLen,the	length of the buffer data.
response	Response data
respLen,the	length of Response data

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString() Send a Command to NEO device

Sends a command to the NEO device .

Parameters

cmd	command to execute.
subCmd,sub	command to execute.
data	buffer of NEO command data.
dataLen,the	length of the buffer data.
response	Response data
respLen,the	length of Response data

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.15.4.49 int device_setBurstMode (IN BYTE *mode*)

Send Burst Mode

Sets the burst mode for the device.

Parameters

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

Returns

success or error code. Values can be parsed with device_getIDGStatusCodeString

See Also

ErrorCode

12.15.4.50 int device_setCurrentDevice (int *deviceType*)

Sets the current device to talk to

The connect status can be checked by device_isConnected().

Parameters

deviceType	Device to connect to
	
	`enum DEVICE_TYPE`
	`{`
	`IDT_DEVICE_UNKNOWN=0,`
	`IDT_DEVICE_AUGUSTA_HID,`
	`IDT_DEVICE_AUGUSTA_KB,`
	`IDT_DEVICE_AUGUSTA_S_HID,`
	`IDT_DEVICE_AUGUSTA_S_KB,`
	`IDT_DEVICE_AUGUSTA_S_TTK_HID,`
	`IDT_DEVICE_SPECTRUM_PRO,`
	`IDT_DEVICE_MINISMART_II,`
	`IDT_DEVICE_L100,`
	`IDT_DEVICE_UNIPAY,`
	`IDT_DEVICE_UNIPAY_I_V,`
	`IDT_DEVICE_VP3300_AJ,`
	`IDT_DEVICE_KIOSK_III,`
	`IDT_DEVICE_KIOSK_III_S,`
	`IDT_DEVICE_PIP_READER,`
	`IDT_DEVICE_VENDI,`
	`IDT_DEVICE_VP3300_USB,`
	`IDT_DEVICE_UNIPAY_I_V_TTK,`
	`IDT_DEVICE_VP3300_BT,`
	`IDT_DEVICE_VP8800,`
	`IDT_DEVICE_SREDKEY2_HID,`
	`IDT_DEVICE_SREDKEY2_KB,`
	`IDT_DEVICE_NEO2,`
	`IDT_DEVICE_MINISMART_II_COM = IDT_DEVICE_NEO2+5,`
	`IDT_DEVICE_SPECTRUM_PRO_COM,`
	`IDT_DEVICE_KIOSK_III_COM,`
	`IDT_DEVICE_KIOSK_III_S_COM,`
	`IDT_DEVICE_VP3300_COM,`
	`IDT_DEVICE_NEO2_COM,`
	`IDT_DEVICE_MAX_DEVICES = IDT_DEVICE_NEO2_COM+5`
	`};`

Returns

RETURN_CODE: 1: success, 0: failed

12.15.4.51 int device_setMerchantRecord (int *index*, int *enabled*, char * *merchantID*, char * *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 device_getIDGStatusCodeString()

12.15.4.52 int device_setPollMode (IN BYTE *mode*)

Set 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 device_getIDGStatusCodeString()

12.15.4.53 int device_setRTCDateTime (IN BYTE * *dateTime,* IN int *dateTimeLen*)

set RTC date and time of the device

Parameters

dateTime	<dateTime data=""> is: 6 byte data for YYMMDDHHMMSS in hex. For example 0x171003102547 stands for 2017 Oct 3rd 10:25:47
dateTimeLen	should be always 6 bytes

Returns

success or error code. Values can be parsed with device_getResponseCodeString

See Also

ErrorCode

12.15.4.54 void device_setSDKWaitTime (int *waitTime*)

Set SDK Wait Time

Set the SDK wait time for transactions

Parameters

waitTime	The SDK wait time for transaction in seconds

12.15.4.55 void device_setThreadStackSize (int *threadSize*)

Set Thread Stack Size

Set the stack size setting for newly created threads

12.15.4.56 void device_setTransactionExponent (int *exponent*)

Sets the transaction exponent to be used with device_startTransaction. Default value is 2

Parameters

exponent,The	exponent to use when calling device_startTransaction

12.15.4.57 int device_startRKI ( )

Start remote key injection.

Returns

success or error code.

See Also

ErrorCode

12.15.4.58 int device_startTransaction (IN double *amount,* IN double *amtOther,* IN int *type,* IN const int *_timeout,* IN BYTE * *tags,* IN int *tagsLen*)

Start device Transaction Request

Authorizes the transaction for an MSR/CTLS/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)
type	Transaction type (tag value 9C).
timeout	Timeout value in seconds.
tags	Any other tags to be included in the request. Passed as TLV. Example, tag 9F0C with amount 0x000000000100 would be 0x9F0C06000000000100 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. 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 = DFEE1A079F029F36959F37

Returns

success or error code. Values can be parsed with device_getResponseCodeString

See Also

ErrorCode Note: if auto poll is on, it will returm the error IDG_P2_STATUS_CODE_COMMAND_NOT_ALLOWED

12.15.4.59 int emv_activateTransaction (IN int *timeout,* IN BYTE * *tags,* IN int *tagsLen,* IN int *forceOnline*)

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	Tags to be included in the request. Passed as a TLV stream. Example, tag 9F0C with amount 0x000000000100 would be 0x9F0C06000000000100
tagsLen	Length of tags
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 = DFEE1A079F029-F369f9F37

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString >>>>>IMPORTANT: parameters for amount and amtOther MUST BE PASSED AS A DOUBLE VALUE WITH DECIMAL POINT. Example, do not pass 1, but instead pass 1.0 or 1.00. Otherwise, results will be unpredictable

12.15.4.60 void emv_allowFallback (IN int *allow*)

Allow fallback for EMV transactions. Default is TRUE

Parameters

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

12.15.4.61 int emv_authenticateTransaction (IN BYTE * *updatedTLV,* IN int *updatedTLVLen*)

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.

Parameters

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, 9F37 to be included in response = DFEE1-A079F029F36959F37

updatedTLVLen	

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.15.4.62 int emv_authenticateTransactionWithTimeout (IN int *timeout,* IN BYTE * *updatedTLV,* IN int *updatedTLVLen*)

Authenticate EMV Transaction Request with Timeout

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.

Parameters

timeout	Timeout value in seconds.
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, 9F37 to be included in response = DFEE1-A079F029F36959F37
updatedTLVLen	

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.15.4.63 int emv_cancelTransaction ( )

Cancel EMV Transaction

Cancels the currently executing EMV transaction.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.15.4.64 int emv_completeTransaction (IN int *commError,* IN BYTE * *authCode,* IN int *authCodeLen,* IN BYTE * *iad,* IN int *iadLen,* IN BYTE * *tlvScripts,* IN int *tlvScriptsLen,* IN BYTE * *tlv,* IN int *tlvLen*)

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(1) if host was unreachable, or FALSE(0) 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
authCodeLen	the length of authCode
iad	Issuer Authentication Data, if any. Example 0x11223344556677883030 (tag value 91).
iadLen	the length of iadLen
tlvScripts	71/72 scripts, if any
tlvScriptsLen	the length of tlvScriptsLen
tlv	Additional TLV data to return with transaction results (if any)
tlvLen	the length of tlv

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.15.4.65 int emv_getAutoAuthenticateTransaction ( )

Gets auto authenticate value for EMV transactions.

Returns

RETURN_CODE: TRUE = auto authenticate, FALSE = manually authenticate

12.15.4.66 int emv_getAutoCompleteTransaction ( )

Gets auto complete value for EMV transactions.

Returns

RETURN_CODE: TRUE = auto complete, FALSE = manually complete

12.15.4.67 void emv_registerCallBk (pEMV_callBack *pEMVf*)

To register the emv callback function to get the EMV processing response. (Pass NULL to disable the callback.)

12.15.4.68 int emv_removeAllApplicationData ( )

Remove All Application Data

Removes all the Application Data

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.15.4.69 int emv_removeAllCAPK ( )

Remove All Certificate Authority Public Key

Removes all the CAPK

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.15.4.70 int emv_removeAllCRL ( )

Remove All Certificate Revocation List Entries

Removes all CRLEntry entries

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.15.4.71 int emv_removeApplicationData (IN BYTE * *AID*, IN int *AIDLen*)

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
AIDLen	the length of AID data buffer

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.15.4.72 int emv_removeCAPK (IN BYTE * *capk*, IN int *capkLen*)

Remove Certificate Authority Public Key

Removes the CAPK as specified by the RID/Index

Parameters

capk	6 byte CAPK = 5 bytes RID + 1 byte INDEX
capkLen	the length of capk data buffer

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.15.4.73 int emv_removeCRL (IN BYTE * *list*, IN int *lsLen*)

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
lssLen	the length of list data buffer

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.15.4.74 int emv_retrieveAIDList (OUT BYTE * *AIDList*, IN_OUT int * *AIDListLen*)

Retrieve AID list

Returns all the AID names installed on the terminal for CTLS. .

Parameters

AIDList	array of AID name byte arrays
AIDListLen	the length of AIDList array buffer

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.15.4.75 int emv_retrieveApplicationData (IN BYTE * *AID*, IN int *AIDLen*, OUT BYTE * *tlv*, IN_OUT int * *tlvLen*)

Retrieve Application Data by AID

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

Parameters

AID	Name of ApplicationID. Must be between 5 and 16 bytes
AIDLen	the length of AID data buffer.
tlv	The TLV elements of the requested AID
tlvLen	the length of tlv data buffer.

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.15.4.76 int emv_retrieveCAPK (IN BYTE * *capk*, IN int *capkLen*, OUT BYTE * *key*, IN_OUT int * *keyLen*)

Retrieve Certificate Authority Public Key

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

Parameters

capk	6 bytes CAPK = 5 bytes RID + 1 byte Index
capkLen	the length of capk data buffer
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.
keyLen	the length of key data buffer •

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.15.4.77 int emv_retrieveCAPKList (OUT BYTE * *keys*, IN_OUT int * *keysLen*)

Retrieve the Certificate Authority Public Key list

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

Parameters

keys	[key1][key2]...[keyn], each key 6 bytes where key = 5 bytes RID + 1 byte index
keysLen	the length of keys data buffer

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.15.4.78 int emv_retrieveCRL (OUT BYTE * *list*, IN_OUT int * *lssLen*)

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
lssLen	the length of list data buffer

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.15.4.79 int emv_retrieveTerminalData (OUT BYTE * *tlv*, IN_OUT int * *tlvLen*)

Retrieve Terminal Data

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

Parameters

tlv	Response returned as a TLV
tlvLen	the length of tlv data buffer

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.15.4.80 int emv_setApplicationData (IN BYTE * *name*, IN int *nameLen*, IN BYTE * *tlv*, IN int *tlvLen*)

Set Application Data by AID

Sets the Application Data as specified by the application name and TLV data

Parameters

name	Application name, 10-32 ASCII hex characters representing 5-16 bytes Example "a0000000031010"
nameLen	the length of name data buffer of Application name,
tlv	Application data in TLV format
tlvLen	the length of tlv data buffer

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.15.4.81 int emv_setApplicationDataTLV (IN BYTE * *tlv,* IN int *tlvLen*)

Set Application Data by TLV

Sets the Application Data as specified by the 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) Example valid TLV, for Group #2, AID a0000000035010- : "dfee2d01029f0607a0000000051010ffe10101ffe50110ffe30114ffe20106"
tlvLen	the length of tlv data buffer

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.15.4.82 void emv_setAutoAuthenticateTransaction (IN int *authenticate*)

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

Parameters

authenticate	TRUE = auto authenticate, FALSE = manually authenticate

12.15.4.83 void emv_setAutoCompleteTransaction (IN int *complete*)

Enables complete for EMV transactions. If a emv_authenticateTranaction results in code 0x0004 (go online), then emv_completeTransaction can automatically execute if parameter is set to TRUE

Parameters

complete	TRUE = auto complete, FALSE = manually complete

12.15.4.84 int emv_setCAPK (IN BYTE * *capk,* IN int *capkLen*)

Set Certificate Authority Public Key

Sets the CAPK as specified by the CAKey structure

Parameters

capk	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.
capkLen	the length of capk data buffer

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.15.4.85 int emv_setCRL (IN BYTE * *list,* IN int *lsLen*)

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]
lsLen	the length of list data buffer

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.15.4.86 int emv_setTerminalData (IN BYTE * *tlv,* IN int *tlvLen*)

Set Terminal Data

Sets the Terminal Data for CTLS as specified by the TLV. The first TLV must be Configuration Group Number (Tag FFE4). The terminal global data is group 0, so the first TLV would be FFE40100. Other groups can be defined using this method (1 or greater), and those can be retrieved with emv_getConfigurationGroup(int group), and deleted with emv_removeConfigurationGroup(int group). You cannot delete group 0.

Parameters

tlv	TerminalData configuration file
tlvLen	the length of tlv data buffer

Return values

RETURN_CODE	Values can be parsed with device_getIDGStatusCodeString()

12.15.4.87 int emv_setTerminalMajorConfiguration (IN int *configuration*)

Sets the terminal major configuration in ICS .

Parameters

configuration	A configuration value, range 1-23 • 1 = 1C • 2 = 2C • 3 = 3C • 4 = 4C • 5 = 5C ... • 23 = 23C

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.15.4.88 void emv_setTransactionParameters (IN double *amount,* IN double *amtOther,* IN int *type,* IN int *timeout,* IN BYTE * *tags,* IN int *tagsLen*)

Set EMV Transaction Parameters

Set the parameters to be used on EMV transactions for an ICC card when Auto Poll is on

The tags will be returned in the callback routine.

Parameters

amount	Transaction amount value (tag value 9F02)
amtOther	Other amount value, if any (tag value 9F03)
type	Transaction type (tag value 9C).
timeout	Timeout value in seconds.
tags	Any other tags to be included in the request (Maximum Length = 500 bytes). Passed as a string. Example, tag 9F0C with amount 0x000000000100 would be "9F0C06000000000100" 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. 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 = DFEE1A079F029F36959F37

tagsLen	the length of tags

12.15.4.89 int emv_startTransaction (IN double *amount,* IN double *amtOther,* IN int *exponent,* IN int *type,* IN int *timeout,* IN BYTE * *tags,* IN int *tagsLen,* IN int *forceOnline*)

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) - SEE IMPORTANT NOTE BELOW
amtOther	Other amount value, if any (tag value 9F03) - SEE IMPORTANT NOTE BELOW
exponent	Number of characters after decimal point
type	Transaction type (tag value 9C).
timeout	Timeout value in seconds.
tags	Any other tags to be included in the request. Passed as a TLV stream. Example, tag 9F0C with amount 0x000000000100 would be 0x9F0C06000000000100 If tags 9F02 (amount),9-F03 (other amount), or 9C (transaction type) are included, they will take priority over these values supplied as individual parameters to this method.
tagsLen	Length of tags
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 = DFEE1A079F029-F36959F37

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString >>>>>IMPORTANT: parameters for amount and amtOther MUST BE PASSED AS A DOUBLE VALUE WITH DECIMAL POINT. Example, do not pass 1, but instead pass 1.0 or 1.00. Otherwise, results will be unpredictable

12.15.4.90 int icc_exchangeAPDU (IN BYTE * *c_APDU,* IN int *cLen,* OUT BYTE * *reData,* IN_OUT int * *reLen*)

Exchange APDU with plain text For Non-SRED Augusta Only

Sends an APDU packet to the ICC. If successful, response is the APDU data in response parameter.

