DTDevices

Generated by Doxygen 1.8.3.1

Tue Mar 4 2014 17:43:09

Contents

Chapter 1

Deprecated List

Member [DTDevices btDiscoverDevices:maxTime:codTypes:error:]

This function is not recommended to be called on the main thread, use btDiscoverDevicesInBackground instead

Member [DTDevices btGetDeviceName:error:]

This function complements the btDiscoverDevices/btDiscoverPrinters and as such is not recommended, use btDiscoverDevicesInBackground instead

Chapter 2

Module Documentation

2.1 Magnetic Card Encryption Algorithms

Supported encryption algorithms for magnetic track data on various devices.

Macros

- #define ALG_AES256 0

  AES 256 encryption algorithm.
- #define ALG_EH_ECC 1

  Encrypted Head ECC encryption algorithm.
- #define ALG_EH_AES256 2

  Encrypted Head AES256 encryption algorithm.
- #define ALG_EH_IDTECH 3

  Encrypted Head IDTECH encryption algorithm, please refer to IDTECH documentation for detailed format and examples.
- #define ALG_EH_IDTECH_AES128 0x0b

  Encrypted Head IDTECH encryption algorithm, please refer to IDTECH documentation for detailed format and examples.
- #define ALG_EH_MAGTEK 4

  Encrypted Head MAGTEK encryption algorithm.
- #define ALG_EH_MAGTEK_AES128 0x0c

  Encrypted Head MAGTEK encryption algorithm.
- #define ALG_EH_3DES 5

  Encrypted Head 3DES encryption algorithm.
- #define ALG_EH_RSA_OAEP 6

  Encrypted Head RSA encryption algorithm.
- #define ALG_PPAD_3DES_CBC 7

  Pinpad 3DES format, containing:
- #define ALG_EH_VOLTAGE 8

  Encrypted Head Voltage encryption algorithm.
- #define ALG_EH_AES128 9

  Encrypted Head AES128 encryption algorithm.
- #define ALG_PPAD_DUKPT 10

  Pinpad DUKPT format, containing:

2.1.1 Detailed Description

Supported encryption algorithms for magnetic track data on various devices.

2.1.2 Macro Definition Documentation

2.1.2.1 #define ALG_EH_AES128 9

Encrypted Head AES128 encryption algorithm.

Encryption type is CBC. After decryption, the result data will be as follows:

- Random data (4 bytes)
- Device identification text (16 ASCII characters, unused bytes are 0)
- Processed track data in the format: 0xF1 (track1 data), 0xF2 (track2 data) 0xF3 (track3 data). It is possible some of the tracks will be empty, then the identifier will not be present too, for example 0xF1 (track1 data) 0xF3 (track3 data)
- End of track data (byte 0x00)
- CRC16CCIT (2 bytes) - the CRC is performed from the start of the encrypted block (the Random Data block) to the end of the track data (including the 0x00 byte). The data block is rounded to 16 bytes

2.1.2.2 #define ALG_EH_AES256 2

Encrypted Head AES256 encryption algorithm.

Encryption type is CBC. After decryption, the result data will be as follows:

- Random data (4 bytes)
- Device identification text (16 ASCII characters, unused bytes are 0)
- Processed track data in the format: 0xF1 (track1 data), 0xF2 (track2 data) 0xF3 (track3 data). It is possible some of the tracks will be empty, then the identifier will not be present too, for example 0xF1 (track1 data) 0xF3 (track3 data)
- End of track data (byte 0x00)
- CRC16CCIT (2 bytes) - the CRC is performed from the start of the encrypted block (the Random Data block) to the end of the track data (including the 0x00 byte). The data block is rounded to 16 bytes

2.1.2.3 #define ALG_EH_IDTECH 3

Encrypted Head IDTECH encryption algorithm, please refer to IDTECH documentation for detailed format and examples.

Data, that is received via magneticCardEncryptedData has the following format:

- (1 byte) card encoding type, can ignore
- (1 byte) bits marking which track is present
- (1 byte) track 1 UNENCRYPTED length
- (1 byte) track 2 UNENCRYPTED length
- (1 byte) track 3 UNENCRYPTED length

- (track 1 UNENCRYPTED length bytes) track 1 masked data
- (track 2 UNENCRYPTED length bytes) track 2 masked data
- (track 3 UNENCRYPTED length bytes) track 3 masked data
- (variable bytes) Track 1 + Track 2 encrypted data, the length of this block is calculated by substracting from the end
- (20 bytes) track 1 sha1
- (20 bytes) track 2 sha1
- (10 bytes) KSN

Encrypted block contents after decryption (3DES):

- (track 1 UNENCRYPTED length bytes) track 1 data
- (track 2 UNENCRYPTED length bytes) track 2 data

2.1.2.4 #define ALG_EH_IDTECH_AES128 0x0b

Encrypted Head IDTECH encryption algorithm, please refer to IDTECH documentation for detailed format and examples.

Data, that is received via magneticCardEncryptedData has the following format:

- (1 byte) card encoding type, can ignore
- (1 byte) bits marking which track is present
- (1 byte) track 1 UNENCRYPTED length
- (1 byte) track 2 UNENCRYPTED length
- (1 byte) track 3 UNENCRYPTED length
- (track 1 UNENCRYPTED length bytes) track 1 masked data
- (track 2 UNENCRYPTED length bytes) track 2 masked data
- (track 3 UNENCRYPTED length bytes) track 3 masked data
- (variable bytes) Track 1 + Track 2 encrypted data, the length of this block is calculated by substracting from the end
- (20 bytes) track 1 sha1
- (20 bytes) track 2 sha1
- (10 bytes) KSN

Encrypted block contents after decryption (AS128):

- (track 1 UNENCRYPTED length bytes) track 1 data
- (track 2 UNENCRYPTED length bytes) track 2 data

With AES128 version, the normal 3DES DUKPT keys are used, but the encryption algorithm is AES128

2.1.2.5 #define ALG_EH_MAGTEK_AES128 0x0c

Encrypted Head MAGTEK encryption algorithm.

With AES128 version, the normal 3DES DUKPT keys are used, but the encryption algorithm is AES128

2.1.2.6 #define ALG_EH_VOLTAGE 8

Encrypted Head Voltage encryption algorithm.

The Voltage SecureData Payments consolidated message format is designed to be self-contained. The data packet provides all the information necessary to enable decryption at the Host SDK. It is also self-describing, meaning that the recipient can reliably interpret the contents independent of any other information, even though the specific data elements may vary between messages.

The message format is strictly text-based and can be transmitted to the Host even via a web form, because the design deliberately avoids characters and character sequences that may otherwise need to be escaped in such environments.

The message structure borrows concepts from swipe card Track data specifications, specifically the use of start and end sentinels, together with an internal field separator that delimit the contents. These delimiters, particularly the field separator (|) and end sentinel (∼), have been chosen to avoid any possible conflict with ciphertext values that may be generated using any version of the Voltage SecureData Payments encryption protocols.

The message structure allows for all currently supported card data element types (PAN, MID, and Tracks 1, 2 and 3), included in any combination. It is highly flexible, allowing for additional data elements to be incorporated, or for substantially new formats defined for future use.

The packet looks like: _a[FLAGS]|[PAN]|[MID]|[TRACK1]|[TRACK2]|[TRACK3]|[EXP]|[APP]|[ETB]∼

Fields:

"_"	Start Sentinel (1 char)	Start of message is always indicated by an underscore.
"a"	Format Version (1 char)	Single letter is used to identify the format version.
"nn"	Header (2 chars hex)	The header indicates which optional data fields, of those defined in this format version, are included in the message. This comprises two hex digits interpreted as explained in the next section.
"|"	Field Separator (1 char)	Vertical pipe is used as a field separator between message elements. A separator is required following the header to allow this to be of variable length.
"Data(1)"	First Encrypted Data Field (variable)	One or more encrypted data fields are optionally included in the sequence defined for the format version.
"|"	Field Separator (1 char)	
"Data(2)"	Second Encrypted Data Field (variable)	
"|"	Field Separator (1 char)	
...		

"|"	Field Separator (1 char)	
"Base46"	Base64-Encoded ETB (variable)	he Encryption Transmission Block must be Base64-Encoded prior to inclusion in the message.
"∼"	End Sentinel (1 char)	The end of the ETB is always indicated by a tilde character

Encrypted card data elements always appear in the message in a defined sequence, starting with PAN, MID, Track 1, and the rest. The relative position of these fields, delimited by field separators is always fixed although any combination of values is legal. Unused fields are omitted together with their terminating separator. The ciphertext format may vary depending on the encryption algorithm used. For example, Tracks 1 and 2 may use whole-track encryption (TEP1) or structure-preserving encryption (TEP2). Elements can miss from the packet, i.e. if Track 3 was not read, then it will not be in the packet and [FLAGS] will have TR3 bit cleared(0).

Data Fields and Headers:

"1"	PAN	Primary Account Number
"2"	MID	Merchant ID
"3"	TR1	Track 1 data (IATA)
"4"	TR2	Track 2 data (ABA)
"5"	TR3	Track 3 data (THRIFT-TTS)
"6"	EXP	Card expiration data in the form MMYY
"7"	APP	Reserved for application-specific use (unsupported)

The header value comprises two hexadecimal digits that together form an 8 bit mask indicating the data fields present in the associated message

FLAGS Bit Mask Value:

"Bit 7 (most significant)"	APP (unsupported)
"Bit 6"	EXP
"Bit 5"	TR3
"Bit 4"	TR2
"Bit 3"	TR1
"Bit 2"	MID
"Bit 1"	PAN
"Bit 0 (least significant)"	ETB

2.1.2.7 #define ALG_PPAD_3DES_CBC 7

Pinpad 3DES format, containing:

- random data (4 bytes)
- unique ID (4 bytes) - same ID you have sent to the function
- payload length (2 bytes) - length of the TLV block in BIG ENDIAN
- card data (variable, ends with 0x00), in format 0xF1 <track1> 0xF2 <track3> 0xF3 <track3> (some tracks might me missing, in this case indentifier is missing too)
- crc (2 bytes) - CRC16 CCIT on all the bytes before it
- padding (0-7 bytes) zeroes to pad the packet with

2.1.2.8 #define ALG_PPAD_DUKPT 10

Pinpad DUKPT format, containing:

- random data (4 bytes)

- unique ID (4 bytes) - same ID you have sent to the function
- payload length (2 bytes) - length of the TLV block in BIG ENDIAN
- card data (variable, ends with 0x00), in format 0xF1 <track1> 0xF2 <track3> 0xF3 <track3> (some tracks might me missing, in this case indentifier is missing too)
- crc (2 bytes) - CRC16 CCIT on all the bytes before it
- padding (0-7 bytes) zeroes to pad the packet with

2.2 Library Error Codes

Library error codes returned in the NSError objects.

Macros

- #define **Library_Errors_h**
- #define DT_ENONE 0

  Operation successful.
- #define DT_EGENERAL -1

  General error / Unknown error.
- #define DT_ECREATE -2

  Create error.
- #define DT_EOPEN -3

  Open error.
- #define DT_ECLOSE -4

  Close error.
- #define DT_EBUSY -5

  Device or resource busy.
- #define DT_ETIMEOUT -6

  Timeout expired.
- #define DT_ENOSUPPORTED -7

  Unsupported method or operation.
- #define DT_EMEMORY -8

  Memory allocation error.
- #define DT_EPARAM -9

  Invalid parameter.
- #define DT_EIO -10

  Input/Output error.
- #define DT_ECRC -11

  CRC error.
- #define DT_EFLASH -12

  Flash error.
- #define DT_EEEPROM -13

  EEPROM error.
- #define DT_EDEVICE -14

  Device error.
- #define DT_ENOIMPLEMENTED -15

  The operation is not implemented.
- #define DT_ENOEXIST -16

  The device or resource does not exists.
- #define DT_EINVALID_CMD -17

  Invalid command.
- #define DT_ENOT_EXIST_OBJECT -18

  Not exist object.
- #define DT_ENOMORE -19

  No more items.
- #define DT_EFAILED -20

  Command Failed.
- #define DT_EINVALID -21

Invalid command.
- #define DT_ENOT_REGISTERED -22

  Not registered.
- #define DT_EPERMISSION_DENIED -23

  Permission denied.
- #define DT_MIFARE_EBASE -10000

  Mifare operation successful.
- #define DT_MIFARE_ETIMEOUT DT_MIFARE_EBASE-1

  Mifare timeout error.
- #define DT_MIFARE_ECOLLISION DT_MIFARE_EBASE-2

  Mifare collision error.
- #define DT_MIFARE_EPARITY DT_MIFARE_EBASE-3

  Mifare parity error.
- #define DT_MIFARE_EFRAME DT_MIFARE_EBASE-4

  Mifare frame error.
- #define DT_MIFARE_ECRC DT_MIFARE_EBASE-5

  Mifare CRC error.
- #define DT_MIFARE_EFIFO DT_MIFARE_EBASE-6

  Mifare FIFO overflow.
- #define DT_MIFARE_EEEPROM DT_MIFARE_EBASE-7

  Mifare EEPROM error.
- #define DT_MIFARE_EKEY DT_MIFARE_EBASE-8

  Mifare invalid key.
- #define DT_MIFARE_EGENERIC DT_MIFARE_EBASE-9

  Mifare generic error.
- #define DT_MIFARE_EAUTHENTICATION DT_MIFARE_EBASE-10

  Mifare authentication error.
- #define DT_MIFARE_ECODE DT_MIFARE_EBASE-11

  Mifare code error.
- #define DT_MIFARE_EBIT DT_MIFARE_EBASE-12

  Mifare bit count error.
- #define DT_MIFARE_EACCESS DT_MIFARE_EBASE-13

  Mifare access error.
- #define DT_MIFARE_EVALUE DT_MIFARE_EBASE-14

  Mifare value error.
- #define DT_EMSR_EBASE -11000

  EMS base value.
- #define DT_EMSR_EINVALID_COMMAND DT_EMSR_EBASE-0x01

  Encrypted magnetic head invalid command sent.
- #define DT_EMSR_ENO_PERMISSION DT_EMSR_EBASE-0x02

  Encrypted magnetic head no permission error.
- #define DT_EMSR_ECARD DT_EMSR_EBASE-0x03

  Encrypted magnetic head card error.
- #define DT_EMSR_ESYNTAX DT_EMSR_EBASE-0x04

  Encrypted magnetic head command syntax error.
- #define DT_EMSR_ENO_RESPONSE DT_EMSR_EBASE-0x05

  Encrypted magnetic head command no response from the magnetic chip.
- #define DT_EMSR_ENO_DATA DT_EMSR_EBASE-0x06

  Encrypted magnetic head no data available.
- #define DT_EMSR_EINVALID_LENGTH DT_EMSR_EBASE-0x14

  Encrypted magnetic head invalid data length.

