IDTech Windows SDK Guide

for SREDKey2

#80172502-001

Rev. A

Revision History

Revision	Description and Reason for Change	Date
A	Initial Release - Manual;User;SREDKey2;SDK;Win	10/22/2019

Contents

Chapter 1

IDTech Windows SDK Reference Guide for SREDKey2

IDTechSDK.dll is a Windows dynamic link libraries that will be provided by IDTech as the main interface between Windows Form (WinForms) for the SREDKey2 and payment processing solutions.

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

Chapter 2

Important Security Notice

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

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

2.1 Applicability

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

2.2 What Does PA-DSS Mean to You?

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

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

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

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

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

2.3 Third Party Applications

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

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

2.4 PA-DSS Guidelines

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

1. Sensitive Data Storage Guidelines

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

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

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

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

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

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

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

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

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

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

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

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

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

2. Protect stored cardholder data

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

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

Notes:

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

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

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

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

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

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

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

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

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

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

3. Provide secure authentication features

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

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

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

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

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

4. Log payment application activity

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

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

5. Develop secure payment applications

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

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

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

5.1.2 Separate development/test, and production environments

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

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

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

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

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

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

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

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

- Back-out or product de-installation procedures

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

6. Protect wireless transmissions

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

7. Test payment applications to address vulnerabilities

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

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

8. Facilitate secure network implementation

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

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

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

10. Facilitate secure remote software updates

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

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

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

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

11. Encrypt sensitive traffic over public networks

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

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

12. Encrypt all non-console administrative access

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

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

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

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

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

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

2.5 More Information

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

For more information related to security, visit:

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

CAPN questions: capninfocenter@aexp.com

Chapter 3

Connecting to SREDKey2

The SREDKey2 connects through USB. The Audio Jack interface is not supported at this time

3.1 Connect with USB:

The SREDKey2 uses USB-HID (Human Interface Device) and does not need any vendor-specific drivers. Simply by plugging into an available USB port makes it available for SDK connectivity.

Chapter 4

Core Implementation: WinForms

IDTechSDK.dll includes class libraries to interface with the SREDKey2. This guide assume a fair understanding of Visual Studio 2013+, C# and general Windows programming knowledge.

4.1 Integrating with IDTechSDK.dll

- Import the necessary libraries
- Add using statements to utilize library
- Implement the callback function
- Initialize SREDKey2

4.2 Import the necessary libraries

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

- IDTechSDK.dll

The best way to obtain this library, and keep in sync with the latest updates, is to import the library from NuGet. The NuGet package is "IDTechSDK_STD". This will also install any other dependencies needed for the particular project you are building for.

Alternately, you can use the IDTechSDK.dll included with the SDK distribution. This is a legacy version that doesn't have any other dependencies, and will work with a minimum .Net Framework version 4.5. If you use this SDK, you would add the reference as you would any .NET managed library reference. The most direct method would be right-click on the "References" in the Solution Explorer for the project, select "Add Reference...", click "Browse..." and locate IDTechSDK.dll.

4.3 Add using statements to utilize library

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

```
using IDTechSDK;
```

4.4 Implement the callback function

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

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

        case DeviceState.DefaultDeviceTypeChange:
            //The SDK is changing the default device to IDT_DEVICE_TYPES type
            break;
        case DeviceState.Connected:
            //A connection has been made to IDT_DEVICE_TYPES type
            break;
        case DeviceState.Disconnected:
            //A disconnection has occured with IDT_DEVICE_TYPES type
            break;
        case DeviceState.DataReceived:
            //Low-level data received  for IDT_DEVICE_TYPES type
            break;
        case DeviceState.DataSent:
            //Low-level data sent  for IDT_DEVICE_TYPES type
            break;
        case DeviceState.TransactionData:
            //This will capture the cardData object from keypad entry or card swipe when device is in
       USB-HID Mode
            //If the device is in USB-KB Mode, the transaction data will be output through the keyboard
       buffer
            break;
        default:
            break;

    }


}
```

4.5 Initialize SREDKey2:

A Singleton instance has been established in the IDT_SREDKey2 class. Establish the callback, and then any call to the singleton will automatically connect via USB if you then enable USB Monitoring.

```
public SREDKey2_Simple_Demo()
{
    InitializeComponent();
    IDT_SREDKey2.setCallback(MessageCallBack);
     IDT_Device.startUSBMonitoring();
     string firmwareVersion = "";
    RETURN_CODE rt = IDT_SREDKey2.SharedController.device_getFirmwareVersion(ref firmwareVersion);

}
```

Alternately, the IDTechSDK.dll library can be used to drive multiple IDTech Devices. If the plan is to use two or more devices, then the IDT_Device class should be used instead of the invidual device class. You can then tell the SDK what device to use, and then use the same code/API method for both devices:

```
public MultiDevice_Simple_Demo()
{
    InitializeComponent();
    IDT_Device.setCallback(MessageCallBack);
     IDT_Device.startUSBMonitoring();
     string firmwareVersion = "";

     //Let's first get the FW version from SREDKEY2
     IDT_Device.setDeviceType(IDT_DEVICE_Types.IDT_DEVICE_SREDKEY2, DEVICE_INTERFACE_Types.
      DEVICE_INTERFACE_USB,
            DEVICE_PROTOCOL_Types.DEVICE_PROTOCOL_KB);

    RETURN_CODE rt = IDT_Device.SharedController.device_getFirmwareVersion(ref firmwareVersion);

        //Let's now switch to the VP3300 (also connected to computer via USB) and get firmware version from
       that device
```

```
        IDT_Device.setDeviceType(IDT_DEVICE_Types.IDT_DEVICE_VP3300, DEVICE_INTERFACE_Types.
      DEVICE_INTERFACE_USB,
            DEVICE_PROTOCOL_Types.DEVICE_PROTOCOL_IDG);

       rt = IDT_Device.SharedController.device_getFirmwareVersion(ref firmwareVersion);