Parameters

c_APDU	APDU data packet
cLen	APDU data packet length
reData	Unencrypted APDU response
reLen	Unencrypted APDU response data length

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString

12.15.4.91 int icc_getICCReaderStatus (OUT 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_getIDGStatusCodeString

12.15.4.92 int icc_powerOffICC ( )

Power Off ICC

Powers down the ICC

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

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

12.15.4.93 int icc_powerOnICC (OUT BYTE * *ATR,* IN_OUT int * *inLen*)

Power On ICC

Power up the currently selected microprocessor card in the ICC reader

Parameters

ATR,the	ATR data response when succeeded power on ICC,
inLen,the	length of ATR data

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.15.4.94 int msr_cancelMSRSwipe ( )

Disable MSR Swipe Cancels MSR swipe request.

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.15.4.95 void msr_registerCallBk (pMSR_callBack *pMSRf*)

To register the msr callback function to get the MSR card data. (Pass NULL to disable the callback.)

12.15.4.96 void msr_registerCallBkp (pMSR_callBackp *pMSRf*)

To register the msr callback function to get the MSR card data pointer. (Pass NULL to disable the callback.)

12.15.4.97 int msr_startMSRSwipe (IN int *_timeout*)

Start MSR Swipe Enables MSR, waiting for swipe to occur. Allows track selection. Returns IDTMSRData instance to deviceDelegate::swipeMSRData:()

Parameters

timeout	Swipe Timeout Value

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString() Note: if auto poll mode is on, it will return command not allowed error

12.15.4.98 void parseMSRData (IN BYTE * *resData*, IN int *resLen*, IN_OUT IDTMSRData * *cardData*)

Parser the MSR data from the buffer into IDTMSTData structure

Parameters

resData	MSR card data buffer
resLen	the length of resData
cardData	the parser result with IDTMSTData structure

12.15.4.99 void pin_registerCallBk (pPIN_callBack *pPINf*)

To register the pin callback function to get the PINPad data. (Pass NULL to disable the callback.)

12.15.4.100 void registerHotplugCallBk (pMessageHotplug *pMsgHotplug*)

To register the USB HID hot-plug callback function which implemented in the application to monitor the hotplug message from the SDK.

12.15.4.101 void registerLogCallBk (pSendDataLog *pFSend*, pReadDataLog *pFRead*)

To register the log callback function which implemented in the application to monitor sending/reading data between application and reader.

12.15.4.102 char* SDK_Version ( )

To Get SDK version

Returns

return the SDK version string

12.15.4.103 int setAbsoluteLibraryPath (const char * *absoluteLibraryPath*)

Set the path to use when searching for ID TECH's libraries. If this is not set, the libraries will be searched for with the system's default procedures.

Parameters

absoluteLibrary-Path	The absolute path to ID TECH's libraries.

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.16 Source_C/libIDT_VP8800.h File Reference

VP8800 API.

```
#include <stdarg.h>
#include "IDTDef.h"
```

Macros

- #define IN
- #define OUT
- #define IN_OUT

Typedefs

- typedef void(∗ pMessageHotplug)(int, int)
- typedef void(∗ pSendDataLog)(unsigned char ∗, int)
- typedef void(∗ pReadDataLog)(unsigned char ∗, int)
- typedef void(∗ pEMV_callBack)(int, int, unsigned char ∗, int, IDTTransactionData ∗, EMV_Callback ∗, int)
- typedef void(∗ pMSR_callBack)(int, IDTMSRData)
- typedef void(∗ pMSR_callBackp)(int, IDTMSRData ∗)
- typedef void(∗ pPIN_callBack)(int, IDTPINData ∗)
- typedef void(∗ pCMR_callBack)(int, IDTCMRData ∗)
- typedef void(∗ pCSFS_callBack)(BYTE status)
- typedef void(∗ ftpComm_callBack)(int, int, int)
- typedef void(∗ httpComm_callBack)(BYTE ∗, int)
- typedef void(∗ v4Comm_callBack)(BYTE, BYTE, BYTE ∗, int)
- typedef void(∗ pLog_callback)(BYTE, char ∗)

Functions

- void registerHotplugCallBk (pMessageHotplug pMsgHotplug)
- void registerLogCallBk (pSendDataLog pFSend, pReadDataLog pFRead)
- void emv_registerCallBk (pEMV_callBack pEMVf)
- void msr_registerCallBk (pMSR_callBack pMSRf)
- void msr_registerCallBkp (pMSR_callBackp pMSRf)
- void ctls_registerCallBk (pMSR_callBack pCTLSf)
- void ctls_registerCallBkp (pMSR_callBackp pCTLSf)
- void pin_registerCallBk (pPIN_callBack pPINf)
- void device_registerCameraCallBk (pCMR_callBack pCMRf)
- void device_registerCardStatusFrontSwitchCallBk (pCSFS_callBack pCSFSf)
- void comm_registerHTTPCallback (httpComm_callBack cBack)
- void comm_registerV4Callback (v4Comm_callBack cBack)
- char ∗ SDK_Version ()
- int setAbsoluteLibraryPath (const char ∗absoluteLibraryPath)
- int device_init ()
- int device_setCurrentDevice (int deviceType)
- int device_close ()
- void device_getIDGStatusCodeString (IN int returnCode, OUT char ∗despcrition)
- int device_isConnected ()
- int device_isAttached (int deviceType)
- int device_startTransaction (IN double amount, IN double amtOther, IN int type, IN const int _timeout, IN BYTE ∗tags, IN int tagsLen)

- void device_setTransactionExponent (int exponent)
- int device_activateTransaction (IN const int _timeout, IN BYTE *tags, IN int tagsLen)
- int device_cancelTransaction ()
- int device_getDriveFreeSpace (OUT int *free, OUT int *used)
- int device_listDirectory (IN char *directoryName, IN int directoryNameLen, IN int recursive, IN int onSD, OUT char *directory, IN_OUT int *directoryLen)
- int device_createDirectory (IN char *directoryName, IN int directoryNameLen)
- int device_deleteDirectory (IN char *dirName, IN int dirNameLen)
- int device_transferFile (IN char *fileName, IN int fileNameLen, IN BYTE *file, IN int fileLen)
- int device_deleteFile (IN char *fileName, IN int fileNameLen)
- int device_getFirmwareVersion (OUT char *firmwareVersion)
- int device_getFirmwareVersion_Len (OUT char *firmwareVersion, IN_OUT int *firmwareVersionLen)
- int device_pingDevice ()
- int device_controlUserInterface (IN BYTE *values)
- int device_controlIndicator (IN int indicator, IN int enable)
- int device_getCurrentDeviceType ()
- int device_SendDataCommandNEO (IN int cmd, IN int subCmd, IN BYTE *data, IN int dataLen, OUT BYTE *response, IN_OUT int *respLen)
- int device_enablePassThrough (int enablePassThrough)
- int device_enhancedPassthrough (IN BYTE *data, IN int dataLen)
- int device_setMerchantRecord (int index, int enabled, char *merchantID, char *merchantURL)
- int device_getMerchantRecord (IN int index, OUT BYTE *record)
- int device_getMerchantRecord_Len (IN int index, OUT BYTE *record, IN_OUT int *recordLen)
- int device_getTransactionResults (IDTMSRData *cardData)
- int device_calibrateParameters (BYTE delta)
- int config_getSerialNumber (OUT char *sNumber)
- int config_getSerialNumber_Len (OUT char *sNumber, IN_OUT int *sNumberLen)
- int device_getSDKWaitTime ()
- void device_setSDKWaitTime (int waitTime)
- int device_getThreadStackSize ()
- void device_setThreadStackSize (int threadSize)
- int ctls_startTransaction (IN double amount, IN double amtOther, IN int type, IN const int _timeout, IN BYTE *tags, IN int tagsLen)
- int ctls_activateTransaction (IN const int _timeout, IN BYTE *tags, IN int tagsLen)
- int ctls_cancelTransaction ()
- int ctls_retrieveApplicationData (IN BYTE *AID, IN int AIDLen, OUT BYTE *tlv, IN_OUT int *tlvLen)
- int ctls_setApplicationData (IN BYTE *tlv, IN int tlvLen)
- int ctls_removeApplicationData (IN BYTE *AID, IN int AIDLen)
- int ctls_removeAllApplicationData ()
- int ctls_retrieveAIDList (OUT BYTE *AIDList, IN_OUT int *AIDListLen)
- int ctls_retrieveTerminalData (OUT BYTE *tlv, IN_OUT int *tlvLen)
- int ctls_setTerminalData (IN BYTE *tlv, IN int tlvLen)
- int ctls_retrieveCAPK (IN BYTE *capk, IN int capkLen, OUT BYTE *key, IN_OUT int *keyLen)
- int ctls_setCAPK (IN BYTE *capk, IN int capkLen)
- int ctls_removeCAPK (IN BYTE *capk, IN int capkLen)
- int ctls_removeAllCAPK ()
- int ctls_retrieveCAPKList (OUT BYTE *keys, IN_OUT int *keysLen)
- int ctls_setConfigurationGroup (IN BYTE *tlv, IN int tlvLen)
- int ctls_getConfigurationGroup (IN int group, OUT BYTE *tlv, OUT int *tlvLen)
- int ctls_getAllConfigurationGroups (OUT BYTE *tlv, IN_OUT int *tlvLen)
- int ctls_removeConfigurationGroup (int group)
- int ctls_displayOnlineAuthResult (IN int statusCode, IN BYTE *TLV, IN int TLVLen)
- void emv_allowFallback (IN int allow)
- void emv_setAutoAuthenticateTransaction (IN int authenticate)
- void emv_setAutoCompleteTransaction (IN int complete)

- int emv_getAutoAuthenticateTransaction ()
- int emv_getAutoCompleteTransaction ()
- int emv_startTransaction (IN double amount, IN double amtOther, IN int exponent, IN int type, IN int timeout, IN BYTE *tags, IN int tagsLen, IN int forceOnline)
- int emv_activateTransaction (IN int timeout, IN BYTE *tags, IN int tagsLen, IN int forceOnline)
- int emv_authenticateTransaction (IN BYTE *updatedTLV, IN int updatedTLVLen)
- int emv_authenticateTransactionWithTimeout (IN int timeout, IN BYTE *updatedTLV, IN int updatedTLVLen)
- int emv_completeTransaction (IN int commError, IN BYTE *authCode, IN int authCodeLen, IN BYTE *iad, IN int iadLen, IN BYTE *tlvScripts, IN int tlvScriptsLen, IN BYTE *tlv, IN int tlvLen)
- int emv_cancelTransaction ()
- int emv_retrieveApplicationData (IN BYTE *AID, IN int AIDLen, OUT BYTE *tlv, IN_OUT int *tlvLen)
- int emv_setApplicationData (IN BYTE *name, IN int nameLen, IN BYTE *tlv, IN int tlvLen)
- int emv_setApplicationDataTLV (IN BYTE *tlv, IN int tlvLen)
- int emv_removeApplicationData (IN BYTE *AID, IN int AIDLen)
- int emv_removeAllApplicationData ()
- int emv_retrieveAIDList (OUT BYTE *AIDList, IN_OUT int *AIDListLen)
- int emv_retrieveTerminalData (OUT BYTE *tlv, IN_OUT int *tlvLen)
- int emv_setTerminalData (IN BYTE *tlv, IN int tlvLen)
- int emv_retrieveCAPK (IN BYTE *capk, IN int capkLen, OUT BYTE *key, IN_OUT int *keyLen)
- int emv_setCAPK (IN BYTE *capk, IN int capkLen)
- int emv_removeCAPK (IN BYTE *capk, IN int capkLen)
- int emv_removeAllCAPK ()
- int emv_retrieveCAPKList (OUT BYTE *keys, IN_OUT int *keysLen)
- int emv_retrieveExceptionList (OUT BYTE *exceptionList, IN_OUT int *exceptionListLen)
- int emv_setException (IN BYTE *exception, IN int exceptionLen)
- int emv_removeException (IN BYTE *exception, IN int exceptionLen)
- int emv_removeAllExceptions ()
- int emv_retrieveExceptionLogStatus (OUT BYTE *exceptionLogStatus, IN_OUT int *exceptionLogStatusLen)
- int emv_removeTransactionLog ()
- int emv_retrieveTransactionLogStatus (OUT BYTE *transactionLogStatus, IN_OUT int *transactionLogStatusLen)
- int emv_retrieveTransactionLog (OUT BYTE *transactionLog, IN_OUT int *transactionLogLen, IN_OUT int *remainingTransactionLogLen)
- int emv_getEMVKernelVersion (OUT char *version)
- int emv_getEMVKernelVersion_Len (OUT char *version, IN_OUT int *versionLen)
- int emv_getEMVKernelCheckValue (OUT BYTE *checkValue, IN_OUT int *checkValueLen)
- int emv_getEMVConfigurationCheckValue (OUT BYTE *checkValue, IN_OUT int *checkValueLen)
- int emv_retrieveCRL (OUT BYTE *list, IN_OUT int *lssLen)
- int emv_setCRL (IN BYTE *list, IN int lsLen)
- int emv_removeCRL (IN BYTE *list, IN int lsLen)
- int emv_removeAllCRL ()
- int lcd_resetInitialState ()
- int lcd_customDisplayMode (IN int enable)
- int lcd_setForeBackColor (IN BYTE *foreRGB, IN int foreRGBLen, IN BYTE *backRGB, IN int backRGBLen)
- int lcd_clearDisplay (IN BYTE control)
- int lcd_captureSignature (IN int timeout)
- int lcd_startSlideShow (IN char *files, IN int filesLen, IN int posX, IN int posY, IN int posMode, IN int touchEnable, IN int recursion, IN int touchTerminate, IN int delay, IN int loops, IN int clearScreen)
- int lcd_cancelSlideShow (OUT BYTE *statusCode, IN_OUT int *statusCodeLen)
- int lcd_setDisplayImage (IN char *file, IN int fileLen, IN int posX, IN int posY, IN int posMode, IN int touchEnable, IN int clearScreen)
- int lcd_setBackgroundImage (IN char *file, IN int fileLen, IN int enable)
- int lcd_displayText (IN int posX, IN int posY, IN int displayWidth, IN int displayHeight, IN int fontDesignation, IN int fontID, IN int screenPosition, IN char *displayText, OUT BYTE *graphicsID)