- #define DT_EMSR_ETAMPERED DT_EMSR_EBASE-0x15

  Encrypted magnetic head is tampered.
- #define DT_EMSR_EINVALID_SIGNATURE DT_EMSR_EBASE-0x16

  Encrypted magnetic head invalid signature.
- #define DT_EMSR_EHARDWARE DT_EMSR_EBASE-0x17

  Encrypted magnetic head hardware failure.
- #define DT_PPAD_EBASE -16500

  Pinpad base value.
- #define DT_PPAD_EGENERAL DT_PPAD_EBASE-1

  Generic error.
- #define DT_PPAD_EINVALID_COMMAND DT_PPAD_EBASE-2

  Invalid command or subcommand code.
- #define DT_PPAD_EINVALID_PARAMETER DT_PPAD_EBASE-3

  Invalid paremeter.
- #define DT_PPAD_EINVALID_ADDRESS DT_PPAD_EBASE-4

  Address is outside limits.
- #define DT_PPAD_EINVALID_VALUE DT_PPAD_EBASE-5

  Value is outside limits.
- #define DT_PPAD_EINVALID_LENGTH DT_PPAD_EBASE-6

  Length is outside limits.
- #define DT_PPAD_ENO_PERMISSION DT_PPAD_EBASE-7

  The action is not permitted in current state.
- #define DT_PPAD_ENO_DATA DT_PPAD_EBASE-8

  There is no data to be returned.
- #define DT_PPAD_ETIMEOUT DT_PPAD_EBASE-9

  Timeout occured.
- #define DT_PPAD_EINVALID_KEY_NUMBER DT_PPAD_EBASE-10

  Invalid key number.
- #define DT_PPAD_EINVALID_KEY_ATTRIBUTES DT_PPAD_EBASE-11

  Invalid key attributes (usage)
- #define DT_PPAD_EINVALID_DEVICE DT_PPAD_EBASE-12

  Calling of non-existing device.
- #define DT_PPAD_ENOT_SUPPORTED DT_PPAD_EBASE-13

  (not used in this FW version)
- #define DT_PPAD_EPIN_LIMIT_EXCEEDED DT_PPAD_EBASE-14

  Pin entering limit exceed.
- #define DT_PPAD_EFLASH DT_PPAD_EBASE-15

  Error in flash commands.
- #define DT_PPAD_EHARDWARE DT_PPAD_EBASE-16

  Hardware error.
- #define DT_PPAD_EINVALID_CRC DT_PPAD_EBASE-17

  (not used in this FW version)
- #define DT_PPAD_ECANCELLED DT_PPAD_EBASE-18

  Operation cancelled.
- #define DT_PPAD_EINVALID_SIGNATURE DT_PPAD_EBASE-19

  Invalid signature.
- #define DT_PPAD_EINVALID_HEADER DT_PPAD_EBASE-20

  Invalid data in header.
- #define DT_PPAD_EINVALID_PASSWORD DT_PPAD_EBASE-21

  Incorrent password.
- #define DT_PPAD_EINVALID_KEY_FORMAT DT_PPAD_EBASE-22

Invalid key format.
- #define DT_PPAD_ESCR DT_PPAD_EBASE-23

  Error in smart card reader.
- #define DT_PPAD_EHAL DT_PPAD_EBASE-24

  Error code is returned from HAL functions.
- #define DT_PPAD_EINVALID_KEY DT_PPAD_EBASE-25

  Invalid key (or missing)
- #define DT_PPAD_EINVALID_PIN DT_PPAD_EBASE-26

  The PIN length is <4 or >12.
- #define DT_PPAD_EINVALID_REMAINDER DT_PPAD_EBASE-27

  Issuer or ICC key invalid remainder length.
- #define DT_PPAD_ENOT_INITIALIZED DT_PPAD_EBASE-28

  (no used in this FW version)
- #define DT_PPAD_ELIMIT_REACHED DT_PPAD_EBASE-29

  (no used in this FW version)
- #define DT_PPAD_EINVALID_SEQUENCE DT_PPAD_EBASE-30

  (no used in this FW version)
- #define DT_PPAD_ENOT_PERMITTED DT_PPAD_EBASE-31

  The action is not permited.
- #define DT_PPAD_ENO_TMK DT_PPAD_EBASE-32

  TMK is not loaded.
- #define DT_PPAD_EWRONG_KEY DT_PPAD_EBASE-33

  Wrong key format.
- #define DT_PPAD_EDUPLICATE_KEY DT_PPAD_EBASE-34

  Duplicated key.
- #define DT_PPAD_EKEYBOARD_GENERAL DT_PPAD_EBASE-35

  General keyboard error.
- #define DT_PPAD_EKEYBOARD_NOT_CALIBRATED DT_PPAD_EBASE-36

  Keyboard not calibrated.
- #define DT_PPAD_EKEYBOARD_FAILURE DT_PPAD_EBASE-37

  Keyboard failure.

2.2.1 Detailed Description

Library error codes returned in the NSError objects.

2.2.2 Macro Definition Documentation

2.2.2.1 #define DT_EBUSY -5

Device or resource busy.

2.2.2.2 #define DT_ECLOSE -4

Close error.

2.2.2.3 #define DT_ECRC -11

CRC error.

2.2.2.4 #define DT_ECREATE -2

Create error.

2.2.2.5 #define DT_EDEVICE -14

Device error.

2.2.2.6 #define DT_EEEPROM -13

EEPROM error.

2.2.2.7 #define DT_EFLASH -12

Flash error.

2.2.2.8 #define DT_EGENERAL -1

General error / Unknown error.

2.2.2.9 #define DT_EINVALID_CMD -17

Invalid command.

2.2.2.10 #define DT_EIO -10

Input/Output error.

2.2.2.11 #define DT_EMEMORY -8

Memory allocation error.

2.2.2.12 #define DT_EMSR_ECARD DT_EMSR_EBASE-0x03

Encrypted magnetic head card error.

2.2.2.13 #define DT_EMSR_EHARDWARE DT_EMSR_EBASE-0x17

Encrypted magnetic head hardware failure.

2.2.2.14 #define DT_EMSR_EINVALID_COMMAND DT_EMSR_EBASE-0x01

Encrypted magnetic head invalid command sent.

2.2.2.15 #define DT_EMSR_EINVALID_LENGTH DT_EMSR_EBASE-0x14

Encrypted magnetic head invalid data length.

2.2.2.16 #define DT_EMSR_EINVALID_SIGNATURE DT_EMSR_EBASE-0x16

Encrypted magnetic head invalid signature.

2.2.2.17 #define DT_EMSR_ENO_DATA DT_EMSR_EBASE-0x06

Encrypted magnetic head no data available.

2.2.2.18 #define DT_EMSR_ENO_PERMISSION DT_EMSR_EBASE-0x02

Encrypted magnetic head no permission error.

2.2.2.19 #define DT_EMSR_ENO_RESPONSE DT_EMSR_EBASE-0x05

Encrypted magnetic head command no response from the magnetic chip.

2.2.2.20 #define DT_EMSR_ESYNTAX DT_EMSR_EBASE-0x04

Encrypted magnetic head command syntax error.

2.2.2.21 #define DT_EMSR_ETAMPERED DT_EMSR_EBASE-0x15

Encrypted magnetic head is tampered.

2.2.2.22 #define DT_ENOEXIST -16

The device or resource does not exists.

2.2.2.23 #define DT_ENOIMPLEMENTED -15

The operation is not implemented.

2.2.2.24 #define DT_ENOMORE -19

No more items.

2.2.2.25 #define DT_ENONE 0

Operation successful.

2.2.2.26 #define DT_ENOSUPPORTED -7

Unsupported method or operation.

2.2.2.27 #define DT_ENOT_EXIST_OBJECT -18

Not exist object.

2.2.2.28 #define DT_EOPEN -3

Open error.

2.2.2.29 #define DT_EPARAM -9

Invalid parameter.

2.2.2.30 #define DT_ETIMEOUT -6

Timeout expired.

2.2.2.31 #define DT_MIFARE_EACCESS DT_MIFARE_EBASE-13

Mifare access error.

2.2.2.32 #define DT_MIFARE_EAUTHENTICATION DT_MIFARE_EBASE-10

Mifare authentication error.

2.2.2.33 #define DT_MIFARE_EBASE -10000

Mifare operation successful.

2.2.2.34 #define DT_MIFARE_EBIT DT_MIFARE_EBASE-12

Mifare bit count error.

2.2.2.35 #define DT_MIFARE_ECODE DT_MIFARE_EBASE-11

Mifare code error.

2.2.2.36 #define DT_MIFARE_ECOLLISION DT_MIFARE_EBASE-2

Mifare collision error.

2.2.2.37 #define DT_MIFARE_ECRC DT_MIFARE_EBASE-5

Mifare CRC error.

2.2.2.38 #define DT_MIFARE_EEEPROM DT_MIFARE_EBASE-7

Mifare EEPROM error.

2.2.2.39 #define DT_MIFARE_EFIFO DT_MIFARE_EBASE-6

Mifare FIFO overflow.

2.2.2.40 #define DT_MIFARE_EFRAME DT_MIFARE_EBASE-4

Mifare frame error.

2.2.2.41 #define DT_MIFARE_EGENERIC DT_MIFARE_EBASE-9

Mifare generic error.

2.2.2.42 #define DT_MIFARE_EKEY DT_MIFARE_EBASE-8

Mifare invalid key.

2.2.2.43 #define DT_MIFARE_EPARITY DT_MIFARE_EBASE-3

Mifare parity error.

2.2.2.44 #define DT_MIFARE_ETIMEOUT DT_MIFARE_EBASE-1

Mifare timeout error.

2.2.2.45 #define DT_MIFARE_EVALUE DT_MIFARE_EBASE-14

Mifare value error.

2.2.2.46 #define DT_PPAD_ENO_TMK DT_PPAD_EBASE-32

TMK is not loaded.

The action cannot be executed

2.3 Delegate Notifications

Notifications sent by the sdk on various events - barcode scanned, magnetic card data, communication status, etc.

Functions

- (void) - <DTDeviceDelegate>::connectionState:

    Notifies about the current connection state.
- (void) - <DTDeviceDelegate>::deviceButtonPressed:

    Notification sent when some of the device's buttons is pressed.
- (void) - <DTDeviceDelegate>::deviceButtonReleased:

    Notification sent when some of the device's buttons is released.
- (void) - <DTDeviceDelegate>::barcodeData:type:

    Notification sent when barcode is successfuly read.
- (void) - <DTDeviceDelegate>::barcodeData:isotype:

    Notification sent when barcode is successfuly read.
- (void) - <DTDeviceDelegate>::barcodeNSData:type:

    Notification sent when barcode is successfuly read.
- (void) - <DTDeviceDelegate>::barcodeNSData:isotype:

    Notification sent when barcode is successfuly read.
- (void) - <DTDeviceDelegate>::magneticCardData:track2:track3:

    Notification sent when magnetic card is successfuly read.
- (void) - <DTDeviceDelegate>::magneticCardEncryptedData:tracks:data:

    Notification sent when magnetic card is successfuly read.
- (void) - <DTDeviceDelegate>::magneticCardEncryptedData:tracks:data:track1masked:track2masked-:track3:

    Notification sent when magnetic card is successfuly read.
- (void) - <DTDeviceDelegate>::magneticCardEncryptedData:tracks:data:track1masked:track2masked-:track3:source:

    Notification sent when magnetic card is successfuly read.
- (void) - <DTDeviceDelegate>::magneticCardRawData:

    Notification sent when magnetic card is successfuly read.
- (void) - <DTDeviceDelegate>::magneticCardEncryptedRawData:data:

    Notification sent when magnetic card is successfuly read.
- (void) - <DTDeviceDelegate>::firmwareUpdateProgress:percent:

    Notification sent when firmware update process advances.
- (void) - <DTDeviceDelegate>::bluetoothDiscoverComplete:

    Notification sent when bluetooth discovery finds new bluetooth device.
- (void) - <DTDeviceDelegate>::bluetoothDeviceDiscovered:name:

    Notification sent when bluetooth discovery finds new bluetooth device.
- (void) - <DTDeviceDelegate>::bluetoothDeviceConnected:

    Notification sent when bluetooth device is connected.
- (void) - <DTDeviceDelegate>::bluetoothDeviceDisconnected:

    Notification sent when bluetooth connection is lost.
- (BOOL) - <DTDeviceDelegate>::bluetoothDeviceRequestedConnection:name:

    Notification sent when a bluetooth device requests.
- (NSString ∗) - <DTDeviceDelegate>::bluetoothDevicePINCodeRequired:name:

    Notification sent when a bluetooth device requests.
- (void) - <DTDeviceDelegate>::magneticJISCardData:

    Notification sent when JIS I & II magnetic card is successfuly read.
- (void) - <DTDeviceDelegate>::rfCardDetected:info:

Notification sent when a new supported RFID card enters the field.