}
```

The SDK is driven by the commands in the IDT_Device class. This class has all the functions for many different ID Tech devices. As a result, there may be many functions that may not be applicable to your particular device. That is why we also provide the individual device class, like IDT_SREDKey2. When you use that individual device class, that acts as a filter of all the available commands that can be executed in the IDT_Device class. If you attempt to use an API method in IDT_Device class not intended for your particular device, either the SDK or the device FW will return an applicable error code as a response.

Chapter 5

Error Code Reference

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

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

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

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

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

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

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

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

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

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

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

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

        }
        return "";
    }
```

Chapter 6

Namespace Index

6.1 Packages

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

Chapter 7

Class Index

7.1 Class List

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

Chapter 8

Namespace Documentation

8.1 IDTechSDK Namespace Reference

Classes

- class IDT_SREDKey2

Chapter 9

Class Documentation

9.1 IDTechSDK.IDT_SREDKey2 Class Reference

Public Member Functions

- RETURN_CODE device_getOutputType (ref int response)
- RETURN_CODE device_setOutputType (int value)
- RETURN_CODE config_getModelNumber (ref string response)
- RETURN_CODE device_sendVivoCommandP3 (byte command, byte subCommand, byte[] data, ref byte[] response)
- RETURN_CODE device_sendVivoCommandP3_ext (byte command, byte subCommand, byte[] data, ref byte[] response, int timeout, bool noResponse)
- RETURN_CODE device_sendVivoCommandP2 (byte command, byte subCommand, byte[] data, ref byte[] response)
- RETURN_CODE device_sendVivoCommandP2_ext (byte command, byte subCommand, byte[] data, ref byte[] response, int timeout, bool noResponse)
- RETURN_CODE config_getSerialNumber (ref string response)
- RETURN_CODE device_getFirmwareVersion (ref string response)
- string device_getResponseCodeString (RETURN_CODE eCode)
- RETURN_CODE device_sendDataCommand (string cmd, bool calcLRC, ref byte[] response)
- RETURN_CODE device_sendDataCommand_ext (string cmd, bool calcLRC, ref byte[] response, int timeout, bool noResponse)
- RETURN_CODE msr_disable ()
- RETURN_CODE msr_enable ()
- IDTechSDK.RETURN_CODE device_setLanguage (int val)
- RETURN_CODE msr_getSwipeEncryption (ref byte encryption)
- RETURN_CODE device_getTransArmorID (ref string TID)
- RETURN_CODE device_setTransArmorID (string TID)
- RETURN_CODE device_enableAdminKey (bool enable)
- RETURN_CODE msr_setExpirationMask (bool mask)
- RETURN_CODE msr_getExpirationMask (ref byte value)
- RETURN_CODE msr_setClearPANID (byte val)
- RETURN_CODE msr_getClearPANID (ref byte value)
- RETURN_CODE msr_getSwipeMaskOption (ref byte option)
- RETURN_CODE msr_setSwipeMaskOption (bool track1, bool track2, bool track3)
- RETURN_CODE msr_setSwipeForcedEncryptionOption (bool track1, bool track2, bool track3, bool track3card0)
- RETURN_CODE msr_getSwipeForcedEncryptionOption (ref byte option)
- RETURN_CODE msr_getFunctionStatus (ref bool enabled, ref bool isBufferMode, ref bool withNotification)
- RETURN_CODE device_rebootDevice ()

- RETURN_CODE device_sendVivoCommandP2 (byte command, byte subCommand, byte[] data, ref byte[] response, string ip="")
- RETURN_CODE device_sendVivoCommandP2_ext (byte command, byte subCommand, byte[] data, ref byte[] response, int timeout, bool noResponse, string ip="")
- RETURN_CODE device_sendVivoCommandP3 (byte command, byte subCommand, byte[] data, ref byte[] response, string ip="")
- RETURN_CODE device_sendVivoCommandP3_ext (byte command, byte subCommand, byte[] data, ref byte[] response, int timeout, bool noResponse, string ip="")
- RETURN_CODE device_sendVivoCommandP4 (byte command, byte subCommand, byte[] data, ref byte[] response, string ip="")
- RETURN_CODE device_sendVivoCommandP4_ext (byte command, byte subCommand, byte[] data, ref byte[] response, int timeout, bool noResponse, string ip="")
- RETURN_CODE config_getStatusKeySlots (ref byte[] keyslots)
- RETURN_CODE device_updateDeviceFirmware (byte[] firmwareData)
- RETURN_CODE msr_switchUSBInterfaceMode (bool bIsUSBKeyboardMode)
- RETURN_CODE device_startRKI ()
- void setRKIDelay (int setFeatureDelay, int getFeatureDelay)
- void getRKIDelay (ref int setFeatureDelay, ref int getFeatureDelay)

Static Public Member Functions

- static int getCommandTimeout ()
- static void setCommandTimeout (int milliseconds)
- static void setCallback (CallBack my_Callback)
- static void setCallback (IntPtr my_Callback, SynchronizationContext context)
- static RETURN_CODE device_updateFirmwareType (FIRMWARE_TYPE type, byte[] firmwareData, string ip="")
- static String SDK_Version ()
- static void lcd_retrieveMessage (DisplayMessages.DISPLAY_MESSAGE_LANGUAGE lang, Display↩ Messages.DISPLAY_MESSAGE_IDENTIFIER id, ref string line1, ref string line2)

Properties

- static IDT_SREDKey2 SharedController `[get]`

9.1.1 Detailed Description

Class for SREDKey2 MSR reder

9.1.2 Member Function Documentation

9.1.2.1 RETURN_CODE IDTechSDK.IDT_SREDKey2.config_getModelNumber (ref string *response*)

Polls device for Model Number

Parameters

response	Returns Model Number

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

9.1.2.2 RETURN_CODE IDTechSDK.IDT_SREDKey2.config_getSerialNumber (ref string *response*)

Polls device for Serial Number

Parameters

response	Returns Serial Number

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

9.1.2.3 RETURN_CODE IDTechSDK.IDT_SREDKey2.config_getStatusKeySlots (ref byte[] *keyslots*)

Get status of the key slots

There are 10 keyslots for PIN Key at Index 1, slot #0-9 There are 10 keyslots for DATA Key at Index 2, slots #0-9 There is a keyslot for MAC Key at Index 5, slot 0 There is a keyslot for Key Encryption Key at Index 14, slot 0

This routine will return all the key slots values as one continuous byte[] of 4-byte key slot info.