- int lcd_displayText_Len (IN int posX, IN int posY, IN int displayWidth, IN int displayHeight, IN int fontDesignation, IN int fontID, IN int screenPosition, IN char *displayText, OUT BYTE *graphicsID, IN_OUT int *graphicsIDLen)
- int lcd_displayParagraph (IN int posX, IN int posY, IN int displayWidth, IN int displayHeight, IN int fontDesignation, IN int fontID, IN int displayProperties, IN char *displayText)
- int lcd_displayButton (IN int posX, IN int posY, IN int buttonWidth, IN int buttonHeight, IN int fontDesignation, IN int fontID, IN int displayPosition, IN char *buttonLabel, IN int buttonTextColorR, IN int buttonTextColorG, IN int buttonTextColorB, IN int buttonBackgroundColorR, IN int buttonBackgroundColorG, IN int buttonBackgroundColorB, OUT BYTE *graphicsID)
- int lcd_displayButton_Len (IN int posX, IN int posY, IN int buttonWidth, IN int buttonHeight, IN int fontDesignation, IN int fontID, IN int displayPosition, IN char *buttonLabel, IN int buttonTextColorR, IN int buttonTextColorG, IN int buttonTextColorB, IN int buttonBackgroundColorR, IN int buttonBackgroundColorG, IN int buttonBackgroundColorB, OUT BYTE *graphicsID, IN_OUT int *graphicsIDLen)
- int lcd_createList (IN int posX, IN int posY, IN int numOfColumns, IN int numOfRows, IN int fontDesignation, IN int fontID, IN int verticalScrollArrowsVisible, IN int borderedListItems, IN int borderdScrollArrows, IN int touchSensitive, IN int automaticScrolling, OUT BYTE *graphicsID)
- int lcd_createList_Len (IN int posX, IN int posY, IN int numOfColumns, IN int numOfRows, IN int fontDesignation, IN int fontID, IN int verticalScrollArrowsVisible, IN int borderedListItems, IN int borderdScrollArrows, IN int touchSensitive, IN int automaticScrolling, OUT BYTE *graphicsID, IN_OUT int *graphicsIDLen)
- int lcd_addItemToList (IN BYTE *listGraphicsID, IN char *itemName, IN char *itemID, IN int selected)
- int lcd_getSelectedListItem (IN BYTE *listGraphicsID, OUT char *itemID)
- int lcd_getSelectedListItem_Len (IN BYTE *listGraphicsID, OUT char *itemID, IN_OUT int *itemIDLen)
- int lcd_clearEventQueue ()
- int lcd_getInputEvent (IN int timeout, OUT int *dataReceived, OUT BYTE *eventType, OUT BYTE *graphicsID, OUT BYTE *eventData)
- int lcd_getInputEvent_Len (IN int timeout, OUT int *dataReceived, OUT BYTE *eventType, IN_OUT int *eventTypeLen, OUT BYTE *graphicsID, IN_OUT int *graphicsIDLen, OUT BYTE *eventData, IN_OUT int *eventDataLen)
- int lcd_createInputField (IN BYTE *specs, IN int specsLen, OUT BYTE *graphicId)
- int lcd_createInputField_Len (IN BYTE *specs, IN int specsLen, OUT BYTE *graphicId, IN_OUT int *graphicIdLen)
- int lcd_getInputFieldValue (IN BYTE *graphicId, OUT BYTE *retData, IN_OUT int *retDataLen)
- int msr_cancelMSRSwipe ()
- int msr_startMSRSwipe (IN int _timeout)
- int msr_flushTrackData ()
- void parseMSRData (IN BYTE *resData, IN int resLen, IN_OUT IDTMSRData *cardData)
- int pin_getEncryptedOnlinePIN (IN int keyType, IN int timeout)
- int pin_getPAN (IN int getCSC, IN int timeout)
- int pin_promptCreditDebit (IN char *currencySymbol, IN int currencySymbolLen, IN char *displayAmount, IN int displayAmountLen, IN int timeout, OUT BYTE *retData, IN_OUT int *retDataLen)
- int ws_requestCSR (OUT RequestCSR *csr)
- int ws_loadSSLCert (IN char *name, IN int nameLen, IN char *dataDER, IN int dataDERLen)
- int ws_revokeSSLCert (IN char *name, IN int nameLen)
- int ws_deleteSSLCert (IN char *name, IN int nameLen)
- int ws_getCertChainType (OUT int *type)
- int ws_updateRootCertificate (IN char *name, IN int nameLen, IN char *dataDER, IN int dataDERLen, IN char *signature, IN int signatureLen)

12.16.1 Detailed Description

VP8800 API. VP8800 Global API methods.

12.16.2 Macro Definition Documentation

12.16.2.1 #define IN

INPUT parameter.

12.16.2.2 #define IN_OUT

INPUT / OUTPUT PARAMETER.

12.16.2.3 #define OUT

OUTPUT parameter.

12.16.3 Typedef Documentation

12.16.3.1 typedef void(∗ ftpComm_callBack)(int, int, int)

Define the comm callback function to get FTP file transfer status

It should be passed as a parameter in a FTP request, Signature (int, int, int) = response code, current block, total blocks RESPONSE CODES: 100 = FILE DOWNLOAD STARTED 101 = FILE BLOCK XX OF XX RECEIVED 102 = FILE DOWNLOAD COMPLETED 103 = FILE DOWNLOAD TERMINATED PREMATURELY

12.16.3.2 typedef void(∗ httpComm_callBack)(BYTE ∗, int)

Define the comm callback function to get the async url data

It should be registered using the comm_registerHTTPCallback

12.16.3.3 typedef void(∗ pCMR_callBack)(int, IDTCMRData ∗)

Define the camera callback function to get the image data

It should be registered using the device_registerCameraCallBk,

12.16.3.4 typedef void(∗ pCSFS_callBack)(BYTE status)

Define the card status and front switch callback function to get card and front switch status

It should be registered using the device_registerCardStatusFrontSwitchCallBk,

12.16.3.5 typedef void(∗ pEMV_callBack)(int, int, unsigned char ∗, int, IDTTransactionData ∗, EMV_Callback ∗, int)

Define the EMV callback function to get the transaction message/data/result.

It should be registered using the emv_registerCallBk,

12.16.3.6 typedef void(∗ pLog_callback)(BYTE, char ∗)

Define the log callback function to receive log messages.

12.16.3.7 typedef void(∗ pMessageHotplug)(int, int)

Define the USB hot-plug callback function to monitor the info when plug in/out the reader.

It should be registered using the registerHotplugCallBk, The first integer parameter is device type, and the second integer parameter is either 0: Device Plugged Out or 1: Device Plugged In

12.16.3.8 typedef void(∗ pMSR_callBack)(int, IDTMSRData)

Define the MSR callback function to get the MSR card data

It should be registered using the msr_registerCallBk, this callback function is for backward compatibility

12.16.3.9 typedef void(∗ pMSR_callBackp)(int, IDTMSRData ∗)

Define the MSR callback function to get pointer to the MSR card data

It should be registered using the msr_registerCallBk, this callback function is recommended instead of pMSR_callBack

12.16.3.10 typedef void(∗ pPIN_callBack)(int, IDTPINData ∗)

Define the PINPad callback function to get the input PIN Pad data

It should be registered using the pin_registerCallBk,

12.16.3.11 typedef void(∗ pReadDataLog)(unsigned char ∗, int)

Define the read response callback function to monitor the reading response from the reader.

It should be registered using the registerLogCallBk,

12.16.3.12 typedef void(∗ pSendDataLog)(unsigned char ∗, int)

Define the send command callback function to monitor the sending command into the reader.

It should be registered using the registerLogCallBk,

12.16.3.13 typedef void(∗ v4Comm_callBack)(BYTE, BYTE, BYTE ∗, int)

Define the comm callback function to receive the V4 Protocol packets received by the device from an external source (IP/USB/RS-232) It should be registered using the comm_registerV4Callback, Data callback will contain command, sub-command, and data from V4 packet

12.16.4 Function Documentation

12.16.4.1 void comm_registerHTTPCallback (httpComm_callBack *cBack*)

Register Comm HTTP Async Callback

Parameters

cBack	? HTTP Comm callback

12.16.4.2 void comm_registerV4Callback (v4Comm_callBack *cBack*)

Register External V4 Protocol commands Callback

Parameters

cBack	? V4 Protocol Comm callback

12.16.4.3 int config_getSerialNumber (OUT char * *sNumber*)

DEPRECATED : please use config_getSerialNumber_Len(OUT char* sNumber, IN_OUT int *sNumberLen)

Polls device for Serial Number

Parameters

sNumber	Returns Serial Number; needs to have at least 64 bytes of memory

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString

12.16.4.4 int config_getSerialNumber_Len (OUT char * *sNumber,* IN_OUT int * *sNumberLen*)

Polls device for Serial Number

Parameters

sNumber	Returns Serial Number
sNumberLen	Length of Serial Number

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString

12.16.4.5 int ctls_activateTransaction (IN const int *_timeout,* IN BYTE * *tags,* IN int *tagsLen*)

Start CTLS Transaction Request

Authorizes the CTLS 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 stream. Example, tag 9F0C with amount 0x000000000100 would be 0x9F0C06000000000100
tagsLen	The length of tags data buffer.

>>>>>IMPORTANT: parameters for amount and amtOther MUST BE PASSED AS A DOUBLE VALUE WITH DECIMAL POINT. Example, do not pass 1, but instead pass 1.0 or 1.00. Otherwise, results will be unpredictable

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString() Note: if auto poll is on, it will returm the error IDG_P2_STATUS_CODE_COMMAND_NOT_ALLOWED

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

12.16.4.6 int ctls_cancelTransaction ( )

Cancel EMV Transaction

Cancels the currently executing EMV transaction.

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.16.4.7 int ctls_displayOnlineAuthResult (IN int *statusCode,* IN BYTE * *TLV,* IN int *TLVLen*)

Display Online Authorization Result Use this command to display the status of an online authorization request on the reader's display (OK or NOT OK). Use this command after the reader sends an online request to the issuer.

Parameters

statusCode	1 = OK, 0 = NOT OK, 2 = ARC response 89 for Interac
TLV	Optional TLV for AOSA
TLVLen	TLV Length

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.16.4.8 int ctls_getAllConfigurationGroups (OUT BYTE * *tlv*, IN_OUT int * *tlvLen*)

Retrieve All Configuration Groups

Returns all the Configuration Groups installed on the terminal for CTLS

Parameters

tlv	The TLV elements data
tlvLen	the length of tlv data buffer.

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.16.4.9 int ctls_getConfigurationGroup (IN int *group*, OUT BYTE * *tlv*, OUT int * *tlvLen*)

Get Configuration Group

Retrieves the Configuration for the specified Group.

Parameters

group	Configuration Group
tlv	return data
tlvLen	the length of tlv data buffer

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.16.4.10 void ctls_registerCallBk (pMSR_callBack *pCTLSf*)

To register the ctls callback function to get the MSR card data. (Pass NULL to disable the callback.)

12.16.4.11 void ctls_registerCallBkp (pMSR_callBackp *pCTLSf*)

To register the ctls callback function to get the MSR card data pointer. (Pass NULL to disable the callback.)

12.16.4.12 int ctls_removeAllApplicationData ( )

Remove All Application Data

Removes all the Application Data

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.16.4.13 int ctls_removeAllCAPK ( )

Remove All Certificate Authority Public Key

Removes all the CAPK

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.16.4.14 int ctls_removeApplicationData (IN BYTE * *AID*, IN int *AIDLen*)

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
AIDLen	the length of AID data buffer

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.16.4.15 int ctls_removeCAPK (IN BYTE * *capk*, IN int *capkLen*)

Remove Certificate Authority Public Key

Removes the CAPK as specified by the RID/Index

Parameters

capk	6 byte CAPK = 5 bytes RID + 1 byte INDEX
capkLen	the length of capk data buffer

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.16.4.16 int 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	Values can be parsed with device_getIDGStatusCodeString()

12.16.4.17 int ctls_retrieveAIDList (OUT BYTE * *AIDList*, IN_OUT int * *AIDListLen*)

Retrieve AID list

Returns all the AID names installed on the terminal for CTLS. .

Parameters

AIDList	array of AID name byte arrays
AIDListLen	the length of AIDList array buffer

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.16.4.18 int ctls_retrieveApplicationData (IN BYTE * *AID*, IN int *AIDLen*, OUT BYTE * *tlv*, IN_OUT int * *tlvLen*)

Retrieve Application Data by AID

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

Parameters

AID	Name of ApplicationID. Must be between 5 and 16 bytes
AIDLen	the length of AID data buffer.
tlv	The TLV elements of the requested AID
tlvLen	the length of tlv data buffer.

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.16.4.19 int ctls_retrieveCAPK (IN BYTE * *capk*, IN int *capkLen*, OUT BYTE * *key*, IN_OUT int * *keyLen*)

Retrieve Certificate Authority Public Key

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

Parameters

capk	6 bytes CAPK = 5 bytes RID + 1 byte Index
capkLen	the length of capk data buffer
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.
keyLen	the length of key data buffer •

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.16.4.20 int ctls_retrieveCAPKList (OUT BYTE * *keys*, IN_OUT int * *keysLen*)

Retrieve the Certificate Authority Public Key list

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

Parameters

keys	[key1][key2]...[keyn], each key 6 bytes where key = 5 bytes RID + 1 byte index
keysLen	the length of keys data buffer

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.16.4.21 int ctls_retrieveTerminalData (OUT BYTE * *tlv*, IN_OUT int * *tlvLen*)

Retrieve Terminal Data

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

Parameters

tlv	Response returned as a TLV
tlvLen	the length of tlv data buffer

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.16.4.22 int ctls_setApplicationData (IN BYTE * *tlv*, IN int *tlvLen*)

Set Application Data by AID

Sets the Application Data for CTLS as specified by TLV data

Parameters

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

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

Parameters

tlvLen	the length of tlv data buffer

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.16.4.23 int ctls_setCAPK (IN BYTE * *capk*, IN int *capkLen*)

Set Certificate Authority Public Key

Sets the CAPK as specified by the CAKey structure

Parameters

capk	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.
capkLen	the length of capk data buffer

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.16.4.24 int ctls_setConfigurationGroup (IN BYTE * *tlv*, IN int *tlvLen*)

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
tlvLen	the length of tlv data buffer

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.16.4.25 int ctls_setTerminalData (IN BYTE * *tlv*, IN int *tlvLen*)

Set Terminal Data

Sets the Terminal Data for CTLS as specified by the TLV. The first TLV must be Configuration Group Number (Tag FFE4). The terminal global data is group 0, so the first TLV would be FFE40100. Other groups can be defined using this method (1 or greater), and those can be retrieved with ctls_getConfigurationGroup(int group), and deleted with ctls_removeConfigurationGroup(int group). You cannot delete group 0.