- (void) - <DTDeviceDelegate>::rfCardRemoved:

  Notification sent when the card leaves the field.
- (void) - <DTDeviceDelegate>::deviceFeatureSupported:value:

  Notification sent when some of the features gets enabled or disabled.
- (void) - <DTDeviceDelegate>::smartCardInserted:

  Notification sent when smartcard was inserted.
- (void) - <DTDeviceDelegate>::smartCardRemoved:

  Notification sent when smartcard was removed.
- (void) - <DTDeviceDelegate>::PINEntryCompleteWithError:

  Notification sent when PIN entry procedure have completed or was cancelled.
- (void) - <DTDeviceDelegate>::paperStatus:

  Notification sent when printer's paper sensor changes.
- (void) - <DTDeviceDelegate>::sdkDebug:source:

  Notification sent to display debug messages from the sdk or device.
- (void) - <DTDeviceDelegate>::emv2OnTransactionStarted

  Notification sent when EMV kernel detects a card and start processing it.
- (void) - <DTDeviceDelegate>::emv2OnUserInterfaceCode:status:holdTime:

  Notification sent when the EMV kernel wants to update the user interface.
- (void) - <DTDeviceDelegate>::emv2OnApplicationSelection:

  Notification sent when the card has multiple applications and one needs to be selected.
- (void) - <DTDeviceDelegate>::emv2OnOnlineProcessing:

  Notification sent when the kernel and the card require online processing.
- (void) - <DTDeviceDelegate>::emv2OnTransactionFinished:

  Notification sent when the transaction is complete.

2.3.1 Detailed Description

Notifications sent by the sdk on various events - barcode scanned, magnetic card data, communication status, etc.

2.3.2 Function Documentation

2.3.2.1 - (void) barcodeData: (NSString ∗) *barcode* isotype:(NSString ∗) *isotype*

Notification sent when barcode is successfuly read.

This notification is used when barcode type is set to BARCODE_TYPE_ISO15424

Parameters

barcode	- string containing barcode data
type	- barcode type, according to ISO 15424

2.3.2.2 - (void) barcodeData: (NSString ∗) *barcode* type:(int) *type*

Notification sent when barcode is successfuly read.

This notification is used when barcode type is set to BARCODE_TYPE_DEFAULT or BARCODE_TYPE_EXTEN-DED.

Parameters

barcode	- string containing barcode data
type	- barcode type, one of the BAR_∗ constants

2.3.2.3 - (void) barcodeNSData: (NSData ∗) *barcode* isotype:(NSString ∗) *isotype*

Notification sent when barcode is successfuly read.

This notification is used when barcode type is set to BARCODE_TYPE_ISO15424

Parameters

barcode	- string containing barcode data
type	- barcode type, according to ISO 15424

2.3.2.4 - (void) barcodeNSData: (NSData ∗) *barcode* type:(int) *type*

Notification sent when barcode is successfuly read.

This notification is used when barcode type is set to BARCODE_TYPE_DEFAULT or BARCODE_TYPE_EXTEN-DED.

Parameters

barcode	- NSData containing barcode data
type	- barcode type, one of the BAR_∗ constants

2.3.2.5 - (void) bluetoothDeviceConnected: (NSString ∗) *address*

Notification sent when bluetooth device is connected.

Parameters

address	bluetooth address of the device
name	bluetooth name of the device

2.3.2.6 - (void) bluetoothDeviceDisconnected: (NSString ∗) *address*

Notification sent when bluetooth connection is lost.

Parameters

address	bluetooth address of the device

2.3.2.7 - (void) bluetoothDeviceDiscovered: (NSString ∗) *address* name:(NSString ∗) *name*

Notification sent when bluetooth discovery finds new bluetooth device.

Parameters

address	bluetooth address of the device
name	bluetooth name of the device

2.3.2.8 - (NSString ∗) bluetoothDevicePINCodeRequired: (NSString ∗) *address* name:(NSString ∗) *name*

Notification sent when a bluetooth device requests.

Parameters

address	bluetooth address of the device
name	bluetooth name of the device

2.3.2.9 - (BOOL) bluetoothDeviceRequestedConnection: (NSString ∗) *address* name:(NSString ∗) *name*

Notification sent when a bluetooth device requests.

Parameters

address	bluetooth address of the device
name	bluetooth name of the device

2.3.2.10 - (void) bluetoothDiscoverComplete: (BOOL) *success*

Notification sent when bluetooth discovery finds new bluetooth device.

Parameters

success	true if the discovery complete successfully, even if it not resulted in any device found, false if there was an error communicating with the bluetooth module

2.3.2.11 - (void) connectionState: (int) *state*

Notifies about the current connection state.

Parameters

state	- connection state, one of:	
	CONN_DISCONNECTED	there is no connection to any device and the sdk will not try to make one even if the device is attached
	CONN_CONNECTING	no device is currently connected, but the sdk is actively trying to
	CONN_CONNECTED	One or more devices are connected

2.3.2.12 - (void) deviceButtonPressed: (int) *which*

Notification sent when some of the device's buttons is pressed.

Parameters

which	button identifier, one of:	
	0	right scan button

2.3.2.13 - (void) deviceButtonReleased: (int) *which*

Notification sent when some of the device's buttons is released.

Parameters

which	button identifier, one of: <table><tr><td>0</td><td>right scan button</td></tr></table>

2.3.2.14 - (void) deviceFeatureSupported: (int) *feature* value:(int) *value*

Notification sent when some of the features gets enabled or disabled.

Parameters

feature	feature type, one of the FEAT_∗ constants
value	FEAT_UNSUPPORTED if the feature is not supported on the connected device(s), FEAT_S-UPPORTED or one of the specific constants for each feature otherwise

2.3.2.15 - (void) emv2OnApplicationSelection: (NSArray ∗) *applications*

Notification sent when the card has multiple applications and one needs to be selected.

This can only happen with smart cards, NFC cards automatically select the application.

Parameters

applications	an array of strings with application names, when ready call emv2SelectApplication with the correct application index

2.3.2.16 - (void) emv2OnOnlineProcessing: (NSData ∗) *data*

Notification sent when the kernel and the card require online processing.

Data consists of tags needed for online processing and should be processed by the financial institution. Call emv2-SetOnlineResult when done with the online connection to notify the kernel of the result

Parameters

data	TLV list

2.3.2.17 - (void) emv2OnTransactionFinished: (NSData ∗) *data*

Notification sent when the transaction is complete.

Data consists of all the tags available, including plain text ones for display purposes and encrypted for sending over to the backend

Parameters

data	TLV list

2.3.2.18 - (void) emv2OnUserInterfaceCode: (int) *code* status:(int) *status* holdTime:(NSTimeInterval) *holdTime*

Notification sent when the EMV kernel wants to update the user interface.

Parameters

code	user interface code, one of the EMV_UI_∗ constants
status	user interface status or -1 if status is unavailable
holdTime	the time to display the message or -1 if time is unavailable

2.3.2.19 - (void) firmwareUpdateProgress: (int) *phase* percent:(int) *percent*

Notification sent when firmware update process advances.

Do not call any other functions until firmware update is complete! During the firmware update notifications will be posted.

Parameters

phase	update phase, one of:	
	UPDATE_INIT	Initializing firmware update
	UPDATE_ERASE	Erasing flash memory
	UPDATE_WRITE	Writing data
	UPDATE_FINISH	Update complete
percent	firmware update progress in percents	

2.3.2.20 - (void) magneticCardData: (NSString ∗) *track1* track2:(NSString ∗) *track2* track3:(NSString ∗) *track3*

Notification sent when magnetic card is successfuly read.

Parameters

track1	- data contained in track 1 of the magnetic card or nil
track2	- data contained in track 2 of the magnetic card or nil
track3	- data contained in track 3 of the magnetic card or nil

2.3.2.21 - (void) magneticCardEncryptedData: (int) *encryption* tracks:(int) *tracks* data:(NSData ∗) *data*

Notification sent when magnetic card is successfuly read.

The data is being sent encrypted.

Parameters

encryption	encryption algorithm used, one of ALG_∗ constants

For AES256, after decryption, the result data will be as follows:

- Random data (4 bytes)
- Device identification text (16 ASCII characters, unused bytes are 0)
- Processed track data in the format: 0xF1 (track1 data), 0xF2 (track2 data) 0xF3 (track3 data). It is possible some of the tracks will be empty, then the identifier will not be present too, for example 0xF1 (track1 data) 0xF3 (track3 data)
- End of track data (byte 0x00)

- CRC16 (2 bytes) - the CRC is performed from the start of the encrypted block (the Random Data block) to the end of the track data (including the 0x00 byte). The data block is rounded to 16 bytes

In the more secure way, where the decryption key resides in a server only, the card read process will look something like:

- (User) swipes the card
- (iOS program) receives the data via magneticCardEncryptedData and sends to the server
- (iOS program)[optional] sends current device serial number along with the data received from magneticCard-EncryptedData. This can be used for data origin verification
- (Server) decrypts the data, extracts all the information from the fields
- (Server)[optional] if the ipod program have sent the device serial number before, the server compares the received serial number with the one that's inside the encrypted block
- (Server) checks if the card data is the correct one, i.e. all needed tracks are present, card is the same type as required, etc and sends back notification to the ipod program.

For IDTECH with DUKPT the data contains:

- DATA[0]: CARD TYPE: 0 - payment card
- DATA[1]: TRACK FLAGS
- DATA[2]: TRACK 1 LENGTH
- DATA[3]: TRACK 2 LENGTH
- DATA[4]: TRACK 3 LENGTH
- DATA[??]: TRACK 1 DATA MASKED
- DATA[??]: TRACK 2 DATA MASKED
- DATA[??]: TRACK 3 DATA
- DATA[??]: TRACK 1 AND TRACK 2 TDES ENCRYPTED
- DATA[??]: TRACK 1 SHA1 (0x14 BYTES)
- DATA[??]: TRACK 2 SHA1 (0x14 BYTES)
- DATA[??]: DUKPT SERIAL AND COUNTER (0x0A BYTES)

Parameters

tracks	contain information which tracks are successfully read and inside the encrypted data as bit fields, bit 1 corresponds to track 1, etc, so value of 7 means all tracks are read
data	contains the encrypted card data

2.3.2.22 - (void) magneticCardEncryptedData: (int) *encryption* tracks:(int) *tracks* data:(NSData ∗) *data* track1masked:(NSString ∗) *track1masked* track2masked:(NSString ∗) *track2masked* track3:(NSString ∗) *track3*

Notification sent when magnetic card is successfuly read.

The data is being sent encrypted.

Parameters

encryption	encryption algorithm used, one of:	
	0	AES 256
	1	IDTECH with DUKPT

For AES256, after decryption, the result data will be as follows:

- Random data (4 bytes)
- Device identification text (16 ASCII characters, unused bytes are 0)
- Processed track data in the format: 0xF1 (track1 data), 0xF2 (track2 data) 0xF3 (track3 data). It is possible some of the tracks will be empty, then the identifier will not be present too, for example 0xF1 (track1 data) 0xF3 (track3 data)
- End of track data (byte 0x00)
- CRC16 (2 bytes) - the CRC is performed from the start of the encrypted block (the Random Data block) to the end of the track data (including the 0x00 byte). The data block is rounded to 16 bytes

In the more secure way, where the decryption key resides in a server only, the card read process will look something like:

- (User) swipes the card
- (iOS program) receives the data via magneticCardEncryptedData and sends to the server
- (iOS program)[optional] sends current Linea serial number along with the data received from magneticCard-EncryptedData. This can be used for data origin verification
- (Server) decrypts the data, extracts all the information from the fields
- (Server)[optional] if the ipod program have sent the Linea serial number before, the server compares the received serial number with the one that's inside the encrypted block
- (Server) checks if the card data is the correct one, i.e. all needed tracks are present, card is the same type as required, etc and sends back notification to the ipod program.

For IDTECH with DUKPT the data contains:

- DATA[0]: CARD TYPE: 0 - payment card
- DATA[1]: TRACK FLAGS
- DATA[2]: TRACK 1 LENGTH
- DATA[3]: TRACK 2 LENGTH
- DATA[4]: TRACK 3 LENGTH
- DATA[??]: TRACK 1 DATA MASKED
- DATA[??]: TRACK 2 DATA MASKED
- DATA[??]: TRACK 3 DATA
- DATA[??]: TRACK 1 AND TRACK 2 TDES ENCRYPTED
- DATA[??]: TRACK 1 SHA1 (0x14 BYTES)
- DATA[??]: TRACK 2 SHA1 (0x14 BYTES)
- DATA[??]: DUKPT SERIAL AND COUNTER (0x0A BYTES)

Parameters

tracks	contain information which tracks are successfully read and inside the encrypted data as bit fields, bit 1 corresponds to track 1, etc, so value of 7 means all tracks are read
data	contains the encrypted card data
track1masked	when possible, track1 data will be masked and returned here
track2masked	when possible, track2 data will be masked and returned here

2.3.2.23 - (void) magneticCardEncryptedData: (int) *encryption* tracks:(int) *tracks* data:(NSData $*$) *data* track1masked:(NSString $*$) *track1masked* track2masked:(NSString $*$) *track2masked* track3:(NSString $*$) *track3* source:(int) *source*

Notification sent when magnetic card is successfuly read.

The data is being sent encrypted.

Parameters

encryption	encryption algorithm used, one of:	
	0	AES 256
	1	IDTECH with DUKPT

For AES256, after decryption, the result data will be as follows:

- Random data (4 bytes)
- Device identification text (16 ASCII characters, unused bytes are 0)
- Processed track data in the format: 0xF1 (track1 data), 0xF2 (track2 data) 0xF3 (track3 data). It is possible some of the tracks will be empty, then the identifier will not be present too, for example 0xF1 (track1 data) 0xF3 (track3 data)
- End of track data (byte 0x00)
- CRC16 (2 bytes) - the CRC is performed from the start of the encrypted block (the Random Data block) to the end of the track data (including the 0x00 byte). The data block is rounded to 16 bytes

In the more secure way, where the decryption key resides in a server only, the card read process will look something like:

- (User) swipes the card
- (iOS program) receives the data via magneticCardEncryptedData and sends to the server
- (iOS program)[optional] sends current Linea serial number along with the data received from magneticCard-EncryptedData. This can be used for data origin verification
- (Server) decrypts the data, extracts all the information from the fields
- (Server)[optional] if the ipod program have sent the Linea serial number before, the server compares the received serial number with the one that's inside the encrypted block
- (Server) checks if the card data is the correct one, i.e. all needed tracks are present, card is the same type as required, etc and sends back notification to the ipod program.