- Byte 0 = Key Index Number
- Byte 1-2 = Key Slot Number
- Byte 3 = State: 0x00 = Unused, 0x01 = Valid, 0x02 = End of life, 0xFF = Not Available

Parameters

keySlots	Key slot data

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

9.1.2.4 RETURN_CODE IDTechSDK.IDT_SREDKey2.device_enableAdminKey (bool *enable*)

Enable Admin Key

Parameters

enable	TRUE = enable, FALSE = disable

Returns

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

9.1.2.5 RETURN_CODE IDTechSDK.IDT_SREDKey2.device_getFirmwareVersion (ref string *response*)

Polls device for Firmware Version

Parameters

response	Response returned of Firmware Version

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

9.1.2.6 RETURN_CODE IDTechSDK.IDT_SREDKey2.device_getOutputType (ref int *response*)

Get Captured Data Output Type

Parameters

response	• 0 : Original Format • 1 : Enhanced Format • 2 : XML Format

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

9.1.2.7 string IDTechSDK.IDT_SREDKey2.device_getResponseCodeString (RETURN_CODE *eCode*)

Get the description of response result.

Parameters

eCode	the response result.

Return values

the	string for description of response result

- case 0: "no error, beginning task";
- case 1: "no response from reader";
- case 2: "invalid response data";
- case 3: "time out for task or CMD";
- case 4: "wrong parameter";
- case 5: "SDK is doing MSR or ICC task";
- case 6: "SDK is doing PINPad task";
- case 7: "SDK is doing CTLS task";
- case 8: "SDK is doing EMV task";

- case 9: "SDK is doing Other task";
- case 10: "err response or data";
- case 11: "no reader attached";
- case 12: "mono audio is enabled";
- case 13: "did connection";
- case 14: "audio volume is too low";
- case 15: "task or CMD be canceled";
- case 16: "UF wrong string format";
- case 17: "UF file not found";
- case 18: "UF wrong file format";
- case 19: "Attempt to contact online host failed";
- case 20: "Attempt to perform RKI failed";
- case 0x300: "Key Type(TDES) of Session Key is not same as the related Master Key.";
- case 0x400: "Related Key was not loaded.";
- case 0x500: "Key Same.";
- case 0x501: "Key is all zero";
- case 0x502: "TR-31 format error";
- case 0x702: "PAN is Error Key.";
- case 0x705: "No Internal MSR PAN (or Internal MSR PAN is erased timeout)";
- case 0X0C01: "Incorrect Frame Tag";
- case 0X0C02: "Incorrect Frame Type";
- case 0X0C03: "Unknown Frame Type";
- case 0X0C04: "Unknown Command";
- case 0X0C05: "Unknown Sub-Command";
- case 0X0C06: "CRC Error";
- case 0X0C07: "Failed";
- case 0X0C08: "Timeout";
- case 0X0C0A: "Incorrect Parameter";
- case 0X0C0B: "Command Not Supported";
- case 0X0C0C: "Sub-Command Not Supported";
- case 0X0C0D: "Parameter Not Supported / Status Abort Command";
- case 0X0C0F: "Sub-Command Not Allowed";
- case 0X0D01: "Incorrect Header Tag";
- case 0X0D02: "Unknown Command";
- case 0X0D03: "Unknown Sub-Command";
- case 0X0D04: "CRC Error in Frame";

- case 0X0D05: "Incorrect Parameter";
- case 0X0D06: "Parameter Not Supported";
- case 0X0D07: "Mal-formatted Data";
- case 0X0D08: "Timeout";
- case 0X0D0A: "Failed / NACK";
- case 0X0D0B: "Command not Allowed";
- case 0X0D0C: "Sub-Command not Allowed";
- case 0X0D0D: "Buffer Overflow (Data Length too large for reader buffer)";
- case 0X0D0E: "User Interface Event";
- case 0X0D11: "Communication type not supported, VT-1, burst, etc.";
- case 0X0D12: "Secure interface is not functional or is in an intermediate state.";
- case 0X0D13: "Data field is not mod 8";
- case 0X0D14: "Pad 0x80 not found where expected";
- case 0X0D15: "Specified key type is invalid";
- case 0X0D1: "Could not retrieve key from the SAM(InitSecureComm)";
- case 0X0D17: "Hash code problem";
- case 0X0D18: "Could not store the key into the SAM(InstallKey)";
- case 0X0D19: "Frame is too large";
- case 0X0D1A: "Unit powered up in authentication state but POS must resend the InitSecureComm command";
- case 0X0D1B: "The EEPROM may not be initialized because SecCommInterface does not make sense";
- case 0X0D1C: "Problem encoding APDU";
- case 0X0D20: "Unsupported Index(ILM) SAM Transceiver error – problem communicating with the SAM(Key Mgr)";
- case 0X0D2: "Unexpected Sequence Counter in multiple frames for single bitmap(ILM) Length error in data returned from the SAM(Key Mgr)";
- case 0X0D22: "Improper bit map(ILM)";
- case 0X0D23: "Request Online Authorization";
- case 0X0D24: "ViVOCard3 raw data read successful";
- case 0X0D25: "Message index not available(ILM) ViVOcomm activate transaction card type(ViVOcomm)";
- case 0X0D26: "Version Information Mismatch(ILM)";
- case 0X0D27: "Not sending commands in correct index message index(ILM)";
- case 0X0D28: "Time out or next expected message not received(ILM)";
- case 0X0D29: "ILM languages not available for viewing(ILM)";
- case 0X0D2A: "Other language not supported(ILM)";
- case 0X0D41: "Unknown Error from SAM";
- case 0X0D42: "Invalid data detected by SAM";