Parameters

tlv	TerminalData configuration file
tlvLen	the length of tlv data buffer

Return values

RETURN_CODE	Values can be parsed with device_getIDGStatusCodeString()

12.16.4.26 int ctls_startTransaction (IN double *amount*, IN double *amtOther*, IN int *type*, IN const int *_timeout*, IN BYTE * *tags*, IN int *tagsLen*)

Start CTLS Transaction Request

Authorizes the CTLS transaction for an ICC card

The tags will be returned in the callback routine.

Parameters

amount	Transaction amount value (tag value 9F02) - SEE IMPORTANT NOTE BELOW
amtOther	Other amount value, if any (tag value 9F03) - SEE IMPORTANT NOTE BELOW
type	Transaction type (tag value 9C).
timeout	Timeout value in seconds.
tags	Any other tags to be included in the request. Passed as TLV stream. Example, tag 9F0C with amount 0x000000000100 would be 0x9F0C06000000000100 If tags 9F02 (amount),9-F03 (other amount), or 9C (transaction type) are included, they will take priority over these values supplied as individual parameters to this method.
tagsLen	The length of tags data buffer.

>>>>>IMPORTANT: parameters for amount and amtOther MUST BE PASSED AS A DOUBLE VALUE WITH DECIMAL POINT. Example, do not pass 1, but instead pass 1.0 or 1.00. Otherwise, results will be unpredictable

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString() Note: if auto poll is on, it will returm the error IDG_P2_STATUS_CODE_COMMAND_NOT_ALLOWED

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

12.16.4.27 int device_activateTransaction (IN const int *_timeout,* IN BYTE * *tags,* IN int *tagsLen*)

Start Transaction Request

Authorizes the transaction for an MSR/CTLS/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 a TLV. Example, tag 9F0C with amount 0x000000000100 would be 0x9F0C06000000000100 Be sure to include 9F02 (amount)and9-C (transaction type).
tagsLen	The length of tags data buffer.

>>>>>IMPORTANT: parameters for amount and amtOther MUST BE PASSED AS A DOUBLE VALUE WITH DECIMAL POINT. Example, do not pass 1, but instead pass 1.0 or 1.00. Otherwise, results will be unpredictable

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString() Note: if auto poll is on, it will returm the error IDG_P2_STATUS_CODE_COMMAND_NOT_ALLOWED

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 device_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

12.16.4.28 int device_calibrateParameters (BYTE *delta*)

Calibrate reference parameters

Parameters

delta	Delta value (0x02 standard default value)

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.16.4.29 int device_cancelTransaction ( )

Cancel Transaction

Cancels the currently executing transaction.

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.16.4.30 int device_close ( )

Close the device

Returns

RETURN_CODE: 0: success, 0x0A: failed

12.16.4.31 int device_controlIndicator (IN int *indicator,* IN int *enable*)

Control Indicators

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

Parameters

indicator	description as follows: • 00h: ICC LED • 01h: Blue MSR • 02h: Red MSR • 03h: Green MSR
enable	TRUE = ON, FALSE = OFF

Returns

success or error code. Values can be parsed with device_getResponseCodeString

See Also

ErrorCode

12.16.4.32 int device_controlUserInterface (IN BYTE * *values*)

Control User Interface

Controls the User Interface: Display, Beep, LED

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

```
- 04h: Four short beeps
- 05h: One long beep of 200 ms
- 06h: One long beep of 400 ms
- 07h: One long beep of 600 ms
- 08h: One long beep of 800 ms
Byte[2] = LED Number
- 00h: LED 0 (Power LED) 01h: LED 1
- 02h: LED 2
- 03h: LED 3
- FFh: All LEDs
Byte[3] = LED Status
- 00h: LED Off
- 01h: LED On
```

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.16.4.33 int device_createDirectory (IN char * *directoryName*, IN int *directoryNameLen*)

Create Directory This command adds a subdirectory to the indicated path.

Parameters

directoryName	Directory Name. The data for this command is a ASCII string with the complete path and directory name you want to create. You do not need to specify the root directory. Indicate subdirectories with a forward slash (/).
directoryNameLen	Directory Name Length.

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.16.4.34 int device_deleteDirectory (IN char * *dirName*, IN int *dirNameLen*)

Delete Directory This command deletes an empty directory. For NEO 2 devices, it will delete the directory even the directory is not empty.

Parameters

dirName	Complete path of the directory you want to delete. You do not need to specify the root directory. Indicate subdirectories with a forward slash (/). For NEO 2 devices, to delete the root directory, simply pass "" with 0 for dirNameLen.
dirNameLen	Directory Name Length.

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.16.4.35 int device_deleteFile (IN char * *fileName*, IN int *fileNameLen*)

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 (/).
filenameLen	File Name Length.

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.16.4.36 int device_enablePassThrough (int *enablePassThrough*)

Start Remote Key Injection

Starts a remote key injection request with IDTech RKI servers. This function is reserved and not implemented.

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

Enable Pass Through

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

Parameters

enablePass-Through	1 = pass through ON, 0 = pass through OFF

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.16.4.37 int device_enhancedPassthrough (IN BYTE * *data*, IN int *dataLen*)

Enables pass through mode for ICC. Required when direct ICC commands are required (power on/off ICC, exchange APDU)

Parameters

data	The data includes Poll Timeout, Flags, Contact Interface to Use, Beep Indicator, LED Status, and Display Strings.
dataLen	length of data

Returns

success or error code. Values can be parsed with device_getIDGStatusCodeString

See Also

ErrorCode

12.16.4.38 int device_getCurrentDeviceType ( )

Get current active device type

Returns

: return the device type defined as DEVICE_TYPE in the IDTDef.h

12.16.4.39 int device_getDriveFreeSpace (OUT int * *free*, OUT int * *used*)

Drive Free Space This command returns the free and used disk space on the flash drive.

Parameters

free	Free bytes available on device
used	Used bytes on on device

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.16.4.40 int device_getFirmwareVersion (OUT char * *firmwareVersion*)

DEPRECATED : please use device_getFirmwareVersion_Len(OUT char* firmwareVersion, IN_OUT int *firmwareVersionLen)

Polls device for Firmware Version

Parameters

firmwareVersion	Response returned of Firmware Version; needs to have at least 128 bytes of memory

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.16.4.41 int device_getFirmwareVersion_Len (OUT char * *firmwareVersion,* IN_OUT int * *firmwareVersionLen*)

Polls device for Firmware Version

Parameters

firmwareVersion	Response returned of Firmware Version
firmwareVersionLen	Length of Firmware Version

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.16.4.42 void device_getIDGStatusCodeString (IN int *returnCode,* OUT char * *despcrition*)

Review the return code description.

Parameters

returnCode	the response result.
description	

Return values

the	string for description of response result

- 0: "no error, beginning task";
- 1: "no response from reader";
- 2: "invalid response data";
  - 01: " Incorrect Header Tag";

- 02: " Unknown Command";
- 03: " Unknown Sub-Command";
- 04: " CRC Error in Frame";
- 05: " Incorrect Parameter";
- 06: " Parameter Not Supported";
- 07: " Mal-formatted Data";
- 08: " Timeout";
- 0A: " Failed / NACK";
- 0B: " Command not Allowed";
- 0C: " Sub-Command not Allowed";
- 0D: " Buffer Overflow (Data Length too large for reader buffer)";
- 0E: " User Interface Event";
- 10: " Need clear firmware(apply in boot loader only)";
- 11: " Communication type not supported, VT-1, burst, etc. Need encrypted firmware (apply in boot loader only)";
- 12: " Secure interface is not functional or is in an intermediate state.";
- 13: " Data field is not mod 8";
- 14: " Pad 0x80 not found where expected";
- 15: " Specified key type is invalid";
- 16: " Could not retrieve key from the SAM (InitSecureComm)";
- 17: " Hash code problem";
- 18: " Could not store the key into the SAM (InstallKey)";
- 19: " Frame is too large";
- 1A: " Unit powered up in authentication state but POS must resend the InitSecureComm command";
- 1B: " The EEPROM may not be initialized because SecCommInterface does not make sense";
- 1C: " Problem encoding APDU Module-Specific Status Codes ";
- 20: " Unsupported Index (ILM) SAM Transceiver error - problem communicating with the SAM (Key Mgr)";
- 21: " Unexpected Sequence Counter in multiple frames for single bitmap (ILM)Length error in data returned from the SAM (Key Mgr) - 22: " Improper bit map (ILM)"; - 23: " Request Online Authorization"; - 24: " ViVOCard3 raw data read successful"; - 25: " Message index not available (ILM) ViVOcomm activate transaction card type (ViVOcomm)"; - 26: " Version Information Mismatch (ILM)"; - 27: " Not sending commands in correct index message index (ILM)"; - 28: " Time out or next expected message not received (ILM)"; - 29: " ILM languages not available for viewing (ILM)"; - 2A: " Other language not supported (ILM)"; - 41: " from 41 to 4F, Module-specific errors for Key Manager";
- 50: " Auto-Switch OK";
- 51: " Auto-Switch failed";
- 70: " Antenna Error 80h Use another card";
- 81: " Insert or swipe card";
- 90: " Data encryption Key does not exist";
- 91: " Data encryption Key KSN exhausted";

12.16.4.43 int device_getMerchantRecord (IN int *index,* OUT BYTE * *record*)

DEPRECATED : please use device_getMerchantRecord_Len(IN int index, OUT BYTE * record, IN_OUT int *recordLen)

Get Merchant Record

Gets the merchant record for the device.

Parameters

index	Merchant Record index, valid values 1-6
record;	needs to have at least 99 bytes of memory response data from reader. Merchant Record Index: 1 byte enabled: 1 byte Merchant ID: 32 bytes Length of Merchant URL: 1 byte Merchant URL: 64 bytes

Returns

success or error code. Values can be parsed with device_getIDGStatusCodeString()

See Also

ErrorCode

12.16.4.44 int device_getMerchantRecord_Len (IN int *index*, OUT BYTE * *record*, IN_OUT int * *recordLen*)

Get Merchant Record

Gets the merchant record for the device.

Parameters

index	Merchant Record index, valid values 1-6
record	response data from reader. Merchant Record Index: 1 byte enabled: 1 byte Merchant ID: 32 bytes Length of Merchant URL: 1 byte Merchant URL: 64 bytes
recordLen	Length of record

Returns

success or error code. Values can be parsed with device_getIDGStatusCodeString()

See Also

ErrorCode

12.16.4.45 int device_getSDKWaitTime ( )

Get SDK Wait Time

Get the SDK wait time for transactions

Returns

SDK wait time in seconds

12.16.4.46 int device_getThreadStackSize ( )

Get Thread Stack Size

Get the stack size setting for newly created threads

Returns

Thread Stack Size

12.16.4.47 int device_getTransactionResults (IDTMSRData * *cardData*)

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

Parameters

cardData	The transaction results

Returns

success or error code. Values can be parsed with device_getResponseCodeString

See Also

ErrorCode

12.16.4.48 int device_init ( )

Initial the device by USB

It will detect the device and trying connect.

The connect status can be checked by device_isConnected().

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.16.4.49 int device_isAttached (int *deviceType*)

Check if the device is attached to the USB port The function device_init() must be called before this function.

Parameters

deviceType,the	device type of the USB device

Returns

1 if the device is attached, or 0 if the device is not attached

12.16.4.50 int device_isConnected ( )

Check the device conntected status

Returns

DEVICE_DISCONNECT=0, or DEVICE_CONNECTED = 1

12.16.4.51 int device_listDirectory (IN char * *directoryName,* IN int *directoryNameLen,* IN int *recursive,* IN int *onSD,* OUT char * *directory,* IN_OUT int * *directoryLen*)

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
directoryName-Len	Directory Name Length. If null, root directory is listed
recursive	Included sub-directories
onSD	TRUE = use flash storage The returned directory information The returned directory information length

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.16.4.52 int device_pingDevice ( )

Ping Device

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

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.16.4.53 void device_registerCameraCallBk (pCMR_callBack *pCMRf*)

To register the camera callback function to get the image data. (Pass NULL to disable the callback.)

12.16.4.54 void device_registerCardStatusFrontSwitchCallBk (pCSFS_callBack *pCSFSf*)

To register the card status and front switch callback function to get status. (Pass NULL to disable the callback.)

12.16.4.55 int device_SendDataCommandNEO (IN int *cmd*, IN int *subCmd*, IN BYTE * *data*, IN int *dataLen*, OUT BYTE * *response*, IN_OUT int * *respLen*)

Send a Command to device

Sends a command to the device .

Parameters

cmd	buffer of command to execute.
cmdLen,the	length of the buffer cmd.
data	buffer of IDG command data.
dataLen,the	length of the buffer data.
response	Response data
respLen,the	length of Response data

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString() Send a Command to NEO device

Sends a command to the NEO device .

Parameters

cmd	command to execute.
subCmd,sub	command to execute.
data	buffer of NEO command data.
dataLen,the	length of the buffer data.
response	Response data
respLen,the	length of Response data

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.16.4.56 int device_setCurrentDevice (int *deviceType*)

Sets the current device to talk to

The connect status can be checked by device_isConnected().