For IDTECH with DUKPT the data contains:

- DATA[0]: CARD TYPE: 0 - payment card
- DATA[1]: TRACK FLAGS
- DATA[2]: TRACK 1 LENGTH

- DATA[3]: TRACK 2 LENGTH
- DATA[4]: TRACK 3 LENGTH
- DATA[??]: TRACK 1 DATA MASKED
- DATA[??]: TRACK 2 DATA MASKED
- DATA[??]: TRACK 3 DATA
- DATA[??]: TRACK 1 AND TRACK 2 TDES ENCRYPTED
- DATA[??]: TRACK 1 SHA1 (0x14 BYTES)
- DATA[??]: TRACK 2 SHA1 (0x14 BYTES)
- DATA[??]: DUKPT SERIAL AND COUNTER (0x0A BYTES)

Parameters

tracks	contain information which tracks are successfully read and inside the encrypted data as bit fields, bit 1 corresponds to track 1, etc, so value of 7 means all tracks are read
data	contains the encrypted card data
track1masked	when possible, track1 data will be masked and returned here
track2masked	when possible, track2 data will be masked and returned here
source	the track data source, one of the CARD_∗ constants

2.3.2.24 - (void) magneticCardEncryptedRawData: (int) *encryption* data:(NSData ∗) *data*

Notification sent when magnetic card is successfuly read.

The raw card data is encrypted via the selected encryption algorithm. After decryption, the result data will be as follows:

- Random data (4 bytes)
- Device identification text (16 ASCII characters, unused bytes are 0)
- Track data: the maximum length of a single track is 704 bits (88 bytes), so track data contains 3x88 bytes
- CRC16 (2 bytes) - the CRC is performed from the start of the encrypted block (the Random Data block) to the end of the track data. The data block is rounded to 16 bytes

Parameters

encryption	encryption algorithm used, one of ALG_∗ constants
data	- Contains the encrypted raw card data

2.3.2.25 - (void) magneticCardRawData: (NSData ∗) *tracks*

Notification sent when magnetic card is successfuly read.

Parameters

tracks	contains the raw magnetic card data. These are the bits directly from the magnetic head. The maximum length of a single track is 704 bits (88 bytes), so the command returns the 3 tracks as 3x88 bytes block

2.3.2.26 - (void) magneticJISCardData: (NSString ∗) *data*

Notification sent when JIS I & II magnetic card is successfuly read.

Parameters

data	- data contained in the magnetic card

2.3.2.27 - (void) paperStatus: (BOOL) *present*

Notification sent when printer's paper sensor changes.

Parameters

present	TRUE if paper is present, FALSE if printer is out of paper or cover is open

2.3.2.28 - (void) PINEntryCompleteWithError: (NSError ∗) *error*

Notification sent when PIN entry procedure have completed or was cancelled.

Parameters

error	nil if no error occured, or NSError object if the generation failed

2.3.2.29 - (void) rfCardDetected: (int) *cardIndex* info:(DTRFCardInfo ∗) *info*

Notification sent when a new supported RFID card enters the field.

Parameters

cardIndex	the index of the card, use this index with all subsequent commands to the card
info	information about the card

2.3.2.30 - (void) rfCardRemoved: (int) *cardIndex*

Notification sent when the card leaves the field.

Parameters

cardIndex	the index of the card, use this index with all subsequent commands to the card

2.3.2.31 - (void) sdkDebug: (NSString ∗) *logText* source:(int) *source*

Notification sent to display debug messages from the sdk or device.

Parameters

logText	debug message
source	source device type, 0 means the connected device, 1 is the sdk

2.3.2.32 - (void) smartCardInserted: (SC_SLOTS) *slot*

Notification sent when smartcard was inserted.

Parameters

slot	smart card slot number

2.3.2.33 - (void) smartCardRemoved: (SC_SLOTS) *slot*

Notification sent when smartcard was removed.

Parameters

slot	smart card slot number

2.4 General functions

Functions to connect/disconnect, set delegate, make sounds, update firmware, control various device settings.

Functions

- (id) + DTDevices::sharedDevice

  Creates and initializes new class instance or returns already initalized one.
- (void) - DTDevices::addDelegate:

  Allows unlimited delegates to be added to a single class instance.
- (void) - DTDevices::removeDelegate:

  Removes delegate, previously added with addDelegate.
- (void) - DTDevices::connect

  Tries to connect to supported devices in the background, connection status notifications will be passed through the delegate.
- (void) - DTDevices::disconnect

  Stops the sdk from trying to connect to supported devices and breaks existing connections.
- (BOOL) - **DTDevices::isPresent:**
- (BOOL) - DTDevices::setActiveDeviceType:error:

  The sdk can work with many devices at the same time, but some functions can be executed on a single device at a time (for example barcodeStartScan), this function sets the prefered device to execute the function by type.
- (BOOL) - DTDevices::setAutoOffWhenIdle:whenDisconnected:error:

  Sets the time in seconds, after which Linea will shut down to conserve battery.
- (BOOL) - DTDevices::getBatteryCapacity:voltage:error:

  Returns active device's battery capacity.
- (BOOL) - DTDevices::playSound:beepData:length:error:

  Plays a sound using the built-in speaker on the active device.
- (BOOL) - DTDevices::getCharging:error:

  Returns if the connected device is charging the iOS device from it's own battery.
- (BOOL) - DTDevices::setCharging:error:

  Enables or disables Lines's capability to charge the handheld from it's own battery.
- (BOOL) - DTDevices::getPassThroughSync:error:

  Returns the current state of the pass-through synchronization.
- (BOOL) - DTDevices::setPassThroughSync:error:

  Enables or disables pass-through synchronization when you plug usb cable.
- (BOOL) - DTDevices::setUSBChargeCurrent:error:

  Sets the charge current that lightning connector based Lineas will allow the iPod/iPhone/iPad to be charged with when connected via USB port.
- (NSDictionary ∗) - DTDevices::getFirmwareFileInformation:error:

  Returns information about the specified firmware data.
- (BOOL) - DTDevices::updateFirmwareData:error:

  Updates connected device's firmware with specified firmware data.
- (int) - DTDevices::getSupportedFeature:error:

  Returns if a feature is supported on connected device(s) and what type it is.

2.4.1 Detailed Description

Functions to connect/disconnect, set delegate, make sounds, update firmware, control various device settings.

2.4.2 Function Documentation

2.4.2.1 - (void) addDelegate: (id) *newDelegate*

Allows unlimited delegates to be added to a single class instance.

This is useful in the case of global class and every view can use addDelegate when the view is shown and remove-Delegate when no longer needs to monitor events

Parameters

newDelegate	the delegate that will be notified of events

2.4.2.2 - (void) connect

Tries to connect to supported devices in the background, connection status notifications will be passed through the delegate.

Once connect is called, it will automatically try to reconnect until disconnect is called. Note that "connect" call works in background and will notify the caller of connection success via connectionState delegate. Do not assume the library has fully connected to the device after this call, but wait for the notification.

2.4.2.3 - (BOOL) getBatteryCapacity: (int ∗) *capacity* voltage:(float ∗) *voltage* error:(NSError ∗∗) *error*

Returns active device's battery capacity.

Note

Reading battery voltages during charging is unreliable!

Parameters

capacity	returns battery capacity in percents, ranging from 0 when battery is dead to 100 when fully charged. Pass nil if you don't want that information
voltage	returns battery voltage in Volts, pass nil if you don't want that information
error	pointer to NSError object, where error information is stored in case function fails. You can pass nil if you don't want that information

Returns

TRUE if function succeeded, FALSE otherwise

2.4.2.4 - (BOOL) getCharging: (BOOL ∗) *charging* error:(NSError ∗∗) *error*

Returns if the connected device is charging the iOS device from it's own battery.

Linea firmware versions prior to 2.13 will return true if external charge is attached, 2.13 and later will return only if Linea's own battery is used for charging.

Parameters

charging	returns TRUE if charging is enabled (from internal battery, external charging is omitted)
error	pointer to NSError object, where error information is stored in case function fails. You can pass nil if you don't want that information

Returns

TRUE if function succeeded, FALSE otherwise

2.4.2.5 - (NSDictionary ∗) getFirmwareFileInformation: (NSData ∗) *data* error:(NSError ∗∗) *error*

Returns information about the specified firmware data.

Based on it, and the connected device's name, model and firmware version you can chose to update or not the firmware

Parameters

data	- firmware data		
	"deviceName"	Device name, for example "Linea"	
	"deviceModel"	Device model, for example "XBAMBL</td></tr><tr><td>"firmware-Revision"</td><td>-Firmware revision as string, for example 2.41</td></tr><tr><td>"firmware-RevisionNumber"	Firmware revision as number MAJOR∗100+MINOR, i.e. 2.41 will be returned as 241
error	pointer to NSError object, where error information is stored in case function fails. You can pass nil if you don't want that information		

Returns

firmware information if function succeeded, nil otherwise

2.4.2.6 - (BOOL) getPassThroughSync: (BOOL ∗) *enabled* error:(NSError ∗∗) *error*

Returns the current state of the pass-through synchronization.

Parameters

enabled	returns if the sync is enabled or disabled
error	pointer to NSError object, where error information is stored in case function fails. You can pass nil if you don't want that information

Returns

TRUE if function succeeded, FALSE otherwise

2.4.2.7 - (int) getSupportedFeature: (int) *feature* error:(NSError ∗∗) *error*

Returns if a feature is supported on connected device(s) and what type it is.

Parameters

feature	one of the FEAT_∗ constants
error	pointer to NSError object, where error information is stored in case function fails. You can pass nil if you don't want that information

Returns

FEAT_UNSUPPORTED if feature is not supported, FEAT_SUPPORTED or one or more feature specific types otherwise

2.4.2.8 - (BOOL) playSound: (int) *volume* beepData:(int ∗) *data* length:(int) *length* error:(NSError ∗∗) *error*

Plays a sound using the built-in speaker on the active device.

Note

A sample beep containing of 2 tones, each with 400ms duration, first one 2000Hz and second - 5000Hz will look int beepData[]={2000,400,5000,400}

Parameters

volume	controls the volume (0-100). Currently have no effect
data	an array of integer values specifying pairs of tone(Hz) and duration(ms).
length	length in bytes of beepData array
error	pointer to NSError object, where error information is stored in case function fails. You can pass nil if you don't want that information

Returns

TRUE if function succeeded, FALSE otherwise

2.4.2.9 - (void) removeDelegate: (id) *newDelegate*

Removes delegate, previously added with addDelegate.

Parameters

newDelegate	the delegate that will be no longer be notified of events

2.4.2.10 - (BOOL) setActiveDeviceType: (int) *type* error:(NSError ∗∗) *error*

The sdk can work with many devices at the same time, but some functions can be executed on a single device at a time (for example barcodeStartScan), this function sets the prefered device to execute the function by type.

Parameters

type	device type to be made active, one of the DEVICE_TYPE_∗ constants
error	pointer to NSError object, where error information is stored in case function fails. You can pass nil if you don't want that information

Returns

TRUE if function succeeded, FALSE otherwise

2.4.2.11 - (BOOL) setAutoOffWhenIdle: (NSTimeInterval) *timeIdle* whenDisconnected:(NSTimeInterval) *timeDisconnected* error:(NSError ∗∗) *error*

Sets the time in seconds, after which Linea will shut down to conserve battery.

This works with lightning connector Lineas only (LP5, LPTab4, LPTabMini)

Note

When Linea is being used by a program, only the idle time is taken in effect, but when Linea is disconnected BOTH parameters have effect - if idle time is 10 seconds and disconnected time is 30, then Linea will awlays disconnect in 10 seconds of inactivity! Thus idle time should always be bigger than disconnected time!

Parameters

timeIdle	this is the idle time, connected or not, after which Linea will turn off. The default value is 5400 seconds (90 minutes)
time-Disconnected	this is the time with no active program connection, after which Linea will turn off. The default value is 30 seconds
error	pointer to NSError object, where error information is stored in case function fails. You can pass nil if you don't want that information

Returns

TRUE if function succeeded, FALSE otherwise

2.4.2.12 - (BOOL) setCharging: (BOOL) *enabled* error:(NSError $**$) *error*

Enables or disables Lines's capability to charge the handheld from it's own battery.

Charging can stop if connected device's battery goes too low.

While Linea can act as external battery for the iPod/iPhone, there are certain limitations if you decide to implement it. The internal battery is not big enough, so if the iPod/iPhone consumes a lot of power from it, it will go down very fast and force the firmware to cut the charge to prevent going down to dangerous levels. The proper use of this charging function depends on how the program, running on the iPod/iPhone, is used and how the iPod/iPhone is discharged

There are two possible ways to use Linea's charge:

- Emergency mode - in the case iPod/iPhone usage is designed in a way it will last long enough between charging sessions and using Linea's charge is not generally needed, the charge can be used if the iPod/i-Phone for some reason goes too low (like $<$50%), so it is given some power to continue working until next charging. An example will be store, where devices are being charged every night, but extreme usage on some iPod drains the battery before the end of the shift. This is the less efficient way to charge it, also, Linea will refuse to start the charge if it's own battery goes below 3.8v, so depending on the usage, barcode type and if the barcode engine is set to work all the time, it may not be possible to start the charge.

- Max life mode - it is the case where both devices are required to operate as long as possible. Usually, the iPod/iPhone's battery will be drained way faster than Linea's, especially with wifi enabled programs and to keep both devices operating as long as possible, the charging should be desinged in a way so iPod/iPhone is able to use most of Linea's battery. This is possible, if you start charging when iPod/iPhone is almost full - at around 75-80% or higher. This way the iPod will consume small amount of energy, allowing our battery to slowly be used almost fully to charge it.

LibraryDemo application contains sample implementation of max life mode charging.

Note

Reading battery voltages during charging is unreliable!
Enabling charge can fail if connected device's battery is low. Disabling charge will fail if there is external charger or usb cable attached.