- case 0X0D43: "Incomplete data detected by SAM";
- case 0X0D44: "Reserved";
- case 0X0D45: "Invalid key hash algorithm";
- case 0X0D46: "Invalid key encryption algorithm";
- case 0X0D47: "Invalid modulus length";
- case 0X0D48: "Invalid exponent";
- case 0X0D49: "Key already exists";
- case 0X0D4A: "No space for new RID";
- case 0X0D4B: "Key not found";
- case 0X0D4C: "Crypto not responding";
- case 0X0D4D: "Crypto communication error";
- case 0X0D4E: "Module-specific error for Key Manager";
- case 0X0D4F: "All key slots are full (maximum number of keys has been installed)";
- case 0X0D50: "Auto-Switch OK";
- case 0X0D51: "Auto-Switch failed";
- case 0X0D90: "Account DUKPT Key not exist";
- case 0X0D91: "Account DUKPT Key KSN exausted";
- case 0x0D00: "This Key had been loaded.";
- case 0x0E00: "Base Time was loaded.";
- case 0x0F00: "Encryption Or Decryption Failed.";
- case 0x1000: "Battery Low Warning (It is High Priority Response while Battery is Low.)";
- case 0x1800: "Send “Cancel Command” after send “Get Encrypted PIN” &”Get Numeric “& “Get Amount””;
- case 0x1900: "Press “Cancel” key after send “Get Encrypted PIN” &”Get Numeric “& “Get Amount””;
- case 0x30FF: "Security Chip is not connect";
- case 0x3000: "Security Chip is deactivation & Device is In Removal Legally State.";
- case 0x3101: "Security Chip is activation & Device is In Removal Legally State.";
- case 0x5500: "No Admin DUKPT Key.";
- case 0x5501: "Admin DUKPT Key STOP.";
- case 0x5502: "Admin DUKPT Key KSN is Error.";
- case 0x5503: "Get Authentication Code1 Failed.";
- case 0x5504: "Validate Authentication Code Error.";
- case 0x5505: "Encrypt or Decrypt data failed.";
- case 0x5506: "Not Support the New Key Type.";
- case 0x5507: "New Key Index is Error.";
- case 0x5508: "Step Error.";
- case 0x5509: "KSN Error";

- case 0x550A: "MAC Error.";
- case 0x550B: "Key Usage Error.";
- case 0x550C: "Mode Of Use Error.";
- case 0x550F: "Other Error.";
- case 0x6000: "Save or Config Failed / Or Read Config Error.";
- case 0x6200: "No Serial Number.";
- case 0x6900: "Invalid Command - Protocol is right, but task ID is invalid.";
- case 0x6A01: "Unsupported Command – Protocol and task ID are right, but command is invalid – In this State";
- case 0x6A00: "Unsupported Command - Protocol and task ID are right, but command is invalid.";
- case 0x6B00: "Unknown parameter in command - Protocol task ID and command are right, but parameter is invalid.";
- case 0x6C00: "Unknown parameter in command – Protocol task ID and command are right, but length is out of the requirement.";
- case 0x7200: "Device is suspend (MKSK suspend or press password suspend).";
- case 0x7300: "PIN DUKPT is STOP (21 bit 1).";
- case 0x7400: "Device is Busy.";
- case 0xE100: "Can not enter sleep mode";
- case 0xE200: "File has existed";
- case 0xE300: "File has not existed";
- case 0xE313: "IO line low -- Card error after session start";
- case 0xE400: "Open File Error";
- case 0xE500: "SmartCard Error";
- case 0xE600: "Get MSR Card data is error";
- case 0xE700: "Command time out";
- case 0xE800: "File read or write is error";
- case 0xE900: "Active 1850 error!";
- case 0xEA00: "Load bootloader error";
- case 0xEF00: "Protocol Error- STX or ETX or check error.";
- case 0xEB00: "Picture is not exist";
- case 0x2C02: "No Microprocessor ICC seated";
- case 0x2C06: "no card seated to request ATR";
- case 0x2D01: "Card Not Supported,";
- case 0x2D03: "Card Not Supported, wants CRC";
- case 0x690D: "Command not supported on reader without ICC support";
- case 0x8100: "ICC error time out on power-up";
- case 0x8200: "invalid TS character received - Wrong operation step";

- case 0x8300: "Decode MSR Error";
- case 0x8400: "TriMagII no Response";
- case 0x8500: "No Swipe MSR Card";
- case 0x8510: "No Financial Card";
- case 0x8600: "Unsupported F, D, or combination of F and D";
- case 0x8700: "protocol not supported EMV TD1 out of range";
- case 0x8800: "power not at proper level";
- case 0x8900: "ATR length too long";
- case 0x8B01: "EMV invalid TA1 byte value";
- case 0x8B02: "EMV TB1 required";
- case 0x8B03: "EMV Unsupported TB1 only 00 allowed";
- case 0x8B04: "EMV Card Error, invalid BWI or CWI";
- case 0x8B06: "EMV TB2 not allowed in ATR";
- case 0x8B07: "EMV TC2 out of range";
- case 0x8B08: "EMV TC2 out of range";
- case 0x8B09: "per EMV96 TA3 must be $>$ 0xF";
- case 0x8B10: "ICC error on power-up";
- case 0x8B11: "EMV T=1 then TB3 required";
- case 0x8B12: "Card Error, invalid BWI or CWI";
- case 0x8B13: "Card Error, invalid BWI or CWI";
- case 0x8B17: "EMV TC1/TB3 conflict-";
- case 0x8B20: "EMV TD2 out of range must be T=1";
- case 0x8C00: "TCK error";
- case 0xA304: "connector has no voltage setting";
- case 0xA305: "ICC error on power-up invalid (SBLK(IFSD) exchange";
- case 0xE301: "ICC error after session start";
- case 0xFF00: "Request to go online";
- case 0xFF01: "EMV: Accept the offline transaction";
- case 0xFF02: "EMV: Decline the offline transaction";
- case 0xFF03: "EMV: Accept the online transaction";
- case 0xFF04: "EMV: Decline the online transaction";
- case 0xFF05: "EMV: Application may fallback to magstripe technology";
- case 0xFF06: "EMV: ICC detected tah the conditions of use are not satisfied";
- case 0xFF07: "EMV: ICC didn't accept transaction";
- case 0xFF08: "EMV: Transaction was cancelled";
- case 0xFF09: "EMV: Application was not selected by kernel or ICC format error or ICC missing data error";