Parameters

deviceType	Device to connect to
	
	`enum DEVICE_TYPE{`
	`IDT_DEVICE_UNKNOWN=0,`
	`IDT_DEVICE_AUGUSTA_HID,`
	`IDT_DEVICE_AUGUSTA_KB,`
	`IDT_DEVICE_AUGUSTA_S_HID,`
	`IDT_DEVICE_AUGUSTA_S_KB,`
	`IDT_DEVICE_AUGUSTA_S_TTK_HID,`
	`IDT_DEVICE_SPECTRUM_PRO,`
	`IDT_DEVICE_MINISMART_II,`
	`IDT_DEVICE_L100,`
	`IDT_DEVICE_UNIPAY,`
	`IDT_DEVICE_UNIPAY_I_V,`
	`IDT_DEVICE_VP3300_AJ,`
	`IDT_DEVICE_KIOSK_III,`
	`IDT_DEVICE_KIOSK_III_S,`
	`IDT_DEVICE_PIP_READER,`
	`IDT_DEVICE_VENDI,`
	`IDT_DEVICE_VP3300_USB,`
	`IDT_DEVICE_UNIPAY_I_V_TTK,`
	`IDT_DEVICE_VP3300_BT,`
	`IDT_DEVICE_VP8800,`
	`IDT_DEVICE_SREDKEY2_HID,`
	`IDT_DEVICE_SREDKEY2_KB,`
	`IDT_DEVICE_NEO2,`
	`IDT_DEVICE_MINISMART_II_COM = IDT_DEVICE_NEO2+5,`
	`IDT_DEVICE_SPECTRUM_PRO_COM,`
	`IDT_DEVICE_KIOSK_III_COM,`
	`IDT_DEVICE_KIOSK_III_S_COM,`
	`IDT_DEVICE_VP3300_COM,`
	`IDT_DEVICE_NEO2_COM,`
	`IDT_DEVICE_MAX_DEVICES = IDT_DEVICE_NEO2_COM+5`
	`};`

Returns

RETURN_CODE: 1: success, 0: failed

12.16.4.57 int device_setMerchantRecord (int *index,* int *enabled,* char * *merchantID,* char * *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 device_getIDGStatusCodeString()

12.16.4.58 void device_setSDKWaitTime (int *waitTime*)

Set SDK Wait Time

Set the SDK wait time for transactions

Parameters

waitTime	The SDK wait time for transaction in seconds

12.16.4.59 void device_setThreadStackSize (int *threadSize*)

Set Thread Stack Size

Set the stack size setting for newly created threads

12.16.4.60 void device_setTransactionExponent (int *exponent*)

Sets the transaction exponent to be used with device_startTransaction. Default value is 2

Parameters

exponent,The	exponent to use when calling device_startTransaction

12.16.4.61 int device_startTransaction (IN double *amount,* IN double *amtOther,* IN int *type,* IN const int *_timeout,* IN BYTE * *tags,* IN int *tagsLen*)

Start Transaction Request

Authorizes the transaction for an MSR/CTLS/ICC card

The tags will be returned in the callback routine.

Parameters

amount	Transaction amount value (tag value 9F02) - SEE IMPORTANT NOTE BELOW
amtOther	Other amount value, if any (tag value 9F03) - SEE IMPORTANT NOTE BELOW
type	Transaction type (tag value 9C).
timeout	Timeout value in seconds.
tags	Any other tags to be included in the request. Passed as TLV. Example, tag 9F0C with amount 0x000000000100 would be 0x9F0C06000000000100 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.
tagsLen	The length of tags data buffer.

>>>>>IMPORTANT: parameters for amount and amtOther MUST BE PASSED AS A DOUBLE VALUE WITH DECIMAL POINT. Example, do not pass 1, but instead pass 1.0 or 1.00. Otherwise, results will be unpredictable

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString() Note: if auto poll is on, it will returm the error IDG_P2_STATUS_CODE_COMMAND_NOT_ALLOWED

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 device_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

12.16.4.62 int device_transferFile (IN char * *fileName,* IN int *fileNameLen,* IN BYTE * *file,* IN int *fileLen*)

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 (/).
filenameLen	File Name Length.
file	The data file.
fileLen	File Length.

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.16.4.63 int emv_activateTransaction (IN int *timeout,* IN BYTE * *tags,* IN int *tagsLen,* IN int *forceOnline*)

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	Tags to be included in the request. Passed as a TLV stream. Example, tag 9F0C with amount 0x000000000100 would be 0x9F0C06000000000100
tagsLen	Length of tags
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 = DFEE1A079F029F36959F37

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString >>>>>IMPORTANT: parameters for amount and amtOther MUST BE PASSED AS A DOUBLE VALUE WITH DECIMAL POINT. Example, do not pass 1, but instead pass 1.0 or 1.00. Otherwise, results will be unpredictable

12.16.4.64 void emv_allowFallback (IN int *allow*)

Allow fallback for EMV transactions. Default is TRUE

Parameters

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

12.16.4.65 int emv_authenticateTransaction (IN BYTE * *updatedTLV,* IN int *updatedTLVLen*)

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.

Parameters

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, 9F37 to be included in response = DFEE1A079F029F36959F37

updatedTLVLen	

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.16.4.66 int emv_authenticateTransactionWithTimeout (IN int *timeout,* IN BYTE * *updatedTLV,* IN int *updatedTLVLen*)

Authenticate EMV Transaction Request with Timeout

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.

Parameters

timeout	Timeout value in seconds.
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, 9F37 to be included in response = DFEE1-A079F029F36959F37
updatedTLVLen	

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.16.4.67 int emv_cancelTransaction ( )

Cancel EMV Transaction

Cancels the currently executing EMV transaction.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.16.4.68 int emv_completeTransaction (IN int *commError,* IN BYTE * *authCode,* IN int *authCodeLen,* IN BYTE * *iad,* IN int *iadLen,* IN BYTE * *tlvScripts,* IN int *tlvScriptsLen,* IN BYTE * *tlv,* IN int *tlvLen*)

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(1) if host was unreachable, or FALSE(0) 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
authCodeLen	the length of authCode
iad	Issuer Authentication Data, if any. Example 0x11223344556677883030 (tag value 91).
iadLen	the length of iadLen
tlvScripts	71/72 scripts, if any
tlvScriptsLen	the length of tlvScriptsLen
tlv	Additional TLV data to return with transaction results (if any)
tlvLen	the length of tlv

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

12.16.4.69 int emv_getAutoAuthenticateTransaction ( )

Gets auto authenticate value for EMV transactions.

Returns

RETURN_CODE: TRUE = auto authenticate, FALSE = manually authenticate

12.16.4.70 int emv_getAutoCompleteTransaction ( )

Gets auto complete value for EMV transactions.

Returns

RETURN_CODE: TRUE = auto complete, FALSE = manually complete

12.16.4.71 int emv_getEMVConfigurationCheckValue (OUT BYTE * *checkValue,* IN_OUT int * *checkValueLen*)

Get EMV Kernel configuration check value info

Parameters

checkValue	Response returned of Kernel configuration check value info
checkValueLen	the length of checkValue

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.16.4.72 int emv_getEMVKernelCheckValue (OUT BYTE * *checkValue,* IN_OUT int * *checkValueLen*)

Get EMV Kernel check value info

Parameters

checkValue	Response returned of Kernel check value info
checkValueLen	the length of checkValue

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.16.4.73 int emv_getEMVKernelVersion (OUT char * *version*)

DEPRECATED : please use emv_getEMVKernelVersion_Len(OUT char* version, IN_OUT int *versionLen)

Polls device for EMV Kernel Version

Parameters

version	Response returned of Kernel Version; needs to have at least 128 bytes of memory.

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.16.4.74 int emv_getEMVKernelVersion_Len (OUT char * *version,* IN_OUT int * *versionLen*)

Polls device for EMV Kernel Version

Parameters

version	Response returned of Kernel Version
versionLen	Length of version

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.16.4.75 void emv_registerCallBk (pEMV_callBack *pEMVf*)

To register the emv callback function to get the EMV processing response. (Pass NULL to disable the callback.)

12.16.4.76 int emv_removeAllApplicationData ( )

Remove All Application Data

Removes all the Application Data

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.16.4.77 int emv_removeAllCAPK ( )

Remove All Certificate Authority Public Key

Removes all the CAPK

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.16.4.78 int emv_removeAllCRL ( )

Remove All Certificate Revocation List Entries

Removes all CRLEntry entries

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.16.4.79 int emv_removeAllExceptions ( )

Remove All EMV Exceptions

Removes all entries from the EMV Exception List

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.16.4.80 int emv_removeApplicationData (IN BYTE * *AID*, IN int *AIDLen*)

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
AIDLen	the length of AID data buffer

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.16.4.81 int emv_removeCAPK (IN BYTE * *capk*, IN int *capkLen*)

Remove Certificate Authority Public Key

Removes the CAPK as specified by the RID/Index

Parameters

capk	6 byte CAPK = 5 bytes RID + 1 byte INDEX
capkLen	the length of capk data buffer

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.16.4.82 int emv_removeCRL (IN BYTE * *list*, IN int *lsLen*)

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
lssLen	the length of list data buffer

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.16.4.83 int emv_removeException (IN BYTE * *exception*, IN int *exceptionLen*)

Remove EMV Exception

Removes an entry to the EMV Exception List

Parameters

exception	EMV Exception entry containing the PAN and Sequence Number where [Exception] is 12 bytes: [1 byte Len][10 bytes PAN][1 byte Sequence Number] PAN, in compressed numeric, padded with F if required (example 0x5413339000001596FFFF)
exceptionLen	The length of the exception.

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.16.4.84 int emv_removeTransactionLog ( )

Clear Transaction Log

Clears the transaction log.

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.16.4.85 int emv_retrieveAIDList (OUT BYTE * *AIDList*, IN_OUT int * *AIDListLen*)

Retrieve AID list

Returns all the AID names installed on the terminal for CTLS. .

Parameters

AIDList	array of AID name byte arrays
AIDListLen	the length of AIDList array buffer

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.16.4.86 int emv_retrieveApplicationData (IN BYTE * *AID*, IN int *AIDLen*, OUT BYTE * *tlv*, IN_OUT int * *tlvLen*)

Retrieve Application Data by AID

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

Parameters

AID	Name of ApplicationID. Must be between 5 and 16 bytes
AIDLen	the length of AID data buffer.
tlv	The TLV elements of the requested AID
tlvLen	the length of tlv data buffer.

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.16.4.87 int emv_retrieveCAPK (IN BYTE * *capk*, IN int *capkLen*, OUT BYTE * *key*, IN_OUT int * *keyLen*)

Retrieve Certificate Authority Public Key

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

Parameters

capk	6 bytes CAPK = 5 bytes RID + 1 byte Index
capkLen	the length of capk data buffer
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.

keyLen	the length of key data buffer •

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.16.4.88 int emv_retrieveCAPKList (OUT BYTE * *keys*, IN_OUT int * *keysLen*)

Retrieve the Certificate Authority Public Key list

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

Parameters

keys	[key1][key2]...[keyn], each key 6 bytes where key = 5 bytes RID + 1 byte index
keysLen	the length of keys data buffer

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.16.4.89 int emv_retrieveCRL (OUT BYTE * *list*, IN_OUT int * *lssLen*)

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
lssLen	the length of list data buffer

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.16.4.90 int emv_retrieveExceptionList (OUT BYTE * *exceptionList*, IN_OUT int * *exceptionListLen*)

Retrieve the EMV Exception List

Returns the EMV Exception entries on the terminal.

Parameters

exceptionList	[Exception1][Exception2]...[Exceptionn], where [Exception] is 12 bytes: [1 byte Len][10 bytes PAN][1 byte Sequence Number]
exceptionListLen	The length of the exception list.

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.16.4.91 int emv_retrieveExceptionLogStatus (OUT BYTE * *exceptionLogStatus,* IN_OUT int * *exceptionLogStatusLen*)

Get EMV Exception Log Status

This command returns information about the EMV Exception log. The version number, record size, and number of records contained in the file are returned.

Parameters

exceptionLogStatus	12 bytes returned • bytes 0-3 = Version Number • bytes 4-7 = Number of records • bytes 8-11 = Size of record
exceptionLogStatusLen	The length of the exception log status.

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.16.4.92 int emv_retrieveTerminalData (OUT BYTE * *tlv,* IN_OUT int * *tlvLen*)

Retrieve Terminal Data

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

Parameters

tlv	Response returned as a TLV
tlvLen	the length of tlv data buffer

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.16.4.93 int emv_retrieveTransactionLog (OUT BYTE * *transactionLog,* IN_OUT int * *transactionLogLen,* IN_OUT int * *remainingTransactionLogLen*)

Get Transaction Log Record

Retrieves oldest transaction record on the Transaction Log. At successful completion, the oldest transaction record is deleted from the transaction log

Parameters

transactionLog	Transaction Record
transactionLogLen	The length of the transaction log.
remainingTransactionLogLen	Number of records remaining on the transaction log

Length	Description	Type
4	Transaction Log State (TLS)	Enum (4-byte number, LSB first), SENT ONLINE = 0, NOT SENT = 1
4	Transaction Log Content (TLC)	Enum (4-byte number, LSB first), BATCH = 0, OFFLINE ADVICE = 1, ONLINEADVICE = 2, REVERSAL = 3
4	AppExpDate	unsigned char [4]
3	AuthRespCode	unsigned char [3]
3	MerchantCategoryCode	unsigned char [3]
16	MerchantID	unsigned char [16]
2	PosEntryMode	unsigned char [2]
9	TermID	unsigned char [9]
3	AIP	unsigned char [3]
3	ATC	unsigned char [3]
33	IssuerAppData	unsigned char [33]
6	TVR	unsigned char [6]
3	TSI	unsigned char [3]
11	Pan	unsigned char [11]
2	PanSQNCNum	unsigned char [2]
3	TermCountryCode	unsigned char [3]
7	TranAmount	unsigned char [7]
3	TranCurCode	unsigned char [3]
4	TranDate	unsigned char [4]
2	TranType	unsigned char [2]
9	IFDSerialNum	unsigned char [9]
12	AcquirerID	unsigned char [12]
2	CID	unsigned char [2]
9	AppCryptogram	unsigned char [9]
5	UnpNum	unsigned char [5]
7	AmountAuth	unsigned char [7]
4	AppEffDate	unsigned char [4]
4	CVMResults	unsigned char [4]
129	IssScriptResults	unsigned char [129]
4	TermCap	unsigned char [4]
2	TermType	unsigned char [2]
20	Track2	unsigned char [20]
4	TranTime	unsigned char [4]
7	AmountOther	unsigned char [7]
1	Unused	Unsigned char [1]

Returns

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

12.16.4.94 int emv_retrieveTransactionLogStatus (OUT BYTE * *transactionLogStatus,* IN_OUT int * *transactionLogStatusLen*)

Get Transaction Log Status

This command returns information about the EMV transaction log. The version number, record size, and number of records contained in the file are returned.