Parameters

enabled	TRUE to enable charging, FALSE to disable/stop it
error	pointer to NSError object, where error information is stored in case function fails. You can pass nil if you don't want that information

Returns

TRUE if function succeeded, FALSE otherwise

2.4.2.13 - (BOOL) setPassThroughSync: (BOOL) *enabled* error:(NSError $**$) *error*

Enables or disables pass-through synchronization when you plug usb cable.

In lightning connector devices this is important, as you can no longer have both sync and communication at the same time. Disable the sync for stationary, always on charge systems. Sync mode is persistent, but there is no downside of setting the desired one upon connection.

Parameters

enabled	TRUE to enable pass-through sync, FALSE to disable it
error	pointer to NSError object, where error information is stored in case function fails. You can pass nil if you don't want that information

Returns

TRUE if function succeeded, FALSE otherwise

2.4.2.14 - (BOOL) setUSBChargeCurrent: (int) *current* error:(NSError $**$) *error*

Sets the charge current that lightning connector based Lineas will allow the iPod/iPhone/iPad to be charged with when connected via USB port.

This setting persists.

Note

Note the combined consumption on both Linea (max 300mA) and the iPod/iPhone/iPad, some USB ports may not be strong enough and will turn off. Usually an usb port provides up to 1A, so setting the iOS charge to 500mA is always safe, but high powered usb ports can provide much more.

Warning

You can damage your adapter/port if you increase the charge current beyound its limits!!! Do not put 1A charge on 1A adapters, always use 2A adapter! Do not use 1A charge on PCs, unless it goes through high-power usb HUB!

Parameters

current	the charge current in mA (normally it is 500). Currently linea accepts only 500 and 1000 as parameters
error	pointer to NSError object, where error information is stored in case function fails. You can pass nil if you don't want that information

Returns

TRUE if function succeeded, FALSE otherwise

2.4.2.15 + (id) sharedDevice

Creates and initializes new class instance or returns already initalized one.

Use this function, if you want to access the class from different places

Returns

shared class instance

2.4.2.16 - (BOOL) updateFirmwareData: (NSData $*$) *data* error:(NSError $**$) *error*

Updates connected device's firmware with specified firmware data.

The firmware can only be upgraded or downgraded, if you send the same firmware version, then no update process will be started.

Note

Make sure the user does not interrupt the process or the device will be rendered unusable and can only be recovered via the special firmware update cable

Parameters

data	the firmware data
error	pointer to NSError object, where error information is stored in case function fails. You can pass nil if you don't want that information

Returns

TRUE if function succeeded, FALSE otherwise

2.5 Magnetic Stripe Reader Functions (Unencrypted)

Functions to work with the unencrypted magenetic card reader.

Functions

- (BOOL) - DTDevices::msEnable:

    Enables reading of magnetic cards.
- (BOOL) - DTDevices::msDisable:

    Disables magnetic card reading.
- (NSDictionary ∗) - DTDevices::msProcessFinancialCard:track2:

    Helper function to parse financial card and extract the data - name, number, expiration date.
- (BOOL) - DTDevices::msSetCardDataMode:error:

    Sets Linea's magnetic card data mode.

2.5.1 Detailed Description

Functions to work with the unencrypted magenetic card reader.

2.5.2 Function Documentation

2.5.2.1 - (BOOL) msDisable: (NSError ∗∗) *error*

Disables magnetic card reading.

Parameters

error	pointer to NSError object, where error information is stored in case function fails. You can pass nil if you don't want that information

Returns

TRUE if function succeeded, FALSE otherwise

2.5.2.2 - (BOOL) msEnable: (NSError ∗∗) *error*

Enables reading of magnetic cards.

Current magnetic card heads used in Linea consume so little power, that there is no drawback in leaving it enabled all the time. By default magnetic card reading is enabled upon connect.

Parameters

error	pointer to NSError object, where error information is stored in case function fails. You can pass nil if you don't want that information

Returns

TRUE if function succeeded, FALSE otherwise

2.5.2.3 - (NSDictionary ∗) msProcessFinancialCard: (NSString ∗) *track1* track2:(NSString ∗) *track2*

Helper function to parse financial card and extract the data - name, number, expiration date.

The function will extract as much information as possible.

Parameters

track1	- track1 information or nil
track2	- track2 information or nil

Returns

dictionary containing extracted data or nil if the data is invalid. Keys contained are:

"accountNumber"	Account number
"cardholderName"	Cardholder name, as stored in the card
"expirationYear"	Expiration date - year
"expirationMonth"	Expiration date - month
"serviceCode"	Service code (if any)
"discretionaryData"	Discretionary data (if any)
"firstName"	Extracted cardholder's first name
"lastName"	Extracted cardholder's last name

2.5.2.4 - (BOOL) msSetCardDataMode: (int) *mode* error:(NSError ∗∗) *error*

Sets Linea's magnetic card data mode.

This setting is not persistent and is best to configure it upon connect.

Parameters

mode	magnetic card data mode:	
	MS_PROCESSED_CARD_DATA	Card data will be processed and will be returned via call to magneticCardData
	MS_RAW_CARD_DATA	Card data will not be processed and will be returned via call to magneticCardRawData
error	pointer to NSError object, where error information is stored in case function fails. You can pass nil if you don't want that information	

Returns

TRUE if function succeeded, FALSE otherwise

2.6 Barcode Reader Functions

Functions for scanning barcodes, various barcode settings and direct control of the barcode engine.

Functions

- (NSString ∗) - DTDevices::barcodeType2Text:

    Helper function to return string name of barcode type.
- (BOOL) - DTDevices::barcodeStartScan:

    Starts barcode engine.
- (BOOL) - DTDevices::barcodeStopScan:

    Stops ongoing scan started with startScan.
- (BOOL) - DTDevices::barcodeGetScanButtonMode:error:

    Returns the current scan button mode.
- (BOOL) - DTDevices::barcodeSetScanButtonMode:error:

    Sets scan button mode.
- (BOOL) - DTDevices::barcodeSetScanBeep:volume:beepData:length:error:

    Sets the sound, which is used upon successful barcode scan.
- (BOOL) - DTDevices::barcodeGetScanMode:error:

    Returns the current scan mode.
- (BOOL) - DTDevices::barcodeSetScanMode:error:

    Sets barcode engine scan mode.
- (BOOL) - DTDevices::barcodeGetTypeMode:error:

    Returns the current barcode type mode.
- (BOOL) - DTDevices::barcodeSetTypeMode:error:

    Sets barcode type mode.
- (BOOL) - DTDevices::barcodeEngineResetToDefaults:

    Performs factory reset of the barcode module.
- (BOOL) - DTDevices::barcodeEngineCheckReady:error:

    Performs a check if the barcode engine is ready to operate.
- (BOOL) - DTDevices::barcodeOpticonSetInitString:error:

    Allows for a custom initialization string to be sent to the Opticon barcode engine.
- (BOOL) - DTDevices::barcodeOpticonSetParams:saveToFlash:error:

    Sends configuration parameters directly to the opticon barcode engine.
- (NSString ∗) - DTDevices::barcodeOpticonGetIdent:

    Reads barcode engine's identification.
- (BOOL) - DTDevices::barcodeOpticonUpdateFirmware:bootLoader:error:

    Performs firmware update on the optiocon 2D barcode engines.
- (BOOL) - DTDevices::barcodeCodeSetParam:value:error:

    Sends configuration parameters directly to the code barcode engine.
- (BOOL) - DTDevices::barcodeCodeGetParam:value:error:

    Reads configuration parameters directly from the code barcode engine.
- (BOOL) - DTDevices::barcodeCodeUpdateFirmware:data:error:

    Performs firmware update on the Code 2D barcode engines.
- (NSDictionary ∗) - **DTDevices::barcodeCodeGetInformation:**
- (BOOL) - DTDevices::barcodeIntermecSetInitData:error:

    Allows for a custom initialization string to be sent to the Intermec barcode engine.
- (NSData ∗) - DTDevices::barcodeNewlandQuery:error:

    Sends a custom command to the barcode engine and receives a reply.
- (BOOL) - DTDevices::barcodeNewlandSetInitString:error:

    Allows for a custom initialization string to be sent to the Newland barcode engine.

2.6.1 Detailed Description

Functions for scanning barcodes, various barcode settings and direct control of the barcode engine.

2.6.2 Function Documentation

2.6.2.1 - (BOOL) barcodeCodeGetParam: (int) *setting* value:(uint64_t $*$) *value* error:(NSError $**$) *error*

Reads configuration parameters directly from the code barcode engine.

Refer to the barcode engine documentation for supported parameters.

Parameters

setting	the setting number
value	unpon success, the parameter value will be stored here

Returns

TRUE if operation was successful

Parameters

error	pointer to NSError object, where error information is stored in case function fails. You can pass nil if you don't want that information

Returns

TRUE if function succeeded, FALSE otherwise

2.6.2.2 - (BOOL) barcodeCodeSetParam: (int) *setting* value:(uint64_t) *value* error:(NSError $**$) *error*

Sends configuration parameters directly to the code barcode engine.

Use this function with EXTREME care, you can easily render your barcode engine useless. Refer to the barcode engine documentation for supported parameters.

Parameters

setting	the setting number
value	the value to write to

Returns

TRUE if operation was successful

Parameters

error	pointer to NSError object, where error information is stored in case function fails. You can pass nil if you don't want that information

Returns

TRUE if function succeeded, FALSE otherwise

2.6.2.3 - (BOOL) barcodeCodeUpdateFirmware: (NSString ∗) *name* data:(NSData ∗) *data* error:(NSError ∗∗) *error*

Performs firmware update on the Code 2D barcode engines.

Barcode update can take very long time, it is best to call this function from a thread and update the user interface when firmwareUpdateProgress delegate is called

Parameters

name	the exact name of the firmware file
data	firmware file data to load
error	pointer to NSError object, where error information is stored in case function fails. You can pass nil if you don't want that information

Returns

TRUE if function succeeded, FALSE otherwise

2.6.2.4 - (BOOL) barcodeEngineCheckReady: (BOOL ∗) *ready* error:(NSError ∗∗) *error*

Performs a check if the barcode engine is ready to operate.

Parameters

ready	TRUE if the engine is ready to operate, FALSE otherwise
error	pointer to NSError object, where error information is stored in case function fails. You can pass nil if you don't want that information

Returns

TRUE if function succeeded, FALSE otherwise

2.6.2.5 - (BOOL) barcodeEngineResetToDefaults: (NSError ∗∗) *error*

Performs factory reset of the barcode module.

This function is taxing, slow and should not be called often, emergency use only.

Parameters

error	pointer to NSError object, where error information is stored in case function fails. You can pass nil if you don't want that information

Returns

TRUE if function succeeded, FALSE otherwise

2.6.2.6 - (BOOL) barcodeGetScanButtonMode: (int ∗) *mode* error:(NSError ∗∗) *error*

Returns the current scan button mode.

See setScanButtonMode for more detailed description. This setting is not persistent and is best to configure it upon connect.

Parameters

mode	returns scan button mode, one of the: BUTTON_DISABLED — Scan button will become inactive BUTTON_ENABLED — Scan button will triger barcode scan when pressed
error	pointer to NSError object, where error information is stored in case function fails. You can pass nil if you don't want that information

Returns

TRUE if function succeeded, FALSE otherwise

2.6.2.7 - (BOOL) barcodeGetScanMode: (int ∗) *mode* error:(NSError ∗∗) *error*

Returns the current scan mode.

This setting is not persistent and is best to configure it upon connect.

Parameters

mode	returns scanning mode, one of the MODE_∗ constants
error	pointer to NSError object, where error information is stored in case function fails. You can pass nil if you don't want that information

Returns

TRUE if function succeeded, FALSE otherwise

2.6.2.8 - (BOOL) barcodeGetTypeMode: (int ∗) *mode* error:(NSError ∗∗) *error*

Returns the current barcode type mode.

See setBarcodeTypeMode for more detailed description. This setting will not persists.

Parameters

mode	returns barcode type mode, one of the: BARCODE_TYPE_DEFAULT — default barcode types, listed in BARCODES enumeration BARCODE_TYPE_EXTENDED — extended barcode types, listed in BARCODES_EX enumeration
error	pointer to NSError object, where error information is stored in case function fails. You can pass nil if you don't want that information

Returns

TRUE if function succeeded, FALSE otherwise

2.6.2.9 - (BOOL) barcodeIntermecSetInitData: (NSData ∗) *data* error:(NSError ∗∗) *error*

Allows for a custom initialization string to be sent to the Intermec barcode engine.

The data is sent directly, if the barcode is currently powered on, and every time it gets initialized. The setting does not persists, so it is best this command is called upon new connection.

Parameters

data	barcode engine initialization data (consult barcode engine manual)
error	pointer to NSError object, where error information is stored in case function fails. You can pass nil if you don't want that information

Returns

TRUE if function succeeded, FALSE otherwise

2.6.2.10 - (NSData ∗) barcodeNewlandQuery: (NSData ∗) *command* error:(NSError ∗∗) *error*

Sends a custom command to the barcode engine and receives a reply.

Parameters

command	command data (consult barcode engine manual). You must only pass the data field, the header and checksum are automatically calculated
error	pointer to NSError object, where error information is stored in case function fails. You can pass nil if you don't want that information

Returns

response data, if function succeeded, nil otherwise. Response is stripped of headers and checksum, only the real data is provided

2.6.2.11 - (BOOL) barcodeNewlandSetInitString: (NSString ∗) *data* error:(NSError ∗∗) *error*

Allows for a custom initialization string to be sent to the Newland barcode engine.

The string is sent directly, if the barcode is currently powered on, and every time it gets initialized. The settings does persists and are stored in barcode module's flash, but the is written only upon change, so it is safe to repeatedly call this function on every connect.

Parameters

data	barcode engine initialization data (consult barcode engine manual)
error	pointer to NSError object, where error information is stored in case function fails. You can pass nil if you don't want that information

Returns

TRUE if function succeeded, FALSE otherwise

2.6.2.12 - (NSString ∗) barcodeOpticonGetIdent: (NSError ∗∗) *error*

Reads barcode engine's identification.