- case 0xFF0A: "EMV: Transaction is terminated";
- case 0xFF0B: "EMV: Other EMV Error";
- case 0xFFFF: "NO RESPONSE";
- case 0xF002: "ICC communication timeout";
- case 0xF003: "ICC communication Error";
- case 0xF00F: "ICC Card Seated and Highest Priority, disable MSR work request";
- case 0xF200: "AID List / Application Data is not exist";
- case 0xF201: "Terminal Data is not exist";
- case 0xF202: "TLV format is error";
- case 0xF203: "AID List is full";
- case 0xF204: "Any CA Key is not exist";
- case 0xF205: "CA Key RID is not exist";
- case 0xF206: "CA Key Index it not exist";
- case 0xF207: "CA Key is full";
- case 0xF208: "CA Key Hash Value is Error";
- case 0xF209: "Transaction format error";
- case 0xF20A: "The command will not be processing";
- case 0xF20B: "CRL is not exist";
- case 0xF20C: "CRL number exceed max number";
- case 0xF20D: "Amount,Other Amount,Trasaction Type are missing";
- case 0xF20E: "The Identification of algorithm is mistake";
- case 0xF20F: "No Financial Card";
- case 0xF210: "In Encrypt Result state, TLV total Length is greater than Max Length";
- case 0x1001: "INVALID ARG";
- case 0x1002: "FILE_OPEN_FAILED";
- case 0x1003: "FILE OPERATION_FAILED";
- case 0x2001: "MEMORY_NOT_ENOUGH";
- case 0x3002: "SMARTCARD_FAIL";
- case 0x3003: "SMARTCARD_INIT_FAILED";
- case 0x3004: "FALLBACK_SITUATION";
- case 0x3005: "SMARTCARD_ABSENT";
- case 0x3006: "SMARTCARD_TIMEOUT";
- case 0x5001: "EMV_PARSING_TAGS_FAILED";
- case 0x5002: "EMV_DUPLICATE_CARD_DATA_ELEMENT";
- case 0x5003: "EMV_DATA_FORMAT_INCORRECT";
- case 0x5004: "EMV_NO_TERM_APP";

- case 0x5005: "EMV_NO_MATCHING_APP";
- case 0x5006: "EMV_MISSING_MANDATORY_OBJECT";
- case 0x5007: "EMV_APP_SELECTION_RETRY";
- case 0x5008: "EMV_GET_AMOUNT_ERROR";
- case 0x5009: "EMV_CARD_REJECTED";
- case 0x5010: "EMV_AIP_NOT_RECEIVED";
- case 0x5011: "EMV_AFL_NOT_RECEIVED";
- case 0x5012: "EMV_AFL_LEN_OUT_OF_RANGE";
- case 0x5013: "EMV_SFI_OUT_OF_RANGE";
- case 0x5014: "EMV_AFL_INCORRECT";
- case 0x5015: "EMV_EXP_DATE_INCORRECT";
- case 0x5016: "EMV_EFF_DATE_INCORRECT";
- case 0x5017: "EMV_ISS_COD_TBL_OUT_OF_RANGE";
- case 0x5018: "EMV_CRYPTOGRAM_TYPE_INCORRECT";
- case 0x5019: "EMV_PSE_NOT_SUPPORTED_BY_CARD";
- case 0x5020: "EMV_USER_SELECTED_LANGUAGE";
- case 0x5021: "EMV_SERVICE_NOT_ALLOWED";
- case 0x5022: "EMV_NO_TAG_FOUND";
- case 0x5023: "EMV_CARD_BLOCKED";
- case 0x5024: "EMV_LEN_INCORRECT";
- case 0x5025: "CARD_COM_ERROR";
- case 0x5026: "EMV_TSC_NOT_INCREASED";
- case 0x5027: "EMV_HASH_INCORRECT";
- case 0x5028: "EMV_NO_ARC";
- case 0x5029: "EMV_INVALID_ARC";
- case 0x5030: "EMV_NO_ONLINE_COMM";
- case 0x5031: "TRAN_TYPE_INCORRECT";
- case 0x5032: "EMV_APP_NO_SUPPORT";
- case 0x5033: "EMV_APP_NOT_SELECT";
- case 0x5034: "EMV_LANG_NOT_SELECT";
- case 0x5035: "EMV_NO_TERM_DATA";
- case 0x6001: "CVM_TYPE_UNKNOWN";
- case 0x6002: "CVM_AIP_NOT_SUPPORTED";
- case 0x6003: "CVM_TAG_8E_MISSING";
- case 0x6004: "CVM_TAG_8E_FORMAT_ERROR";
- case 0x6005: "CVM_CODE_IS_NOT_SUPPORTED";