Parameters

transactionLog-Status	12 bytes returned • bytes 0-3 = Version Number • bytes 4-7 = Number of records • bytes 8-11 = Size of record
transactionLog-StatusLen	The length of the transaction log status.

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.16.4.95 int emv_setApplicationData (IN BYTE * *name*, IN int *nameLen*, IN BYTE * *tlv*, IN int *tlvLen*)

Set Application Data by AID

Sets the Application Data as specified by the application name and TLV data

Parameters

name	Application name, 10-32 ASCII hex characters representing 5-16 bytes Example "a0000000031010"
nameLen	the length of name data buffer of Application name,
tlv	Application data in TLV format
tlvLen	the length of tlv data buffer

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.16.4.96 int emv_setApplicationDataTLV (IN BYTE * *tlv*, IN int *tlvLen*)

Set Application Data by TLV

Sets the Application Data as specified by the 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) Example valid TLV, for Group #2, AID a0000000035010-: "dfee2d01029f0607a0000000051010ffe10101ffe50110ffe30114ffe20106"
tlvLen	the length of tlv data buffer

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.16.4.97 void emv_setAutoAuthenticateTransaction (IN int *authenticate*)

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

Parameters

authenticate	TRUE = auto authenticate, FALSE = manually authenticate

12.16.4.98 void emv_setAutoCompleteTransaction (IN int *complete*)

Enables complete for EMV transactions. If a emv_authenticateTranaction results in code 0x0004 (go online), then emv_completeTransaction can automatically execute if parameter is set to TRUE

Parameters

complete	TRUE = auto complete, FALSE = manually complete

12.16.4.99 int emv_setCAPK (IN BYTE * *capk,* IN int *capkLen*)

Set Certificate Authority Public Key

Sets the CAPK as specified by the CAKey structure

Parameters

capk	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.

capkLen	the length of capk data buffer

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.16.4.100 int emv_setCRL (IN BYTE * *list*, IN int *lsLen*)

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]
lsLen	the length of list data buffer

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.16.4.101 int emv_setException (IN BYTE * *exception*, IN int *exceptionLen*)

Set EMV Exception

Adds an entry to the EMV Exception List

Parameters

exception	EMV Exception entry containing the PAN and Sequence Number where [Exception] is 12 bytes: [1 byte Len][10 bytes PAN][1 byte Sequence Number] PAN, in compressed numeric, padded with F if required (example 0x5413339000001596FFFF)
exceptionLen	The length of the exception.

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.16.4.102 int emv_setTerminalData (IN BYTE * *tlv*, IN int *tlvLen*)

Set Terminal Data

Sets the Terminal Data for CTLS as specified by the TLV. The first TLV must be Configuration Group Number (Tag FFE4). The terminal global data is group 0, so the first TLV would be FFE40100. Other groups can be defined using this method (1 or greater), and those can be retrieved with emv_getConfigurationGroup(int group), and deleted with emv_removeConfigurationGroup(int group). You cannot delete group 0.

Parameters

tlv	TerminalData configuration file
tlvLen	the length of tlv data buffer

Return values

RETURN_CODE	Values can be parsed with device_getIDGStatusCodeString()

12.16.4.103 int emv_startTransaction (IN double *amount*, IN double *amtOther*, IN int *exponent*, IN int *type*, IN int *timeout*, IN BYTE * *tags*, IN int *tagsLen*, IN int *forceOnline*)

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) - SEE IMPORTANT NOTE BELOW
amtOther	Other amount value, if any (tag value 9F03) - SEE IMPORTANT NOTE BELOW
exponent	Number of characters after decimal point
type	Transaction type (tag value 9C).
timeout	Timeout value in seconds.
tags	Any other tags to be included in the request. Passed as a TLV stream. Example, tag 9F0C with amount 0x000000000100 would be 0x9F0C06000000000100 If tags 9F02 (amount),9-F03 (other amount), or 9C (transaction type) are included, they will take priority over these values supplied as individual parameters to this method.
tagsLen	The length of tags
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 = DFEE1A079F029-F36959F37

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

>>>>>IMPORTANT: parameters for amount and amtOther MUST BE PASSED AS A DOUBLE VALUE WITH DECIMAL POINT. Example, do not pass 1, but instead pass 1.0 or 1.00. Otherwise, results will be unpredictable

12.16.4.104 int lcd_addItemToList (IN BYTE * *listGraphicsID*, IN char * *itemName*, IN char * *itemID*, IN int *selected*)

Adds an item to an existing list.

Custom Display Mode must be enabled for custom text.

Parameters

listGraphicsID	Existing list's graphics ID (4 byte array) that is provided during creation
itemName	Item name (Maximum: 127 characters)
itemID	Identifier for the item (Maximum: 31 characters)
selected	If the item should be selected

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.16.4.105 int lcd_cancelSlideShow (OUT BYTE * *statusCode*, IN_OUT int * *statusCodeLen*)

Cancel slide show Cancel the slide show currently running

Parameters

statusCode	If the return code is not Success (0), the kernel may return a four-byte Extended Status Code
statusCodeLen	the length of the Extended Status Code (should be 4 bytes)

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.16.4.106 int lcd_captureSignature (IN int *timeout*)

Enables Signature Capture This command executes the signature capture screen. Once a signature is captured, it is sent to the callback with DeviceState.Signature, and the data will contain a .png of the signature

Parameters

timeout	Timeout waiting for the signature capture

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.16.4.107 int lcd_clearDisplay (IN BYTE *control*)

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

Parameters

control	for L100 only. 0:First Line 1:Second Line 2:Third Line 3:Fourth Line 0xFF: All Screen

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.16.4.108 int lcd_clearEventQueue ( )

Removes all entries from the event queue.

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.16.4.109 int lcd_createInputField (IN BYTE * *specs,* IN int *specsLen,* OUT BYTE * *graphicId*)

DEPRECATED : please use lcd_createInputField_Len(IN BYTE *specs, IN int specsLen, OUT BYTE *graphicId, IN_OUT int *graphicIdLen)

Create an input field on the screen.

Parameters

specs	The specs of the input field:

Length (bytes)	Description
2 - 4	X coordinate in pixels, zero terminated ASCII
----------——	----------——
2 - 4	Y coordinate in pixels, zero terminated ASCII
----------——	----------——
2 - 4	Width in pixels, zero terminated ASCII. Set to 0 (30h) for calculated width.
----------——	----------——
2 - 4	Height in pixels, zero terminated ASCII. Set to 0 (30h) for calculated height.
----------——	----------——
2	Font designation. Default font = 1, zero terminated ASCII
----------——	----------——
2 - 3	Zero terminated ASCII Font ID
----------——	----------——
3	Zero terminated ASCII hexadecimal display option flag
	Bit 0 0 = No Border
	1 = Show Border
	Bit 1 0 = Characters are first displayed on the leftmost area of the screen.
	1 = The first character entered is displayed on the rightmost area of
	the screen, and, as further digits are entered, characters scroll
	from the right to the left.
	Bit 2 - 15 Reserved
----------——	----------——
1 or 9	Foreground color, zero terminated ASCII hexadecimal
----------——	----------——
1 or 9	Background color, zero terminated ASCII hexadecimal
----------——	----------——
1 or 9	Border color, zero terminated ASCII hexadecimal
----------——	----------——
1 - 65	Prefill String, zero terminated ASCII
----------——	----------——
1 - 65	Format String, zero terminated ASCII

Parameters

specsLen	The length of specs

graphicsID	The graphicID of the event (required to be 4 bytes)

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.16.4.110 int lcd_createInputField_Len (IN BYTE * *specs*, IN int *specsLen*, OUT BYTE * *graphicId*, IN_OUT int * *graphicIdLen*)

Create an input field on the screen.

Parameters

specs	The specs of the input field:

Length (bytes)	Description
2 - 4	X coordinate in pixels, zero terminated ASCII
----------—	----------—
2 - 4	Y coordinate in pixels, zero terminated ASCII
----------—	----------—
2 - 4	Width in pixels, zero terminated ASCII. Set to 0 (30h) for calculated width.
----------—	----------—
2 - 4	Height in pixels, zero terminated ASCII. Set to 0 (30h) for calculated height.
----------—	----------—
2	Font designation. Default font = 1, zero terminated ASCII
----------—	----------—
2 - 3	Zero terminated ASCII Font ID
----------—	----------—
3	Zero terminated ASCII hexadecimal display option flag
	Bit 0 0 = No Border
	1 = Show Border
	Bit 1 0 = Characters are first displayed on the leftmost area of the screen.
	1 = The first character entered is displayed on the rightmost area of
	the screen, and, as further digits are entered, characters scroll
	from the right to the left.
	Bit 2 - 15 Reserved
----------—	----------—
1 or 9	Foreground color, zero terminated ASCII hexadecimal
----------—	----------—

1 or 9	Background color, zero terminated ASCII hexadecimal
----------—	----------—
1 or 9	Border color, zero terminated ASCII hexadecimal
----------—	----------—
1 - 65	Prefill String, zero terminated ASCII
----------—	----------—
1 - 65	Format String, zero terminated ASCII

Parameters

specsLen	The length of specs
graphicsID	The graphicID of the event (required to be 4 bytes)
graphicsIDLen	Length of graphicsID

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.16.4.111 int lcd_createList (IN int *posX,* IN int *posY,* IN int *numOfColumns,* IN int *numOfRows,* IN int *fontDesignation,* IN int *fontID,* IN int *verticalScrollArrowsVisible,* IN int *borderedListItems,* IN int *borderdScrollArrows,* IN int *touchSensitive,* IN int *automaticScrolling,* OUT BYTE * *graphicsID*)

DEPRECATED : please use lcd_createList_Len(IN int posX, IN int posY, IN int numOfColumns, IN int numOfRows, IN int fontDesignation, IN int fontID, IN int verticalScrollArrowsVisible, IN int borderedListItems, IN int borderdScrollArrows, IN int touchSensitive, IN int automaticScrolling, OUT BYTE *graphicsID, IN_OUT int *graphicsIDLen)

Creates a display list.

Parameters

posX	X coordinate in pixels
posY	Y coordinate in pixels
numOfColumns	Number of columns to display
numOfRows	Number of rows to display
fontDesignation	Font Designation 1 - Default font
fontID	Font styling | Font ID | Height in pixels | Font Properties | | --- | --- | --- | | 1 | 13 | Regular | | 2 | 17 | Regular | | 3 | 17 | Bold | | 4 | 22 | Regular | | 5 | 20 | Regular | | 6 | 20 | Bold | | 7 | 29 | Regular | | 8 | 38 | Regular | | 9 | 38 | Bold | | 10 | 58 | Regular | | 11 | 58 | Bold, mono-space | | 12 | 14 | Regular, mono-space, 8 pixels wide | | 13 | 15 | Regular, mono-space, 9 pixels wide | | 14 | 17 | Regular, mono-space, 9 pixels wide | | 15 | 20 | Regular, mono-space, 11 pixels wide | | 16 | 21 | Regular, mono-space, 12 pixels wide | | 17 | 25 | Regular, mono-space, 14 pixels wide | | 18 | 30 | Regular, mono-space, 17 pixels wide |
verticalScroll-ArrowsVisible	Display vertical scroll arrows by default
borederedList-Items	Draw border around list items
borederedScroll-Arrows	Draw border around scroll arrows (if visible)
touchSensitive	List items are touch enabled
automatic-Scrolling	Enable automatic scrolling of list when new items exceed display area
graphicsID	A four byte array containing the ID of the created element (optional) if graphicsID is NULL, the SDK will not return graphicsID if graphicsID is not NULL, the SDK will return graphicsID, but it will need 4 bytes of memory

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.16.4.112 int lcd_createList_Len (IN int *posX*, IN int *posY*, IN int *numOfColumns*, IN int *numOfRows*, IN int *fontDesignation*, IN int *fontID*, IN int *verticalScrollArrowsVisible*, IN int *borderedListItems*, IN int *borderdScrollArrows*, IN int *touchSensitive*, IN int *automaticScrolling*, OUT BYTE * *graphicsID*, IN_OUT int * *graphicsIDLen*)

Creates a display list.

Parameters

posX	X coordinate in pixels
posY	Y coordinate in pixels

numOfColumns	Number of columns to display
numOfRows	Number of rows to display
fontDesignation	Font Designation 1 - Default font
fontID	Font styling | Font ID | Height in pixels | Font Properties | | --- | --- | --- | | 1 | 13 | Regular | | 2 | 17 | Regular | | 3 | 17 | Bold | | 4 | 22 | Regular | | 5 | 20 | Regular | | 6 | 20 | Bold | | 7 | 29 | Regular | | 8 | 38 | Regular | | 9 | 38 | Bold | | 10 | 58 | Regular | | 11 | 58 | Bold, mono-space | | 12 | 14 | Regular, mono-space, 8 pixels wide | | 13 | 15 | Regular, mono-space, 9 pixels wide | | 14 | 17 | Regular, mono-space, 9 pixels wide | | 15 | 20 | Regular, mono-space, 11 pixels wide | | 16 | 21 | Regular, mono-space, 12 pixels wide | | 17 | 25 | Regular, mono-space, 14 pixels wide | | 18 | 30 | Regular, mono-space, 17 pixels wide |
verticalScroll-ArrowsVisible	Display vertical scroll arrows by default
borederedList-Items	Draw border around list items
borederedScroll-Arrows	Draw border around scroll arrows (if visible)
touchSensitive	List items are touch enabled
automatic-Scrolling	Enable automatic scrolling of list when new items exceed display area
graphicsID	A four byte array containing the ID of the created element (optional)
graphicsIDLen	Length of graphicsID (optional)

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.16.4.113 int lcd_customDisplayMode (IN int *enable*)

Custom Display Mode Controls the LCD display mode to custom display. Keyboard entry is limited to the Cancel, Clear, Enter and the function keys, if present. PIN entry is not permitted while the reader is in Custom Display Mode

Parameters

enable	TRUE = enabled, FALSE = disabled

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.16.4.114 int lcd_displayButton (IN int *posX,* IN int *posY,* IN int *buttonWidth,* IN int *buttonHeight,* IN int *fontDesignation,* IN int *fontID,* IN int *displayPosition,* IN char * *buttonLabel,* IN int *buttonTextColorR,* IN int *buttonTextColorG,* IN int *buttonTextColorB,* IN int *buttonBackgroundColorR,* IN int *buttonBackgroundColorG,* IN int *buttonBackgroundColorB,* OUT BYTE * *graphicsID*)

DEPRECATED : please use lcd_displayButton_Len(IN int posX, IN int posY, IN int buttonWidth, IN int buttonHeight, IN int fontDesignation, IN int fontID, IN int displayPosition, IN char *buttonLabel, IN int buttonTextColorR, IN int

buttonTextColorG, IN int buttonTextColorB, IN int buttonBackgroundColorR, IN int buttonBackgroundColorG, IN int buttonBackgroundColorB, OUT BYTE ∗graphicsID, IN_OUT int ∗graphicsIDLen)

Displays an interactive button.