Parameters

error	pointer to NSError object, where error information is stored in case function fails. You can pass nil if you don't want that information

Returns

opticon engine ident string if function succeeded, nil otherwise

2.6.2.13 - (BOOL) barcodeOpticonSetInitString: (NSString ∗) *data* error:(NSError ∗∗) *error*

Allows for a custom initialization string to be sent to the Opticon barcode engine.

The string is sent directly, if the barcode is currently powered on, and every time it gets initialized. The setting does not persists, so it is best this command is called upon new connection.

Parameters

data	barcode engine initialization data (consult barcode engine manual)
error	pointer to NSError object, where error information is stored in case function fails. You can pass nil if you don't want that information

Returns

TRUE if function succeeded, FALSE otherwise

2.6.2.14 - (BOOL) barcodeOpticonSetParams: (NSString ∗) *data* saveToFlash:(BOOL) *saveToFlash* error:(NSError ∗∗) *error*

Sends configuration parameters directly to the opticon barcode engine.

Use this function with EXTREME care, you can easily render your barcode engine useless. Refer to the barcode engine documentation on supported commands.

The function encapsulates the data with the ESC and CR so you don't have to send them. It optionally sends Z2 after the command to ensure settings are stored in the flash.

You can send multiple parameters with a single call if you format them as follows:

- commands that take 2 symbols can be sent without any delimiters, like: "C1C2C3"
- commands that take 3 symbols should be prefixed by [, like: "C1[C2AC3" (in this case commands are C1, C2A and C3
- commands that take 4 symbols should be prefixed by], like: "C1C2]C3AB" (in this case commands are C1, C2 and C3AB

Parameters

data	command string
saveToFlash	if TRUE, command also saves the settings to flash. Saving setting is slower, so should be in ideal case executed only once and the program to remember it. The scanner's power usually gets cut when device goes to sleep - 5 seconds of idle time, so any non-stored to flash settings are lost, but if barcodeEnginePowerControl:TRUE is used on 2D engine, then even non-saved to flash settings will persist until device disconnects (iOS goes to sleep, physical disconnect)
error	pointer to NSError object, where error information is stored in case function fails. You can pass nil if you don't want that information

Returns

TRUE if function succeeded, FALSE otherwise

2.6.2.15 - (BOOL) barcodeOpticonUpdateFirmware: (NSData ∗) *firmwareData* bootLoader:(BOOL) *bootLoader* error:(NSError ∗∗) *error*

Performs firmware update on the optiocon 2D barcode engines.

Barcode update can take very long time, it is best to call this function from a thread and update the user interface when firmwareUpdateProgress delegate is called

Parameters

firmwareData	firmware file data to load
bootLoader	TRUE if you are going to update bootloader, FALSE if normal firmware
error	pointer to NSError object, where error information is stored in case function fails. You can pass nil if you don't want that information

Returns

TRUE if function succeeded, FALSE otherwise

2.6.2.16 - (BOOL) barcodeSetScanBeep: (BOOL) *enabled* volume:(int) *volume* beepData:(int ∗) *data* length:(int) *length* error:(NSError ∗∗) *error*

Sets the sound, which is used upon successful barcode scan.

This setting is not persistent and is best to configure it upon connect.

Note

A sample beep containing of 2 tones, each with 400ms duration, first one 2000Hz and second - 5000Hz will look int beepData[]={2000,400,5000,400}

Parameters

enabled	turns on or off beeping
volume	controls the volume (0-100). Currently have no effect
data	an array of integer values specifying pairs of tone(Hz) and duration(ms).
length	length in bytes of beepData array
error	pointer to NSError object, where error information is stored in case function fails. You can pass nil if you don't want that information

Returns

TRUE if function succeeded, FALSE otherwise

2.6.2.17 - (BOOL) barcodeSetScanButtonMode: (int) *mode* error:(NSError ∗∗) *error*

Sets scan button mode.

This setting is not persistent and is best to configure it upon connect.

Parameters

mode	button mode, one of the:	
	BUTTON_DISABLED	Scan button will become inactive
	BUTTON_ENABLED	Scan button will triger barcode scan when pressed
error	pointer to NSError object, where error information is stored in case function fails. You can pass nil if you don't want that information	

Returns

TRUE if function succeeded, FALSE otherwise

2.6.2.18 - (BOOL) barcodeSetScanMode: (int) *mode* error:(NSError ∗∗) *error*

Sets barcode engine scan mode.

This setting is not persistent and is best to configure it upon connect.

Parameters

mode	scanning mode, one of the MODE_∗ constants
error	pointer to NSError object, where error information is stored in case function fails. You can pass nil if you don't want that information

Returns

TRUE if function succeeded, FALSE otherwise

2.6.2.19 - (BOOL) barcodeSetTypeMode: (int) *mode* error:(NSError ∗∗) *error*

Sets barcode type mode.

Barcode type can be returned from the default list (listed in BARCODES), extended one (listed in BARCODES_EX) or ISO/AIM list. The extended one is superset to the default, so current programs will be mostly unaffected if they switch from default to extended (with the exception of barcodes like UPC-A and UPC-E, which will be returned as UPC in the default list, but proper types in the extended. This setting will not persists.

Parameters

mode	barcode type mode, one of the:	
	BARCODE_TYPE_DEFAULT (default)	default barcode types, listed in BARCODES enumeration
	BARCODE_TYPE_EXTENDED	extended barcode types, listed in BARCODES_EX enumeration
error	pointer to NSError object, where error information is stored in case function fails. You can pass nil if you don't want that information	

Returns

TRUE if function succeeded, FALSE otherwise

2.6.2.20 - (BOOL) barcodeStartScan: (NSError ∗∗) *error*

Starts barcode engine.

In single scan mode the laser will be on until barcode is successfully read, the timeout elapses (set via call to setScanTimeout) or if stopScan is called. In multi scan mode the laser will stay on even if barcode is successfully read allowing series of barcodes to be scanned within a single read session. The scanning will stop if no barcode is scanned in the timeout interval (set via call to setScanTimeout) or if stopScan is called.

Parameters

error	pointer to NSError object, where error information is stored in case function fails. You can pass nil if you don't want that information

Returns

TRUE if function succeeded, FALSE otherwise

2.6.2.21 - (BOOL) barcodeStopScan: (NSError ∗∗) *error*

Stops ongoing scan started with startScan.

Parameters

error	pointer to NSError object, where error information is stored in case function fails. You can pass nil if you don't want that information

Returns

TRUE if function succeeded, FALSE otherwise

2.6.2.22 - (NSString ∗) barcodeType2Text: (int) *barcodeType*

Helper function to return string name of barcode type.

Parameters

barcodeType	barcode type returned from scanBarcode

Returns

barcode type name

2.7 Bluetooth Functions

Functions to work with the built-in bluetooth module.

Functions

- (BOOL) - DTDevices::btDiscoverSupportedDevicesInBackground:maxTime:filter:error:

  Performs background discovery of nearby supported bluetooth devices.
- (BOOL) - DTDevices::btDiscoverDevicesInBackground:maxTime:codTypes:error:

  Performs background discovery of the nearby bluetooth devices.
- (BOOL) - DTDevices::btDiscoverPrintersInBackground:maxTime:error:

  Performs background discovery of supported printers.
- (BOOL) - DTDevices::btDiscoverPrintersInBackground:

  Performs background discovery of supported printers.
- (BOOL) - DTDevices::btDiscoverPinpadsInBackground:maxTime:error:

  Performs background discovery of supported printers.
- (BOOL) - DTDevices::btDiscoverPinpadsInBackground:

  Performs background discovery of supported printers.
- (BOOL) - DTDevices::btConnect:pin:error:

  Tries to connect to remote device.
- (BOOL) - DTDevices::btDisconnect:error:

  Disconnects from remote device.
- (BOOL) - DTDevices::btConnectSupportedDevice:pin:error:

  Tries to connect to supported bluetooth device.
- (BOOL) - DTDevices::btWrite:length:error:

  Sends data to the connected remote device.
- (BOOL) - DTDevices::btWrite:error:

  Sends data to the connected remote device.
- (int) - DTDevices::btRead:length:timeout:error:

  Tries to read data from the connected remote device for specified timeout.
- (NSString ∗) - DTDevices::btReadLine:error:

  Tries to read string data, ending with CR/LF up to specifed timeout.
- (BOOL) - DTDevices::btEnableWriteCaching:error:

  Enables or disables write caching on the bluetooth stream.
- (NSArray ∗) - DTDevices::btDiscoverDevices:maxTime:codTypes:error:

  Performs synchronous discovery of the nearby bluetooth devices.
- (NSString ∗) - DTDevices::btGetDeviceName:error:

  Queries device name given the address.
- (BOOL) - DTDevices::btSetDataNotificationMaxTime:maxLength:sequenceData:error:

  Sets the conditions to fire the NSStreamEventHasBytesAvailable event on bluetooth streams.
- (BOOL) - DTDevices::btListenForDevices:discoverable:localName:cod:error:

  Initiates/kills listen for incoming bluetooth connections.

Properties

- NSInputStream ∗ DTDevices::btInputStream

  Bluetooth input stream, you can use it after connecting with btConnect.
- NSOutputStream ∗ DTDevices::btOutputStream

  Bluetooth output stream, you can use it after connecting with btConnect.
- NSArray ∗ DTDevices::btConnectedDevices

  Contains bluetooth addresses of the currently connected bluetooth devices or empty array if no connected devices are found.

2.7.1 Detailed Description

Functions to work with the built-in bluetooth module.

2.7.2 Function Documentation

2.7.2.1 - (BOOL) btConnect: (NSString ∗) *address* pin:(NSString ∗) *pin* error:(NSError ∗∗) *error*

Tries to connect to remote device.

Once connection is established, use bluetooth streams to read/write to the remote device.

Note

active connection with remote device will be broken

Parameters

address	bluetooth address returned from btDiscoverDevices/btDiscoverPrinters
pin	PIN code if needed, or nil to try unencrypted connection
error	pointer to NSError object, where error information is stored in case function fails. You can pass nil if you don't want that information

Returns

TRUE if function succeeded, FALSE otherwise

2.7.2.2 - (BOOL) btConnectSupportedDevice: (NSString ∗) *address* pin:(NSString ∗) *pin* error:(NSError ∗∗) *error*

Tries to connect to supported bluetooth device.

Supported devices are the ones the sdk has built-in support for - printers and pinpads. If successful, additional functions will become available and feature notifications will be sent

Note

active connection with remote device will be broken

Parameters

address	bluetooth address returned from btDiscoverSupportedDevicesInBackground/btDiscover-PrintersInBackground
pin	PIN code if needed, or nil to try unencrypted connection
error	pointer to NSError object, where error information is stored in case function fails. You can pass nil if you don't want that information

Returns

TRUE if function succeeded, FALSE otherwise

2.7.2.3 - (BOOL) btDisconnect: (NSString ∗) *address* error:(NSError ∗∗) *error*

Disconnects from remote device.

Parameters

address	bluetooth address returned from btDiscoverDevices/btDiscoverPrinters
error	pointer to NSError object, where error information is stored in case function fails. You can pass nil if you don't want that information

Returns

TRUE if function succeeded, FALSE otherwise

2.7.2.4 - (NSArray ∗) btDiscoverDevices: (int) *maxDevices* maxTime:(double) *maxTime* codTypes:(int) *codTypes* error:(NSError ∗∗) *error*

Performs synchronous discovery of the nearby bluetooth devices.

Implemented for compatibility only!

Deprecated This function is not recommended to be called on the main thread, use btDiscoverDevicesIn-Background instead

Note

this function cannot be called once connection to remote device was established

Parameters

maxDevices	the maximum results to return
maxTime	the max time to discover, in seconds. Actual time may vary.
codTypes	bluetooth Class Of Device to look for or 0 to search for all bluetooth devices
error	pointer to NSError object, where error information is stored in case function fails. You can pass nil if you don't want that information

Returns

array of strings of bluetooth addresses if function succeeded, nil otherwise

2.7.2.5 - (BOOL) btDiscoverDevicesInBackground: (int) *maxDevices* maxTime:(double) *maxTime* codTypes:(int) *codTypes* error:(NSError ∗∗) *error*

Performs background discovery of the nearby bluetooth devices.

The discovery status and devices found will be sent via delegate notifications

Note

active connection with remote device will be broken

Parameters

maxDevices	the maximum results to return
maxTime	the max time to discover, in seconds. Actual time may vary.
codTypes	bluetooth Class Of Device to look for or 0 to search for all bluetooth devices
error	pointer to NSError object, where error information is stored in case function fails. You can pass nil if you don't want that information

Returns

TRUE if function succeeded, FALSE otherwise

2.7.2.6 - (BOOL) btDiscoverPinpadsInBackground: (NSError $**$) *error*

Performs background discovery of supported printers.

These include MPED-400 and PPAD1. The discovery status and devices found will be sent via delegate notifications

Note

active connection with remote device will be broken

Parameters

error	pointer to NSError object, where error information is stored in case function fails. You can pass nil if you don't want that information

Returns

TRUE if function succeeded, FALSE otherwise

2.7.2.7 - (BOOL) btDiscoverPinpadsInBackground: (int) *maxDevices* maxTime:(double) *maxTime* error:(NSError $**$) *error*

Performs background discovery of supported printers.

These include MPED-400 and PPAD1. The discovery status and devices found will be sent via delegate notifications

Note

active connection with remote device will be broken

Parameters

maxDevices	the maximum results to return, default is 4
maxTime	the max time to discover, in seconds. Actual time may vary.
error	pointer to NSError object, where error information is stored in case function fails. You can pass nil if you don't want that information

Returns

TRUE if function succeeded, FALSE otherwise

2.7.2.8 - (BOOL) btDiscoverPrintersInBackground: (NSError $**$) *error*

Performs background discovery of supported printers.

These include PP-60, DPP-250, DPP-350, SM-112, DPP-450. The discovery status and devices found will be sent via delegate notifications

Note

active connection with remote device will be broken

Parameters

error	pointer to NSError object, where error information is stored in case function fails. You can pass nil if you don't want that information

Returns

TRUE if function succeeded, FALSE otherwise

2.7.2.9 - (BOOL) btDiscoverPrintersInBackground: (int) *maxDevices* maxTime:(double) *maxTime* error:(NSError ∗∗) *error*

Performs background discovery of supported printers.