- case 0x6006: "CVM_COND_CODE_IS_NOT_SUPPORTED";
- case 0x6007: "NO_MORE_CVM";
- case 0x6008: "PIN_BYPASSED_BEFORE";
- case 0x7001: "PK_BUFFER_SIZE_TOO_BIG";
- case 0x7002: "PK_FILE_WRITE_ERROR";
- case 0x7003: "PK_HASH_ERROR";
- case 0x8001: "NO_CARD_HOLDER_CONFIRMATION";
- case 0x8002: "GET_ONLINE_PIN";
- case 0xD000: "Data not exist";
- case 0xD001: "Data access error";
- case 0xD100: "RID not exist";
- case 0xD101: "RID existed";
- case 0xD102: "Index not exist";
- case 0xD200: "Maximum exceeded";
- case 0xD201: "Hash error";
- case 0xD205: "System Busy";
- case 0x0E01: "Unable to go online";
- case 0x0E02: "Technical Issue";
- case 0x0E03: "Declined";
- case 0x0E04: "Issuer Referral transaction";
- case 0x0F01: "Decline the online transaction";
- case 0x0F02: "Request to go online";
- case 0x0F03: "Transaction is terminated";
- case 0x0F05: "Application was not selected by kernel or ICC format error or ICC missing data error";
- case 0x0F07: "ICC didn't accept transaction";
- case 0x0F0A: "Application may fallback to magstripe technology";
- case 0x0F0C: "Transaction was cancelled";
- case 0x0F0D: "Timeout";
- case 0x0F0F: "Other EMV Error";
- case 0x0F10: "Accept the offline transaction";
- case 0x0F11: "Decline the offline transaction";
- case 0x0F21: "ICC detected tah the conditions of use are not satisfied";
- case 0x0F22: "No app were found on card matching terminal configuration";
- case 0x0F23: "Terminal file does not exist";
- case 0x0F24: "CAPK file does not exist";
- case 0x0F25: "CRL Entry does not exist";

- case 0x0FFE: "code when blocking is disabled";
- case 0x0FFF: "code when command is not applicable on the selected device";
- case 0xF005: "ICC Encrypted C-APDU Data Structure Length Error Or Format Error.";
- case 0xBBE0: "CM100 Success";
- case 0xBBE1: "CM100 Parameter Error";
- case 0xBBE2: "CM100 Low Output Buffer";
- case 0xBBE3: "CM100 Card Not Found";
- case 0xBBE4: "CM100 Collision Card Exists";
- case 0xBBE5: "CM100 Too Many Cards Exist";
- case 0xBBE6: "CM100 Saved Data Does Not Exist";
- case 0xBBE8: "CM100 No Data Available";
- case 0xBBE9: "CM100 Invalid CID Returned";
- case 0xBBEA: "CM100 Invalid Card Exists";
- case 0xBBEC: "CM100 Command Unsupported";
- case 0xBBED: "CM100 Error In Command Process";
- case 0xBBEE: "CM100 Invalid Command";
- case 0X9031: "Unknown command";
- case 0X9032: "Wrong parameter (such as the length of the command is incorrect)";
- case 0X9038: "Wait (the command couldn't be finished in BWT)";
- case 0X9039: "Busy (a previously command has not been finished)";
- case 0X903A: "Number of retries over limit";
- case 0X9040: "Invalid Manufacturing system data";
- case 0X9041: "Not authenticated";
- case 0X9042: "Invalid Master DUKPT Key";
- case 0X9043: "Invalid MAC Key";
- case 0X9044: "Reserved for future use";
- case 0X9045: "Reserved for future use";
- case 0X9046: "Invalid DATA DUKPT Key";
- case 0X9047: "Invalid PIN Pairing DUKPT Key";
- case 0X9048: "Invalid DATA Pairing DUKPT Key";
- case 0X9049: "No nonce generated";
- case 0X9949: "No GUID available. Perform getVersion first.";
- case 0X9950: "MAC Calculation unsuccessful. Check BDK value.";
- case 0X904A: "Not ready";
- case 0X904B: "Not MAC data";
- case 0X9050: "Invalid Certificate";

- case 0X9051: "Duplicate key detected";
- case 0X9052: "AT checks failed";
- case 0X9053: "TR34 checks failed";
- case 0X9054: "TR31 checks failed";
- case 0X9055: "MAC checks failed";
- case 0X9056: "Firmware download failed";
- case 0X9060: "Log is full";
- case 0X9061: "Removal sensor unengaged";
- case 0X9062: "Any hardware problems";
- case 0X9070: "ICC communication timeout";
- case 0X9071: "ICC data error (such check sum error)";
- case 0X9072: "Smart Card not powered up";

9.1.2.8 RETURN_CODE IDTechSDK.IDT_SREDKey2.device_getTransArmorID (ref string *TID*)

Get TransArmor ID

Parameters

TID	TransArmor ID

Returns

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

9.1.2.9 RETURN_CODE IDTechSDK.IDT_SREDKey2.device_rebootDevice ( )

Reboot Device

Executes a command to restart the device.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

9.1.2.10 RETURN_CODE IDTechSDK.IDT_SREDKey2.device_sendDataCommand (string *cmd,* bool *calcLRC,* ref byte[ ] *response*)

Send a data command to the device

Sends a command to the device.

Parameters

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

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

9.1.2.11 RETURN_CODE IDTechSDK.IDT_SREDKey2.device_sendDataCommand_ext (string *cmd,* bool *calcLRC,* ref byte[ ] *response,* int *timeout,* bool *noResponse*)

Send a data command to the device - extended

Sends a command to the device.