Parameters

posX	X coordinate in pixels
posY	Y coordinate in pixels
buttonWidth	Width of the button
buttonHeight	Height of the button
fontDesignation	Font designation 1 - Default
Font	ID Font styling | Font ID | Height in pixels | Font Properties | | ------- | ---------------- | ---------- | | 1 | 13 | Regular | | 2 | 17 | Regular | | 3 | 17 | Bold | | 4 | 22 | Regular | | 5 | 20 | Regular | | 6 | 20 | Bold | | 7 | 29 | Regular | | 8 | 38 | Regular | | 9 | 38 | Bold | | 10 | 58 | Regular | | 11 | 58 | Bold, mono-space | | 12 | 14 | Regular, mono-space, 8 pixels wide | | 13 | 15 | Regular, mono-space, 9 pixels wide | | 14 | 17 | Regular, mono-space, 9 pixels wide | | 15 | 20 | Regular, mono-space, 11 pixels wide | | 16 | 21 | Regular, mono-space, 12 pixels wide | | 17 | 25 | Regular, mono-space, 14 pixels wide | | 18 | 30 | Regular, mono-space, 17 pixels wide |
displayPosition	Button display position 0 - Center on line Y without clearing screen and without word wrap 1 - Center on line Y after clearing screen and without word wrap 2 - Display at (X, Y) without clearing screen and without word wrap 3 - Display at (X, Y) after clearing screen and without word wrap 4 - Center button on screen without clearing screen and without word wrap 5 - Center button on screen after clearing screen and without word wrap 64 - Center on line Y without clearing screen and with word wrap 65 - Center on line Y after clearing the screen and with word wrap 66 - Display at (X, Y) without clearing screen and with word wrap 67 - Display at (X, Y) after clearing screen and with word wrap 68 - Center button on screen without clearing screen and with word wrap 69 - Center button on screen after clearing screen and with word wrap
buttonLabel	Button label text (Maximum: 31 characters)
buttonTextColor-R	- Red component for foreground color (0 - 255)
buttonTextColor-G	- Green component for foreground color (0 - 255)
buttonTextColor-B	- Blue component for foreground color (0 - 255)
button-Background-ColorR	- Red component for background color (0 - 255)

buttonBackgroundColorG	- Green component for background color (0 - 255)
buttonBackgroundColorB	- Blue component for background color (0 - 255)
graphicsID	A four byte array containing the ID of the created element (optional) if graphicsID is NULL, the SDK will not return graphicsID if graphicsID is not NULL, the SDK will return graphicsID, but it will need 4 bytes of memory

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.16.4.115 int lcd_displayButton_Len (IN int *posX*, IN int *posY*, IN int *buttonWidth*, IN int *buttonHeight*, IN int *fontDesignation*, IN int *fontID*, IN int *displayPosition*, IN char * *buttonLabel*, IN int *buttonTextColorR*, IN int *buttonTextColorG*, IN int *buttonTextColorB*, IN int *buttonBackgroundColorR*, IN int *buttonBackgroundColorG*, IN int *buttonBackgroundColorB*, OUT BYTE * *graphicsID*, IN_OUT int * *graphicsIDLen*)

Displays an interactive button.

Parameters

posX	X coordinate in pixels
posY	Y coordinate in pixels
buttonWidth	Width of the button
buttonHeight	Height of the button
fontDesignation	Font designation 1 - Default
Font	ID Font styling

Font ID	Height in pixels	Font Properties
1	13	Regular
2	17	Regular
3	17	Bold
4	22	Regular
5	20	Regular
6	20	Bold
7	29	Regular
8	38	Regular
9	38	Bold
10	58	Regular
11	58	Bold, mono-space
12	14	Regular, mono-space, 8 pixels wide
13	15	Regular, mono-space, 9 pixels wide
14	17	Regular, mono-space, 9 pixels wide
15	20	Regular, mono-space, 11 pixels wide
16	21	Regular, mono-space, 12 pixels wide
17	25	Regular, mono-space, 14 pixels wide
18	30	Regular, mono-space, 17 pixels wide

displayPosition	Button display position 0 - Center on line Y without clearing screen and without word wrap 1 - Center on line Y after clearing screen and without word wrap 2 - Display at (X, Y) without clearing screen and without word wrap 3 - Display at (X, Y) after clearing screen and without word wrap 4 - Center button on screen without clearing screen and without word wrap 5 - Center button on screen after clearing screen and without word wrap 64 - Center on line Y without clearing screen and with word wrap 65 - Center on line Y after clearing the screen and with word wrap 66 - Display at (X, Y) without clearing screen and with word wrap 67 - Display at (X, Y) after clearing screen and with word wrap 68 - Center button on screen without clearing screen and with word wrap 69 - Center button on screen after clearing screen and with word wrap
buttonLabel	Button label text (Maximum: 31 characters)
buttonTextColor-R	- Red component for foreground color (0 - 255)
buttonTextColor-G	- Green component for foreground color (0 - 255)
buttonTextColor-B	- Blue component for foreground color (0 - 255)
button-Background-ColorR	- Red component for background color (0 - 255)
button-Background-ColorG	- Green component for background color (0 - 255)
button-Background-ColorB	- Blue component for background color (0 - 255)
graphicsID	A four byte array containing the ID of the created element (optional)
graphicsIDLen	Length of graphicsID (optional)

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.16.4.116 int lcd_displayParagraph (IN int *posX*, IN int *posY*, IN int *displayWidth*, IN int *displayHeight*, IN int *fontDesignation*, IN int *fontID*, IN int *displayProperties*, IN char * *displayText*)

Displays text with scroll feature.

Custom Display Mode must be enabled.

Parameters

posX	X coordinate in pixels
posY	Y coordinate in pixels
displayWidth	Width of the display area in pixels (Minimum: 40px) 0 or NULL - Use the full width to display text
displayHeight	Height of the display area in pixels (Minimum: 100px) 0 or NULL - Use the full height to display text
fontDesignation	Font designation 1 - Default
fontID	Font styling **Font ID** | **Height in pixels** | **Font Properties**

| 1 | 13 | Regular | | 2 | 17 | Regular | | 3 | 17 | Bold | | 4 | 22 | Regular | | 5 | 20 | Regular | | 6 | 20 | Bold | | 7 | 29 | Regular | | 8 | 38 | Regular | | 9 | 38 | Bold | | 10 | 58 | Regular | | 11 | 58 | Bold, mono-space | | 12 | 14 | Regular, mono-space, 8 pixels wide | | 13 | 15 | Regular, mono-space, 9 pixels wide | | 14 | 17 | Regular, mono-space, 9 pixels wide | | 15 | 20 | Regular, mono-space, 11 pixels wide | | 16 | 21 | Regular, mono-space, 12 pixels wide | | 17 | 25 | Regular, mono-space, 14 pixels wide | | 18 | 30 | Regular, mono-space, 17 pixels wide |

Parameters

display-Properties	Display properties for the text 0 - Center on line Y without clearing screen 1 - Center on line Y after clearing screen 2 - Display at (X, Y) without clearing screen 3 - Display at (X, Y) after clearing screen 4 - Center on screen without clearing screen 5 - Center on screen after clearing screen
displayText	Display text (Maximum: 3999 characters plus terminator)

12.16.4.117 int lcd_displayText (IN int *posX,* IN int *posY,* IN int *displayWidth,* IN int *displayHeight,* IN int *fontDesignation,* IN int *fontID,* IN int *screenPosition,* IN char * *displayText,* OUT BYTE * *graphicsID*)

DEPRECATED : please use lcd_displayText_Len(IN int posX, IN int posY, IN int displayWidth, IN int displayHeight, IN int fontDesignation, IN int fontID, IN int screenPosition, IN char *displayText, OUT BYTE *graphicsID, IN_OUT int *graphicsIDLen)

Displays text.

Custom Display Mode must be enabled for custom text. PIN pad entry is not allowed in Custom Display Mode but the Cancel, OK, and Clear keys remain active.

Parameters

posX	X coordinate in pixels
posY	Y coordinate in pixels
displayWidth	Width of the display area in pixels (optional)
displayHeight	Height of the display area in pixels (optional)
fontDesignation	Font designation 1 - Default

Font	ID Font styling
	`| Font ID | Height in pixels | Font Properties |`
	`| ------- | ---------------- | ---------- |`
	`| 1 | 13 | Regular |`
	`| 2 | 17 | Regular |`
	`| 3 | 17 | Bold |`
	`| 4 | 22 | Regular |`
	`| 5 | 20 | Regular |`
	`| 6 | 20 | Bold |`
	`| 7 | 29 | Regular |`
	`| 8 | 38 | Regular |`
	`| 9 | 38 | Bold |`
	`| 10 | 58 | Regular |`
	`| 11 | 58 | Bold, mono-space |`
	`| 12 | 14 | Regular, mono-space, 8 pixels wide |`
	`| 13 | 15 | Regular, mono-space, 9 pixels wide |`
	`| 14 | 17 | Regular, mono-space, 9 pixels wide |`
	`| 15 | 20 | Regular, mono-space, 11 pixels wide |`
	`| 16 | 21 | Regular, mono-space, 12 pixels wide |`
	`| 17 | 25 | Regular, mono-space, 14 pixels wide |`
	`| 18 | 30 | Regular, mono-space, 17 pixels wide |`

screenPosition	Display position 0 - Center on line Y without clearing screen 1 - Center on line Y after clearing screen 2 - Display at (X, Y) without clearing screen 3 - Display at (X, Y) after clearing screen 4 - Display at center of screen without clearing screen 5 - Display at center of screen after clearing screen 6 - Display text right-justified without clearing screen 7 - Display text right-justified after clearing screen
displayText	Display text (Maximum: 1900 characters)
graphicsID	A four byte array containing the ID of the created element (optional) if graphicsID is NULL, the SDK will not return graphicsID if graphicsID is not NULL, the SDK will return graphicsID, but it will need 4 bytes of memory

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.16.4.118 int lcd_displayText_Len (IN int *posX*, IN int *posY*, IN int *displayWidth*, IN int *displayHeight*, IN int *fontDesignation*, IN int *fontID*, IN int *screenPosition*, IN char * *displayText*, OUT BYTE * *graphicsID*, IN_OUT int * *graphicsIDLen*)

Displays text.

Custom Display Mode must be enabled for custom text. PIN pad entry is not allowed in Custom Display Mode but the Cancel, OK, and Clear keys remain active.

Parameters

posX	X coordinate in pixels
posY	Y coordinate in pixels
displayWidth	Width of the display area in pixels (optional)
displayHeight	Height of the display area in pixels (optional)
fontDesignation	Font designation 1 - Default
Font	ID Font styling

Font ID	Height in pixels	Font Properties
1	13	Regular
2	17	Regular
3	17	Bold
4	22	Regular
5	20	Regular
6	20	Bold
7	29	Regular
8	38	Regular
9	38	Bold
10	58	Regular
11	58	Bold, mono-space
12	14	Regular, mono-space, 8 pixels wide
13	15	Regular, mono-space, 9 pixels wide
14	17	Regular, mono-space, 9 pixels wide
15	20	Regular, mono-space, 11 pixels wide
16	21	Regular, mono-space, 12 pixels wide
17	25	Regular, mono-space, 14 pixels wide
18	30	Regular, mono-space, 17 pixels wide

screenPosition	Display position 0 - Center on line Y without clearing screen 1 - Center on line Y after clearing screen 2 - Display at (X, Y) without clearing screen 3 - Display at (X, Y) after clearing screen 4 - Display at center of screen without clearing screen 5 - Display at center of screen after clearing screen 6 - Display text right-justified without clearing screen 7 - Display text right-justified after clearing screen
displayText	Display text (Maximum: 1900 characters)
graphicsID	A four byte array containing the ID of the created element (optional)
graphicsIDLen	Length of graphicsID (optional)

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.16.4.119 int lcd_getInputEvent (IN int *timeout*, OUT int * *dataReceived*, OUT BYTE * *eventType*, OUT BYTE * *graphicsID*, OUT BYTE * *eventData*)

DEPRECATED : please use lcd_getInputEvent_Len(IN int timeout, OUT int *dataReceived, OUT BYTE *eventType, IN_OUT int *eventTypeLen, OUT BYTE *graphicsID, IN_OUT int *graphicsIDLen, OUT BYTE *eventData, IN_OUT int *eventDataLen)

Requests input from the reader.

Parameters

timeout	Timeout amount in seconds 0 - No timeout
dataReceived	Indicates if an event occurred and data was received 0 - No data received 1 - Data received
eventType	The event type (required to be at least 4 bytes), see table below
graphicsID	The graphicID of the event (required to be at least 4 bytes)
eventData	The event data, see table below (required to be at least 73 bytes)

Event Type	Value (4 bytes)	Event Specific Data
Button Event	00030000h	Length = Variable
		Byte 1: State (1 = Pressed, other values RFU)
		Byte 2 - n: Null terminated caption
----------—	----------—	----------——
Checkbox Event	00030001h	Length = 1 byte
		Byte 1: State (1 = Checked, 0 = Unchecked)
----------—	----------—	----------——
Line Item Event	00030002h	Length = 5 bytes
		Byte 1: State (1 = Item Selected, other values RFU)
		Byte 2 - n: Caption of the selected item
----------—	----------—	----------——
Keypad Event	00030003h	Length - 3 bytes
		Byte 1: State (1 = key pressed, 2 = key released, other values RFU)
		Byte 2 - 3: Key pressed and Key release

		0030h - KEYPAD_KEY_0
		0031h - KEYPAD_KEY_1
		0032h - KEYPAD_KEY_2
		0033h - KEYPAD_KEY_3
		0034h - KEYPAD_KEY_4
		0035h - KEYPAD_KEY_5
		0036h - KEYPAD_KEY_6
		0037h - KEYPAD_KEY_7
		0038h - KEYPAD_KEY_8
		0039h - KEYPAD_KEY_9
		Byte 2 - 3: Only Key pressed
		000Dh - KEYPAD_KEY_ENTER
		0008h - KEYPAD_KEY_CLEAR
		001Bh - KEYPAD_KEY_CANCEL
		0070h - FUNC_KEY_F1 (Vend III)
		0071h - FUNC_KEY_F2 (Vend III)
		0072h - FUNC_KEY_F3 (Vend III)
		0073h - FUNC_KEY_F4 (Vend III)
----------	----------	----------
Touchscreen Event	00030004h	Length = 1 - 33 bytes
		Byte 1: State (not used)
		Byte 2 - 33: Image name (zero terminated)
----------	----------	----------
Slideshow Event	00030005h	Length = 1 byte
		Byte 1: State (not used)
----------	----------	----------
Transaction Event	00030006h	Length = 9 bytes
		Byte 1: State (not used)
		Byte 2 - 5: Card type (0 = unknown)
		Byte 6 - 9: Status - A four byte, big-endian field
		Byte 9 is used to store the 1-byte status code
		00 - SUCCESS
		08 - TIMEOUT
		0A - FAILED
		This is not related to the extended status codes
----------	----------	----------
Radio Button Event	00030007h	Length = 73 bytes
		Byte 1: State (1 = Change ins selected button, other values RFU)
		Byte 2 - 33: Null terminated group name

		Byte 34 - 65: Radio button caption

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.16.4.120 int lcd_getInputEvent_Len (IN int *timeout*, OUT int * *dataReceived*, OUT BYTE * *eventType*, IN_OUT int * *eventTypeLen*, OUT BYTE * *graphicsID*, IN_OUT int * *graphicsIDLen*, OUT BYTE * *eventData*, IN_OUT int * *eventDataLen*)

Requests input from the reader.