These include PP-60, DPP-250, DPP-350, SM-112, DPP-450. The discovery status and devices found will be sent via delegate notifications

Note

active connection with remote device will be broken

Parameters

maxDevices	the maximum results to return, default is 4
maxTime	the max time to discover, in seconds. Actual time may vary.
error	pointer to NSError object, where error information is stored in case function fails. You can pass nil if you don't want that information

Returns

TRUE if function succeeded, FALSE otherwise

2.7.2.10 - (BOOL) btDiscoverSupportedDevicesInBackground: (int) *maxDevices* maxTime:(double) *maxTime* filter:(int) *filter* error:(NSError ∗∗) *error*

Performs background discovery of nearby supported bluetooth devices.

Supported devices are the ones some of the sdk has built-in support for - printers and pinpads. The discovery status and devices found will be sent via delegate notifications

Note

this function cannot be called once connection to remote device was established

Parameters

maxDevices	the maximum results to return
maxTime	the max time to discover, in seconds. Actual time may vary.
filter	filter of which devices to discover, a combination of one or more of BLUETOOT_FILTER_∗ constants or BLUETOOTH_FILTER_ALL to get all supported devices
error	pointer to NSError object, where error information is stored in case function fails. You can pass nil if you don't want that information

Returns

TRUE if function succeeded, FALSE otherwise

2.7.2.11 - (BOOL) btEnableWriteCaching: (BOOL) *enabled* error:(NSError $**$) *error*

Enables or disables write caching on the bluetooth stream.

When enabled the writes gets cached and send on bigger chunks, reducing substantially the time taken, if you are sending lot of data in small parts. Write caching has negative effect on the speed if your bluetooth communication is based on request/response format or packets, in this case every write operation will get delayed, resulting in very poor throughput.

Parameters

enabled	enable or disable write caching, by default it is disabled
error	pointer to NSError object, where error information is stored in case function fails. You can pass nil if you don't want that information

Returns

TRUE if function succeeded, FALSE otherwise

2.7.2.12 - (NSString $*$) btGetDeviceName: (NSString $*$) *address* error:(NSError $**$) *error*

Queries device name given the address.

Implemented for compatibility only!

Deprecated This function complements the btDiscoverDevices/btDiscoverPrinters and as such is not recommended, use btDiscoverDevicesInBackground instead

Note

this function cannot be called once connection to remote device was established

Parameters

address	bluetooth address returned from btDiscoverDevices/btDiscoverPrinters
error	pointer to NSError object, where error information is stored in case function fails. You can pass nil if you don't want that information

Returns

bluetooth device name if function succeeded, nil otherwise

2.7.2.13 - (BOOL) btListenForDevices: (BOOL) *enabled* discoverable:(bool) *discoverable* localName:(NSString $*$) *localName* cod:(uint32_t) *cod* error:(NSError $**$) *error*

Initiates/kills listen for incoming bluetooth connections.

Incoming connecton requests will be sent as delegate notifications

Note

this function cannot be called once connection to remote device was established

Parameters

enabled	if YES the bluetooth module will listen for incoming connections, NO disables this functionality
discoverable	if YES the module will be discoverable while waiting. Making the module non-discoverable means only devices, that know it's bluetooth address will be able to connect
localName	if discoverable, then this will be the name seen by the others
cod	Class Of Device, as per bluetooth documentation. Pass 0 if you don't want to set it
error	pointer to NSError object, where error information is stored in case function fails. You can pass nil if you don't want that information

Returns

TRUE if function succeeded, FALSE otherwise

2.7.2.14 - (int) btRead: (void ∗) *data* length:(int) *length* timeout:(double) *timeout* error:(NSError ∗∗) *error*

Tries to read data from the connected remote device for specified timeout.

Note

You can use bluethooth streams instead

Parameters

data	data buffer, where the result will be stored
length	maximum amount of bytes to wait for
timeout	maximim timeout in seconds to wait for data

Returns

the

Parameters

error	pointer to NSError object, where error information is stored in case function fails. You can pass nil if you don't want that information

Returns

actual number of bytes stored in the data buffer if function succeeded, -1 otherwise

2.7.2.15 - (NSString ∗) btReadLine: (double) *timeout* error:(NSError ∗∗) *error*

Tries to read string data, ending with CR/LF up to specifed timeout.

Note

You can use bluethooth streams instead

Parameters

timeout	maximim timeout in seconds to wait for data
error	pointer to NSError object, where error information is stored in case function fails. You can pass nil if you don't want that information

Returns

string with the line read (can be empty string too) if function succeeded, nil otherwise

2.7.2.16 - (BOOL) btSetDataNotificationMaxTime: (double) *maxTime* maxLength:(int) *maxLength* sequenceData:(NSData ∗) *sequenceData* error:(NSError ∗∗) *error*

Sets the conditions to fire the NSStreamEventHasBytesAvailable event on bluetooth streams.

If all special conditions are disabled, then the notification will be fired the moment data arrives. You can have multiple notifications active at the same time, for example maxBytes and maxTime.

Parameters

maxTime	notification will be fired 'maxTime' seconds after the last byte arrives, passing 0 disables it. For example 0.1 means that 100ms after the last byte is received the notification will fire.
maxLength	notification will be fired after 'maxLength' data arrives, passing 0 disables it.
sequenceData	notification will be fired if the received data contains 'sequenceData', passing nil disables it.

2.7.2.17 - (BOOL) btWrite: (NSString ∗) *data* error:(NSError ∗∗) *error*

Sends data to the connected remote device.

Note

You can use bluethooth streams instead

Parameters

data	data string to write
error	pointer to NSError object, where error information is stored in case function fails. You can pass nil if you don't want that information

Returns

TRUE if function succeeded, FALSE otherwise

2.7.2.18 - (BOOL) btWrite: (void ∗) *data* length:(int) *length* error:(NSError ∗∗) *error*

Sends data to the connected remote device.

Note

You can use bluethooth streams instead

Parameters

data	data bytes to write
length	the length of the data in the buffer
error	pointer to NSError object, where error information is stored in case function fails. You can pass nil if you don't want that information

Returns

TRUE if function succeeded, FALSE otherwise

2.7.3 Properties

2.7.3.1 - (NSInputStream∗) btInputStream `[read], [atomic], [assign]`

Bluetooth input stream, you can use it after connecting with btConnect.

See NSInputStream documentation.

2.7.3.2 - (NSOutputStream∗) btOutputStream `[read], [atomic], [assign]`

Bluetooth output stream, you can use it after connecting with btConnect.

See NSOutputStream documentation.

2.8 External Serial Port Functions

Functions to work with Linea Tab's external serial port.

Macros

- #define PARITY_NONE 0

  No parity.
- #define PARITY_EVEN 1

  Even parity.
- #define PARITY_ODD 2

  Odd parity.
- #define DATABITS_7 1

  7 data bits
- #define DATABITS_8 0

  8 data bits
- #define STOPBITS_1 0

  1 stop bits
- #define STOPBITS_2 1

  2 stop bits
- #define FLOW_NONE 0

  No flow control.
- #define FLOW_RTS_CTS 1

  RTS/CTS hardware flow control.
- #define FLOW_DTR_DSR 2

  DSR/DTR hardware flow control.
- #define FLOW_XON_XOFF 3

  XON/XOFF software flow control.

Functions

- (BOOL) - DTDevices::extOpenSerialPort:baudRate:parity:dataBits:stopBits:flowControl:error:

  Opens the external serial port with specified settings.
- (BOOL) - DTDevices::extCloseSerialPort:error:

  Closes the external serial port opened with extOpenSerialPort.
- (BOOL) - DTDevices::extWriteSerialPort:data:error:

  Sends data to the connected remote device via serial port.
- (NSData ∗) - DTDevices::extReadSerialPort:length:timeout:error:

  Reads data from the connected remote device via serial port.

2.8.1 Detailed Description

Functions to work with Linea Tab's external serial port.

2.8.2 Function Documentation

2.8.2.1 - (BOOL) extCloseSerialPort: (int) *port* error:(NSError ∗∗) *error*

Closes the external serial port opened with extOpenSerialPort.

Parameters

port	the port number, currently only 1 is used
error	returns error information, you can pass nil if you don't want it

Returns

TRUE upon success, FALSE otherwise

2.8.2.2 - (BOOL) extOpenSerialPort: (int) *port* baudRate:(int) *baudRate* parity:(int) *parity* dataBits:(int) *dataBits* stopBits:(int) *stopBits* flowControl:(int) *flowControl* error:(NSError ∗∗) *error*

Opens the external serial port with specified settings.

Parameters

port	the port number, currently only 1 is used
baudRate	serial baud rate
parity	serial parity, one of the PARITY_∗ constants (curenty only PARITY_NONE is supported)
dataBits	serial data bits, one of the DATABITS_∗ constants (currently only DATABITS_8 is supported)
stopBits	serial stop bits, one of the STOPBITS_∗ constants (currently only STOPBITS_1 is supported)
flowControl	serial flow control, one of the FLOW_∗ constants (currently only FLOW_NONE is supported)
error	returns error information, you can pass nil if you don't want it

Returns

TRUE upon success, FALSE otherwise

2.8.2.3 - (NSData ∗) extReadSerialPort: (int) *port* length:(int) *length* timeout:(double) *timeout* error:(NSError ∗∗) *error*

Reads data from the connected remote device via serial port.

Parameters

port	the port number, currently only 1 is used
length	the maximum amount of data to read
timeout	timeout in seconds, passing 0 reads and returns the bytes currently in the buffer
error	pointer to NSError object, where error information is stored in case function fails. You can pass nil if you don't want that information

Returns

NSData with bytes received if function succeeded, nil otherwise

2.8.2.4 - (BOOL) extWriteSerialPort: (int) *port* data:(NSData ∗) *data* error:(NSError ∗∗) *error*

Sends data to the connected remote device via serial port.

Parameters

port	the port number, currently only 1 is used
data	data bytes to write
error	pointer to NSError object, where error information is stored in case function fails. You can pass nil if you don't want that information

Returns

TRUE if function succeeded, FALSE otherwise

2.9 TCP/IP Functions

Functions to work with the supported devices over the network.

Functions

- (BOOL) - DTDevices::tcpConnectSupportedDevice:error:

  Tries to connect to supported device over the network.
- (BOOL) - DTDevices::tcpDisconnect:error:

  Disconnects from remote device.

Properties

- NSArray ∗ DTDevices::tcpConnectedDevices

  Contains tcp addresses of the currently connected network devices or empty array if no connected devices are found.

2.9.1 Detailed Description

Functions to work with the supported devices over the network.

2.9.2 Function Documentation

2.9.2.1 - (BOOL) tcpConnectSupportedDevice: (NSString ∗) *address* error:(NSError ∗∗) *error*

Tries to connect to supported device over the network.

Supported devices are the ones the sdk has built-in support for - printers and pinpads. If successful, additional functions will become available and feature notifications will be sent

Note

active connection with remote device will be broken

Parameters

address	network address, either dns or IP. Optionaly provide a port number separated by : i.e. address-:port. Default port is 9100.
error	pointer to NSError object, where error information is stored in case function fails. You can pass nil if you don't want that information

Returns

TRUE if function succeeded, FALSE otherwise

2.9.2.2 - (BOOL) tcpDisconnect: (NSString ∗) *address* error:(NSError ∗∗) *error*

Disconnects from remote device.

Parameters

address	the address to disconnect from, the same one that was used to connect to tcpConnectConnect-SupportedDevice
error	pointer to NSError object, where error information is stored in case function fails. You can pass nil if you don't want that information

Returns

TRUE if function succeeded, FALSE otherwise

2.10 Cryptographic & Security Functions

Starting from firmware 2.13, Linea provides strong cryptographic support for magnetic card data.

Functions

- (NSData ∗) - DTDevices::cryptoRawGenerateRandomData:

  Generates 16 byte block of random numbers, required for some of the other crypto functions.
- (BOOL) - DTDevices::cryptoRawSetKey:encryptedData:keyVersion:keyFlags:error:
- (BOOL) - DTDevices::cryptoSetKey:key:oldKey:keyVersion:keyFlags:error:

  Used to store AES256 keys into Linea internal memory.
- (BOOL) - DTDevices::cryptoGetKeyVersion:keyVersion:error:

  Returns key version.
- (NSData ∗) - DTDevices::cryptoRawAuthenticateDevice:error:
- (BOOL) - DTDevices::cryptoAuthenticateDevice:error:
- (BOOL) - DTDevices::cryptoRawAuthenticateHost:error:
- (BOOL) - DTDevices::cryptoAuthenticateHost:error:

2.10.1 Detailed Description

Starting from firmware 2.13, Linea provides strong cryptographic support for magnetic card data. The encryption is supported on all Linea devices, from software point of view they are all the same, but provide different levels of hardware/firmware security.

An overview of the security, provided by Linea (see each of the crypto functions for further detail):

Hardware/Firmware:

For magnetic card encryption Linea is using AES256, which is the current industry standard encryption algorithm. The encryption key resides in volatile, battery powered ram inside Linea's cpu (for Linea 1.5 onward) and is being lost if anyone tries to break in the Linea device in order to prevent the key from being stolen. Magnetic card data, along with device serial number and some random bytes (to ensure every packet will be different) are being sent to the iOS program in an encrypted way.

Software:

Currently there are 2 types of keys, that can be loaded into Linea:

- AUTHENTICATION KEY - used for device authentication (for example the program can lock itself to work with very specific Linea device) and encryption of the firmware

- ENCRYPTION KEY - used for magnetic card data encryption. To use msr encryption, you don't need to set the AUTHENTICATION KEY.

Keys: The keys can be set/changed in two ways:

1. Using plain key data - this method is easy to use, but less secure, as it relies on program running on iPod/iPhone to have the key inside, an attacker could compromise the system and extract the key from device's memory. Call cryptoSetKey to set the keys this way. If there is an existing key of the same type inside Linea, you have to pass it too.