Parameters

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

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

9.1.2.12 RETURN_CODE IDTechSDK.IDT_SREDKey2.device_sendVivoCommandP2 (byte *command,* byte *subCommand,* byte[ ] *data,* ref byte[ ] *response*)

Send Vivo Command Protocol 2

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

Parameters

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

Returns

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

9.1.2.13 RETURN_CODE IDTechSDK.IDT_SREDKey2.device_sendVivoCommandP2 (byte *command,* byte *subCommand,* byte[ ] *data,* ref byte[ ] *response,* string *ip* = " ")

Send Vivo Command Protocol 2

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

Parameters

command	Command

Parameters

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

Returns

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

9.1.2.14 RETURN_CODE IDTechSDK.IDT_SREDKey2.device_sendVivoCommandP2_ext (byte *command,* byte *subCommand,* byte[ ] *data,* ref byte[ ] *response,* int *timeout,* bool *noResponse*)

Send Vivo Command Protocol 2 Extended

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

Parameters

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

Returns

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

9.1.2.15 RETURN_CODE IDTechSDK.IDT_SREDKey2.device_sendVivoCommandP2_ext (byte *command,* byte *subCommand,* byte[ ] *data,* ref byte[ ] *response,* int *timeout,* bool *noResponse,* string *ip* = " ")

Send Vivo Command Protocol 2 Extended

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

Parameters

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

Returns

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

9.1.2.16 RETURN_CODE IDTechSDK.IDT_SREDKey2.device_sendVivoCommandP3 (byte *command,* byte *subCommand,* byte[ ] *data,* ref byte[ ] *response*)

Send Vivo Command Protocol 3

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

Parameters

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

Returns

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

9.1.2.17 RETURN_CODE IDTechSDK.IDT_SREDKey2.device_sendVivoCommandP3 (byte *command,* byte *subCommand,* byte[ ] *data,* ref byte[ ] *response,* string *ip* = " ")

Send Vivo Command Protocol 3

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

Parameters

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

Returns

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

9.1.2.18 RETURN_CODE IDTechSDK.IDT_SREDKey2.device_sendVivoCommandP3_ext (byte *command,* byte *subCommand,* byte[ ] *data,* ref byte[ ] *response,* int *timeout,* bool *noResponse*)

Send Vivo Command Protocol 3 Extended

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

Parameters

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

Returns

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

9.1.2.19 RETURN_CODE IDTechSDK.IDT_SREDKey2.device_sendVivoCommandP3_ext (byte *command,* byte *subCommand,* byte[ ] *data,* ref byte[ ] *response,* int *timeout,* bool *noResponse,* string *ip =* " ")

Send Vivo Command Protocol 3 Extended

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

Parameters

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

Returns

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

9.1.2.20 RETURN_CODE IDTechSDK.IDT_SREDKey2.device_sendVivoCommandP4 (byte *command,* byte *subCommand,* byte[ ] *data,* ref byte[ ] *response,* string *ip =* " ")

Send Vivo Command Protocol 4

Sends a protocol 4 command to Vivo readers (IDG/NEO)

Parameters

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

Returns

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

9.1.2.21 RETURN_CODE IDTechSDK.IDT_SREDKey2.device_sendVivoCommandP4_ext (byte *command,* byte *subCommand,* byte[ ] *data,* ref byte[ ] *response,* int *timeout,* bool *noResponse,* string *ip =* " ")

Send Vivo Command Protocol 4 Extended

Sends a protocol 4 command to Vivo readers (IDG/NEO)

Parameters

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

Returns

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

9.1.2.22 IDTechSDK.RETURN_CODE IDTechSDK.IDT_SREDKey2.device_setLanguage (int *val*)

Set Device Language.

Parameters

val	Language: 0 = English, 1 = Japanese

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

9.1.2.23 RETURN_CODE IDTechSDK.IDT_SREDKey2.device_setOutputType (int *value*)

Set Captured Data Output Type

Parameters

value	• 0 : Original Format • 1 : Enhanced Format • 2 : XML Format

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

9.1.2.24 RETURN_CODE IDTechSDK.IDT_SREDKey2.device_setTransArmorID (string *TID*)

Set TransArmor ID

Parameters

TID	TransArmor ID

Returns

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

9.1.2.25 RETURN_CODE IDTechSDK.IDT_SREDKey2.device_startRKI ( )

Start Remote Key Injection

Starts a remote key injection request with IDTech RKI servers.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

9.1.2.26 RETURN_CODE IDTechSDK.IDT_SREDKey2.device_updateDeviceFirmware (byte[] *firmwareData*)

Update K81 Firmware

Updates the firmware .

Parameters

firmwareData	Signed binary data of a firmware file provided by IDTech

Returns

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

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

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

Example code starting a firmware update

```
OpenFileDialog diag = new OpenFileDialog();

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

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

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

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

Example monitoring firmware update status / success

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

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

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

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

9.1.2.27 static RETURN_CODE IDTechSDK.IDT_SREDKey2.device_updateFirmwareType (FIRMWARE_TYPE *type,* byte[] *firmwareData,* string *ip* = "") [static]

Update App Firmware

Updates the firmware

Parameters

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

Returns

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

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

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

Example code starting a firmware update

```
OpenFileDialog diag = new OpenFileDialog();

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

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

    byte[] file = File.ReadAllBytes(diag.FileName);
    RETURN_CODE rt = IDT_Device.SharedController.device_updateFirmware(FIRMWARE_TYPE.FIRMWARE_TYPE_1050,
      file);
    if (rt == RETURN_CODE.RETURN_CODE_DO_SUCCESS)
    {

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

Example monitoring firmware update status / success

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

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

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

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

9.1.2.28 static int IDTechSDK.IDT_SREDKey2.getCommandTimeout ( ) [static]

Get Command Timeout

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

Return values

time	Time

9.1.2.29 void IDTechSDK.IDT_SREDKey2.getRKIDelay (ref int *setFeatureDelay,* ref int *getFeatureDelay*)

Get Remote Key Injection Delay

Returns the timing parameters when executing 90-03 command.