Parameters

timeout	Timeout amount in seconds 0 - No timeout
dataReceived	Indicates if an event occurred and data was received 0 - No data received 1 - Data received
eventType	The event type (required to be at least 4 bytes), see table below
eventTypeLen	Length of eventType
graphicsID	The graphicID of the event (required to be at least 4 bytes)
graphicsIDLen	length of graphicID
eventData	The event data, see table below (required to be at least 73 bytes)

Event Type	Value (4 bytes)	Event Specific Data
Button Event	00030000h	Length = Variable
		Byte 1: State (1 = Pressed, other values RFU)
		Byte 2 - n: Null terminated caption
----------	----------	----------
Checkbox Event	00030001h	Length = 1 byte
		Byte 1: State (1 = Checked, 0 = Unchecked)
----------	----------	----------
Line Item Event	00030002h	Length = 5 bytes
		Byte 1: State (1 = Item Selected, other values RFU)
		Byte 2 - n: Caption of the selected item
----------	----------	----------
Keypad Event	00030003h	Length - 3 bytes
		Byte 1: State (1 = key pressed, 2 = key released, other values RFU)
		Byte 2 - 3: Key pressed and Key release
		0030h - KEYPAD_KEY_0
		0031h - KEYPAD_KEY_1
		0032h - KEYPAD_KEY_2
		0033h - KEYPAD_KEY_3
		0034h - KEYPAD_KEY_4
		0035h - KEYPAD_KEY_5
		0036h - KEYPAD_KEY_6

		0037h - KEYPAD_KEY_7
		0038h - KEYPAD_KEY_8
		0039h - KEYPAD_KEY_9
		Byte 2 - 3: Only Key pressed
		000Dh - KEYPAD_KEY_ENTER
		0008h - KEYPAD_KEY_CLEAR
		001Bh - KEYPAD_KEY_CANCEL
		0070h - FUNC_KEY_F1 (Vend III)
		0071h - FUNC_KEY_F2 (Vend III)
		0072h - FUNC_KEY_F3 (Vend III)
		0073h - FUNC_KEY_F4 (Vend III)
----------—	----------—	----------——
Touchscreen Event	00030004h	Length = 1 - 33 bytes
		Byte 1: State (not used)
		Byte 2 - 33: Image name (zero terminated)
----------—	----------—	----------——
Slideshow Event	00030005h	Length = 1 byte
		Byte 1: State (not used)
----------—	----------—	----------——
Transaction Event	00030006h	Length = 9 bytes
		Byte 1: State (not used)
		Byte 2 - 5: Card type (0 = unknown)
		Byte 6 - 9: Status - A four byte, big-endian field
		Byte 9 is used to store the 1-byte status code
		00 - SUCCESS
		08 - TIMEOUT
		0A - FAILED
		This is not related to the extended status codes
----------—	----------—	----------——
Radio Button Event	00030007h	Length = 73 bytes
		Byte 1: State (1 = Change ins selected button, other values RFU)
		Byte 2 - 33: Null terminated group name
		Byte 34 - 65: Radio button caption

Parameters

eventDataLen	Length of eventData

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.16.4.121 int lcd_getInputFieldValue (IN BYTE * *graphicId,* OUT BYTE * *retData,* IN_OUT int * *retDataLen*)

Get the keypad data that was entered into the specified Input Field.

Parameters

graphicsID	The graphicID of the input field (required to be 4 bytes)
retData	return keypad data
retDataLen	The length of retData

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.16.4.122 int lcd_getSelectedListItem (IN BYTE * *listGraphicsID,* OUT char * *itemID*)

DEPRECATED : please use lcd_getSelectedListItem_Len(IN BYTE *listGraphicsID, OUT char *itemID, IN_OUT int *itemIDLen)

Retrieves the selected item's ID.

Parameters

listGraphicsID	Existing list's graphics ID (4 byte array) that is provided during creation
itemID	The selected item's ID (Maximum: 32 characters) Need 33 bytes of memory including '\0'

12.16.4.123 int lcd_getSelectedListItem_Len (IN BYTE * *listGraphicsID,* OUT char * *itemID,* IN_OUT int * *itemIDLen*)

Retrieves the selected item's ID.

Parameters

listGraphicsID	Existing list's graphics ID (4 byte array) that is provided during creation
itemID	The selected item's ID (Maximum: 32 characters) Need 33 bytes of memory including '\0'
itemIDLen	Length of itemID

12.16.4.124 int lcd_resetInitialState ( )

Reset to Initial State This command places the reader UI into the idle state and displays the appropriate idle display.

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.16.4.125 int lcd_setBackgroundImage (IN char * *file,* IN int *fileLen,* IN int *enable*)

Set Background Image You must send images to the reader??s memory and send a Start Custom Mode command to the reader before it will respond to Image commands. Image files must be in .bmp or .png format.

Parameters

file	Complete path and file name of the file you want to use. Example "file.png" will put in root directory, while "ss/file.png" will put in ss directory (which must exist)
fileLen	Length of files
enable	TRUE = Use Background Image, FALSE = Use Background Color

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.16.4.126 int lcd_setDisplayImage (IN char * *file,* IN int *fileLen,* IN int *posX,* IN int *posY,* IN int *posMode,* IN int *touchEnable,* IN int *clearScreen*)

Set Display Image You must send images to the reader??s memory and send a Start Custom Mode command to the reader before it will respond to Image commands. Image files must be in .bmp or .png format.

Parameters

file	Complete path and file name of the file you want to use. Example "file.png" will put in root directory, while "ss/file.png" will put in ss directory (which must exist)
fileLen	Length of files
posX	X coordinate in pixels, Range 0-271
posY	Y coordinate in pixels, Range 0-479
posMode	Position Mode • 0 = Center on Line Y • 1 = Display at (X,Y) • 2 - Center on screen
touchEnable	TRUE = Image is touch sensitive
clearScreen	TRUE = Clear screen

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.16.4.127 int lcd_setForeBackColor (IN BYTE * *foreRGB*, IN int *foreRGBLen*, IN BYTE * *backRGB*, IN int *backRGBLen*)

Set Foreground and Background Color This command sets the foreground and background colors of the LCD.

Parameters

foreRGB	Foreground RGB. 000000 = black, FF0000 = red, 00FF00 = green, 0000FF = blue, FFFFFF = white
Length	of foreRGB. Must be 3.
backRGB	Background RGB. 000000 = black, FF0000 = red, 00FF00 = green, 0000FF = blue, FFFFFF = white
Length	of backRGB. Must be 3.

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.16.4.128 int lcd_startSlideShow (IN char * *files*, IN int *filesLen*, IN int *posX*, IN int *posY*, IN int *posMode*, IN int *touchEnable*, IN int *recursion*, IN int *touchTerminate*, IN int *delay*, IN int *loops*, IN int *clearScreen*)

Start slide show You must send images to the reader??s memory and send a Start Custom Mode command to the reader before it will respond to this commands. Image files must be in .bmp or .png format.

Parameters

files	Complete paths and file names of the files you want to use, separated by commas. If a directory is specified, all files in the dirctory are displayed
filesLen	Length of files
posX	X coordinate in pixels, Range 0-271
posY	Y coordinate in pixels, Range 0-479
posMode	Position Mode • 0 = Center on Line Y • 1 = Display at (X,Y) • 2 - Center on screen
touchEnable	TRUE = Image is touch sensitive
recursion	TRUE = Recursively follow directorys in list
touchTerminate	TRUE = Terminate slideshow on touch (if touch enabled)
delay	Number of seconds between image displays
loops	Number of display loops. A zero indicates continuous display
clearScreen	TRUE = Clear screen

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.16.4.129 int msr_cancelMSRSwipe ( )

Disable MSR Swipe Cancels MSR swipe request.

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.16.4.130 int msr_flushTrackData ( )

Flush Track Data Clears any track data being retained in memory by future PIN Block request.

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.16.4.131 void msr_registerCallBk (pMSR_callBack *pMSRf*)

To register the msr callback function to get the MSR card data. (Pass NULL to disable the callback.)

12.16.4.132 void msr_registerCallBkp (pMSR_callBackp *pMSRf*)

To register the msr callback function to get the MSR card data pointer. (Pass NULL to disable the callback.)

12.16.4.133 int msr_startMSRSwipe (IN int *_timeout*)

Start MSR Swipe Enables MSR, waiting for swipe to occur. Allows track selection. Returns IDTMSRData instance to deviceDelegate::swipeMSRData:()

Parameters

timeout	Swipe Timeout Value

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString() Note: if auto poll mode is on, it will return command not allowed error

12.16.4.134 void parseMSRData (IN BYTE * *resData*, IN int *resLen*, IN_OUT IDTMSRData * *cardData*)

Parser the MSR data from the buffer into IDTMSTData structure

Parameters

resData	MSR card data buffer
resLen	the length of resData
cardData	the parser result with IDTMSTData structure

12.16.4.135 int pin_getEncryptedOnlinePIN (IN int *keyType*, IN int *timeout*)

Get Encrypted DUKPT PIN

Requests PIN Entry for online authorization. PIN block and KSN returned in callback function DeviceState.-TransactionData with cardData.pin_pinblock. A swipe must be captured first before this function can execute

Parameters

keyType	PIN block key type. Valid values 0,3 for TDES, 4 for AES
timeout	PIN entry timout

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.16.4.136 int pin_getPAN (IN int *getCSC*, IN int *timeout*)

Get PAN

Requests PAN Entry on pinpad

Parameters

getCSC	Include Customer Service Code (also known as CVV, CVC)
timeout	PAN entry timout

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.16.4.137 int pin_promptCreditDebit (IN char * *currencySymbol*, IN int *currencySymbolLen*, IN char * *displayAmount*, IN int *displayAmountLen*, IN int *timeout*, OUT BYTE * *retData*, IN_OUT int * *retDataLen*)

Prompt for Credit or Debit

Requests prompt for Credit or Debit. Response returned in callback function as DeviceState.MenuItem with data MENU_SELECTION_CREDIT = 0, MENU_SELECTION_DEBIT = 1

Parameters

currencySymbol	Allowed values are $ (0x24), ???(0xA5), ???(0xA3), ???(0xA4), or NULL
currencySymbol-Len	length of currencySymbol
displayAmount	Amount to display (can be NULL)
displayAmount-Len	length of displayAmount
timeout	Menu entry timout. Valid values 2-20 seconds

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString()

12.16.4.138 void pin_registerCallBk (pPIN_callBack *pPINf*)

To register the pin callback function to get the PINPad data. (Pass NULL to disable the callback.)

12.16.4.139 void registerHotplugCallBk (pMessageHotplug *pMsgHotplug*)

To register the USB HID hot-plug callback function which implemented in the application to monitor the hotplug message from the SDK.

12.16.4.140 void registerLogCallBk (pSendDataLog *pFSend,* pReadDataLog *pFRead*)

To register the log callback function which implemented in the application to monitor sending/reading data between application and reader.

12.16.4.141 char* SDK_Version ( )

To Get SDK version

Returns

return the SDK version string

12.16.4.142 int setAbsoluteLibraryPath (const char * *absoluteLibraryPath*)

Set the path to use when searching for ID TECH's libraries. If this is not set, the libraries will be searched for with the system's default procedures.

Parameters

absoluteLibrary-Path	The absolute path to ID TECH's libraries.

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.16.4.143 int ws_deleteSSLCert (IN char * *name,* IN int *nameLen*)

Delete SSL Certificate Deletes a SSL Certificate by name

Parameters

name	Name of certificate to delete
nameLen	Certificate Name Length

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.16.4.144 int ws_getCertChainType (OUT int * *type*)

Get Certificate Chain Type Returns indicator for using test/production certificate chain

Parameters

type	0 = test certificate chain, 1 = production certificate chain

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.16.4.145 int ws_loadSSLCert (IN char * *name*, IN int *nameLen*, IN char * *dataDER*, IN int *dataDERLen*)

Load SSL Certificate Loads a SSL certificate

Parameters

name	Certificate Name
nameLen	Certificate Name Length
dataDER	DER encoded certificate data
dataDERLen	DER encoded certificate data length

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.16.4.146 int ws_requestCSR (OUT RequestCSR * *csr*)

Request CSR Requests 3 sets of public keys: encrypting Keys, signing/validating keys, signing/validating 3rd party apps

Parameters

csr	RequestCSR structure to return the data

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.16.4.147 int ws_revokeSSLCert (IN char * *name*, IN int *nameLen*)

Revoke SSL Certificate Revokes a SSL Certificate by name

Parameters

name	Name of certificate to revoke
nameLen	Certificate Name Length

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

12.16.4.148 int ws_updateRootCertificate (IN char * *name*, IN int *nameLen*, IN char * *dataDER*, IN int *dataDERLen*, IN char * *signature*, IN int *signatureLen*)

Update Root Certificate Updates the root certificate

Parameters

name	Certificate Name
nameLen	Certificate Name Length
dataDER	DER encoded certificate data
dataDERLen	DER encoded certificate data length
signature	Future Root CA signed (RSASSA PSS SHA256) by current Root CA
signature	length

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

Index