Parameters

setFeatureDelay	Time, in milliseconds, that will be inserted before SetFeatureReport is executed for 90-03 command
getFeatureDelay	Time, in milliseconds, that will be inserted before GetFeatureReport in response to a 90-03 command

9.1.2.30 static void IDTechSDK.IDT_SREDKey2.lcd_retrieveMessage (DisplayMessages.DISPLAY_MESSAGE_LANGUAGE *lang,* DisplayMessages.DISPLAY_MESSAGE_IDENTIFIER *id,* ref string *line1,* ref string *line2*) [static]

Retrieve LCD Message

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

Parameters

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

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

9.1.2.31 RETURN_CODE IDTechSDK.IDT_SREDKey2.msr_disable ( )

Disable MSR function.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

9.1.2.32 RETURN_CODE IDTechSDK.IDT_SREDKey2.msr_enable ( )

Disable MSR function.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

9.1.2.33 RETURN_CODE IDTechSDK.IDT_SREDKey2.msr_getClearPANID (ref byte *value*)

Get Clear PAN Digits

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

Parameters

value	Number of digits in clear. Values are char '0' - '6':

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

9.1.2.34 RETURN_CODE IDTechSDK.IDT_SREDKey2.msr_getExpirationMask (ref byte *value*)

Get Expiration Masking

Get the flag that determines if to mask the expiration date

Parameters

value	'0' = masked, '1' = not-masked

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

9.1.2.35 RETURN_CODE IDTechSDK.IDT_SREDKey2.msr_getFunctionStatus (ref bool *enabled,* ref bool *isBufferMode,* ref bool *withNotification*)

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

Parameters

enabled	• true: MSR Function enabled. • false: MSR Function disabled.
isBufferMode	• true: in the buffer mode. • false: in the auto mode.
withNotification	• true: with notification when swiped MSR Card. • false: without notification when swiped MSR Card.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

9.1.2.36 RETURN_CODE IDTechSDK.IDT_SREDKey2.msr_getSwipeEncryption (ref byte *encryption*)

Get Swipe Data Encryption

For Non-SRED Augusta Only

Returns the encryption used for swipe data

Parameters

encryption	1 = TDES, 2 = AES, 0 = NONE

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

9.1.2.37 RETURN_CODE IDTechSDK.IDT_SREDKey2.msr_getSwipeForcedEncryptionOption (ref byte *option*)

Get Swipe Data Encryption

Gets the swipe force encryption options

Parameters

option	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

Example: Response 0x03 = Track1/Track2 Forced Encryption, Track3/Track3-0 no Forced Encryption

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

9.1.2.38 RETURN_CODE IDTechSDK.IDT_SREDKey2.msr_getSwipeMaskOption (ref byte *option*)

Get Swipe Mask Option

Gets the swipe mask/clear data sending option

Parameters

option	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

9.1.2.39 RETURN_CODE IDTechSDK.IDT_SREDKey2.msr_setClearPANID (byte *val*)

Set Clear PAN Digits

Sets the amount of digits shown in the clear (not masked) at the beginning of the returned PAN value

Parameters

val	Number of digits to show in clear. Range 0-6.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

9.1.2.40 RETURN_CODE IDTechSDK.IDT_SREDKey2.msr_setExpirationMask (bool *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

9.1.2.41 RETURN_CODE IDTechSDK.IDT_SREDKey2.msr_setSwipeForcedEncryptionOption (bool *track1,* bool *track2,* bool *track3,* bool *track3card0*)

Set Swipe Force Encryption

Sets the swipe force encryption options

Parameters

track1	Force encrypt track 1
track2	Force encrypt track 2
track3	Force encrypt track 3
track3card0	Force encrypt track 3 when card type is 0

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

9.1.2.42 RETURN_CODE IDTechSDK.IDT_SREDKey2.msr_setSwipeMaskOption (bool *track1,* bool *track2,* bool *track3*)

Set Swipe Mask Option

Sets the swipe mask/clear data sending option

Parameters

track1	Mask track 1 allowed
track2	Mask track 2 allowed
track3	Mask track 3 allowed

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

9.1.2.43 RETURN_CODE IDTechSDK.IDT_SREDKey2.msr_switchUSBInterfaceMode (bool *blsUSBKeyboardMode*)

Switch the USB interface mode between USB HID and USB KB mode.

For Non-SRED Augusta Only

Parameters

blsUSBKeyboardMode	USB interface mode • true: Enter into USB Keyboard mode • false: Enter into USB HID mode

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

9.1.2.44 static String IDTechSDK.IDT_SREDKey2.SDK_Version ( ) [static]

SDK Version

- All Devices

Returns the current version of SDK

Returns

Framework version

9.1.2.45 static void IDTechSDK.IDT_SREDKey2.setCallback (CallBack *my_Callback*) [static]

Set Callback

Sets the class callback

Parameters

my_Callback	The callback function to receive the response message from device. defined as follows. delegate void CallBack(IDT_DEVICE_Types sender, DeviceState state, byte[] data, IDTTransactionData card, EMV_Callback emvCallback, RETURN_CODE transactionResultCode);

9.1.2.46 static void IDTechSDK.IDT_SREDKey2.setCallback (IntPtr *my_Callback,* SynchronizationContext *context*) [static]

Set Callback

Sets the class callback

Parameters

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

9.1.2.47 static void IDTechSDK.IDT_SREDKey2.setCommandTimeout (int *milliseconds*) [static]

Set Command Timeout

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

Parameters

milliseconds	Time

9.1.2.48 void IDTechSDK.IDT_SREDKey2.setRKIDelay (int *setFeatureDelay,* int *getFeatureDelay*)

Set Remote Key Injection Delay

During Remote Key Injection, it may be necessary to adjust the timing parameters when executing 90-03 command.

Parameters

setFeatureDelay	Time, in milliseconds, that will be inserted before SetFeatureReport is executed for 90-03 command
getFeatureDelay	Time, in milliseconds, that will be inserted before GetFeatureReport in response to a 90-03 command

9.1.3 Property Documentation

9.1.3.1 IDT_SREDKey2 IDTechSDK.IDT_SREDKey2.SharedController [static],[get]

Singleton Instance

Establishes an singleton instance of IDT_SREDKey2 class.

Returns

Instance of IDT_SREDKey2

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

- Source/IDT_SREDKey2.cs

Index