2. Using encrypted key data - this method is harder to implement, but provides better security - the key data, encrypted with old key data is sent from a server in secure environment to the program, running on the iOS, then the program forwards it to the Linea. The program itself have no means to decrypt the data, so an attacker can't possibly extract the key. Refer to cryptoSetKey documentation for more detailed description of the loading process.

The initial loading of the keys should always be done in a secure environment.

Magnetic card encryption:

Once ENCRYPTION KEY is set, all magnetic card data gets encrypted, and is now sent via magneticCard-EncryptedData instead. The LineaDemo program contains sample code to decrypt the data block and extract the contents - the serial number and track data.

As with keys, card data can be extracted on the iOS device itself (less secure, the application needs to have the key inside) or be sent to a secure server to be processed. Note, that the encrypted data contains Linea's serial number too, this can be used for Data Origin Verification, to be sure someone is not trying to mimic data, coming from another device.

Demo program: the sample program now have "Crypto" tab, where key management can be tested:

- New AES 256 key - type in the key you want to set (or change to)
- Old AES 256 key - type in the previous key, or leave blank if you set the key for the first time

[SET AUTHENTICATION KEY] and [SET ENCRYPTION KEY] buttons allow you to use the key info in the text fields above to set the key.

- Decryption key - type in the key, which the demo program will use to try to decrypt card data. This field should contain the ENCRYPTION KEY, or something random, if you want to test failed card data decryption.

2.10.2 Function Documentation

2.10.2.1 - (BOOL) cryptoAuthenticateDevice: (NSData ∗) *key* error:(NSError ∗∗) *error*

Note

Check out the cryptoRawAuthenticateDevice function, if you want to not use the key inside the mobile device.

Generates random data, uses the key to encrypt it, then encrypts the same data with the stored authentication key inside Linea and returns true if both data matches.

The idea: if a program wants to work with specific Linea device, it sets AES256 authentication key once, then on every connect the program uses cryptoAuthenticateDevice with that key. If Linea contains no key, or the key is different, the function will return FALSE. This does not block Linea from operation, what action will be taken if devices mismatch depends on the program.

Parameters

key	32 bytes AES256 key
error	pointer to NSError object, where error information is stored in case function fails. You can pass nil if you don't want that information

Returns

TRUE if if Linea contains the same authentication key, FALSE otherwise

2.10.2.2 - (BOOL) cryptoAuthenticateHost: (NSData ∗) *key* error:(NSError ∗∗) *error*

Note

Check out the cryptoRawAuthenticateHost function, if you want to not use the key inside the mobile device.

Generates random data, uses the key to encrypt it, then sends to Linea to verify against it's internal authentication key. If both keys match, return value is TRUE. This function is used so that Linea knows a "real" device is currently connected, before allowing some functionality. Currently firmware update is protected by this function, once authentication key is set, you have to use it or cryptoRawAuthenticateHost before you attempt firmware update, or it will error out.

Parameters

key	32 bytes AES256 key
error	pointer to NSError object, where error information is stored in case function fails. You can pass nil if you don't want that information

Returns

TRUE if Linea contains the same authentication key, FALSE otherwise

2.10.2.3 - (BOOL) cryptoGetKeyVersion: (int) *keyID* keyVersion:(uint32_t ∗) *keyVersion* error:(NSError ∗∗) *error*

Returns key version.

Valid key ID:

- KEY_AUTHENTICATION - if set, you can use authentication functions - cryptoRawAuthenticateDevice or cryptoAuthenticateDevice. Firmware updates will require authentication too
- KEY_ENCRYPTION - if set, magnetic card data will come encrypted via magneticCardEncryptedData or magneticCardEncryptedRawData

Parameters

keyVersion	returns key version or 0 if the key is not present (key versions are available in firmware 2.43 or later)
error	pointer to NSError object, where error information is stored in case function fails. You can pass nil if you don't want that information

Returns

TRUE if function succeeded, FALSE otherwise

2.10.2.4 - (NSData ∗) cryptoRawAuthenticateDevice: (NSData ∗) *randomData* error:(NSError ∗∗) *error*

Note

RAW crypto functions are harder to use and require more code, but are created to allow no secret keys to reside on the device, but all the operations can be executed with data, sent from a secure server. See crypto-AuthenticateDevice if you plan to use the key in the mobile device.

Encrypts a 16 bytes block of random data with the stored authentication key and returns the result.

The idea: if a program wants to work with specific Linea device, it sets AES256 authentication key once, then on every connect the program generates random 16 byte block of data, encrypts it internally with the said key, then encrypts it with linea too and compares the result. If that Linea contains no key, or the key is different, the resulting data will totally differ from the one generated. This does not block Linea from operation, what action will be taken if devices mismatch depends on the program.

Parameters

randomData	16 bytes block of data (presumably random bytes)
error	pointer to NSError object, where error information is stored in case function fails. You can pass nil if you don't want that information

Returns

random data, encrypted with the Linea authentication key if function succeeded, nil otherwise

2.10.2.5 - (BOOL) cryptoRawAuthenticateHost: (NSData ∗) *encryptedRandomData* error:(NSError ∗∗) *error*

Note

RAW crypto functions are harder to use and require more code, but are created to allow no secret keys to reside on the device, but all the operations can be executed with data, sent from a secure server. See crypto-AuthenticateHost if you plan to use the key in the mobile device.

Tries to decrypt random data, generated from cryptoRawGenerateRandomData with the stored internal authentication key and returns the result. This function is used so that Linea knows a "real" device is currently connected, before allowing some functionality. Currently firmware update is protected by this function, once authentication key is set, you have to use it or cryptoAuthenticateHost before you attempt firmware update, or it will error out.

The idea (considering the iOS device does not have the keys inside, but depends on server):

- (iOS program) generates random data using cryptoRawGenerateRandomData and sends to the server
- (Server) encrypts the random data with the same AES256 key that is in the Linea and sends back to the iOS program
- (iOS program) uses cryptoRawAuthenticateHost to authenticate with the data, function will error out if authentication fails.

Parameters

encrypted-RandomData	16 bytes block of encrypted data
error	pointer to NSError object, where error information is stored in case function fails. You can pass nil if you don't want that information

Returns

TRUE if function succeeded, FALSE otherwise

2.10.2.6 - (NSData ∗) cryptoRawGenerateRandomData: (NSError ∗∗) *error*

Generates 16 byte block of random numbers, required for some of the other crypto functions.

Parameters

error	pointer to NSError object, where error information is stored in case function fails. You can pass nil if you don't want that information

Returns

16 bytes of random numbers if function succeeded, nil otherwise

2.10.2.7 - (BOOL) cryptoRawSetKey: (int) *keyID* encryptedData:(NSData ∗) *encryptedData* keyVersion:(uint32_t) *keyVersion* keyFlags:(uint32_t) *keyFlags* error:(NSError ∗∗) *error*

Note

RAW crypto functions are harder to use and require more code, but are created to allow no secret keys to reside on the device, but all the operations can be executed with data, sent from a secure server. See cryptoSetKey if you plan to use the key in the mobile device.

Used to store AES256 keys into Linea internal memory. Valid keys that can be set:

- KEY_AUTHENTICATION - if set, you can use authentication functions - cryptoRawAuthenticateDevice or cryptoAuthenticateDevice. Firmware updates will require authentication too

- KEY_ENCRYPTION - if set, magnetic card data will come encrypted via magneticCardEncryptedData or magneticCardEncryptedRawData

Generally the key loading process, using "Raw" commands, a program on the iOS device and a server which holds the keys will look similar to:

- (iOS program) calls cryptoRawGenerateRandomData to get 16 bytes block of random data and send these to the server
- (Server) creates byte array of 48 bytes consisting of: [RANDOM DATA: 16 bytes][KEY DATA: 32 bytes]
- (Server) if there is current encryption key set on the Linea (if you want to change existing key) the server encrypts the 48 bytes block with the OLD key
- (Server) sends the result data back to the program
- (iOS program) calls cryptoRawSetKey with KEY_ENCRYPTION and the data it received from the server
- (Linea) tries to decrypt the key data if there was already key present, then extracts the key, verifies the random data and if everything is okay, sets the key

Parameters

keyID	the key type to set - KEY_AUTHENTICATION or KEY_ENCRYPTION
encryptedData	- 48 bytes that consists of 16 bytes random numbers received via call to cryptoRawGenerate-RandomData and 32 byte AES256 key. If there has been previous key of the same type, then all 48 bytes should be encrypted with it.
keyVersion	- the version of the key. On firmware versions less than 2.43 this parameter is ignored and key version is considered to be 0x00000000. Key version is useful for the program to determine what key is inside the head.
keyFlags	- optional key flags, supported on ver 2.58 and onward

- KEY_AUTHENTICATION:

BIT 1	If set to 1, scanning barcodes, reading magnetic card and using the bluetooth module are locked and have to be unlocked with cryptoAuthenticate-Host/cryptoRawAuthenticateHost upon every reinsert of the device

- KEY_ENCRYPTION: No flags are supported

Parameters

error	pointer to NSError object, where error information is stored in case function fails. You can pass nil if you don't want that information

Returns

TRUE if function succeeded, FALSE otherwise

2.10.2.8 - (BOOL) cryptoSetKey: (int) *keyID* key:(NSData $*$) *key* oldKey:(NSData $*$) *oldKey* keyVersion:(uint32_t) *keyVersion* keyFlags:(uint32_t) *keyFlags* error:(NSError $**$) *error*

Used to store AES256 keys into Linea internal memory.

Valid keys that can be set:

- KEY_AUTHENTICATION - if set, you can use authentication functions - cryptoRawAuthenticateDevice or cryptoAuthenticateDevice. Firmware updates will require authentication too
- KEY_ENCRYPTION - if set, magnetic card data will come encrypted via magneticCardEncryptedData or magneticCardEncryptedRawData

Parameters

keyID	the key type to set - KEY_AUTHENTICATION or KEY_ENCRYPTION
key	32 bytes AES256 key to set
oldKey	32 bytes AES256 key that was previously used, or null if there was no previous key. The old key should match the new key, i.e. if you are setting KEY_ENCRYPTION, then you should pass the old KEY_ENCRYPTION.
keyVersion	- the version of the key. On firmware versions less than 2.43 this parameter is ignored and key version is considered to be 0x00000000. Key version is useful for the program to determine what key is inside the head.
keyFlags	- optional key flags, supported on ver 2.58 and onward

- KEY_AUTHENTICATION:

BIT 1	If set to 1, scanning barcodes, reading magnetic card and using the bluetooth module are locked and have to be unlocked with cryptoAuthenticate-Host/cryptoRawAuthenticateHost upon every reinsert of the device

- KEY_ENCRYPTION: No flags are supported

Parameters

error	pointer to NSError object, where error information is stored in case function fails. You can pass nil if you don't want that information

Returns

TRUE if function succeeded, FALSE otherwise

2.11 Encrypted Magnetic Head Functions

Functions to work with encrypted magnetic head.

Macros

- #define **LN_EMSR_EBASE** -11000
- #define LN_EMSR_EINVALID_COMMAND LN_EMSR_EBASE-0x01

    Encrypted magnetic head invalid command sent.
- #define LN_EMSR_ENO_PERMISSION LN_EMSR_EBASE-0x02

    Encrypted magnetic head no permission error.
- #define LN_EMSR_ECARD LN_EMSR_EBASE-0x03

    Encrypted magnetic head card error.
- #define LN_EMSR_ESYNTAX LN_EMSR_EBASE-0x04

    Encrypted magnetic head command syntax error.
- #define LN_EMSR_ENO_RESPONSE LN_EMSR_EBASE-0x05

    Encrypted magnetic head command no response from the magnetic chip.
- #define LN_EMSR_ENO_DATA LN_EMSR_EBASE-0x06

    Encrypted magnetic head no data available.
- #define LN_EMSR_EINVALID_LENGTH LN_EMSR_EBASE-0x14

    Encrypted magnetic head invalid data length.
- #define LN_EMSR_ETAMPERED LN_EMSR_EBASE-0x15

    Encrypted magnetic head is tampered.
- #define LN_EMSR_EINVALID_SIGNATURE LN_EMSR_EBASE-0x16

    Encrypted magnetic head invalid signature.
- #define LN_EMSR_EHARDWARE LN_EMSR_EBASE-0x17

    Encrypted magnetic head hardware failure.

Functions

- (BOOL) - DTDevices::emsrSetActiveHead:error:

    In case there are multiple encrypted heads on the device, sets the active one.
- (NSDictionary ∗) - DTDevices::emsrGetFirmwareInformation:error:

    Returns information about the specified head firmware data.
- (BOOL) - DTDevices::emsrIsTampered:error:

    Checks if the head was tampered or not.
- (BOOL) - DTDevices::emsrGetKeyVersion:keyVersion:error:

    Retrieves the key version (if any) of a loaded key.
- (BOOL) - DTDevices::emsrLoadInitialKey:error:

    Loads Terminal Master Key (TMK) or reenable after tampering.
- (BOOL) - DTDevices::emsrLoadKey:error:

    Loads new key, in plain or encrypted with already loaded AES256 Key Encryption Key (KEK).
- (NSData ∗) - DTDevices::emsrGetDUKPTSerial:

    Returns DUKPT serial number, if DUKPT key is set.
- (NSString ∗) - DTDevices::emsrGetDeviceModel:

    Returns head's model.
- (BOOL) - DTDevices::emsrGetFirmwareVersion:error:

    Returns head's firmware version as number MAJOR∗100+MINOR, i.e.
- (BOOL) - DTDevices::emsrGetSecurityVersion:error:

    Returns head's security version as number MAJOR∗100+MINOR, i.e.

- (NSData ∗) - DTDevices::emsrGetSerialNumber:

  Return head's unique serial number as byte array.
- (BOOL) - DTDevices::emsrUpdateFirmware:error:

  Performs firmware update on the encrypted head.
- (NSArray ∗) - DTDevices::emsrGetSupportedEncryptions:

  Returns supported encryption algorhtms by the encrypted head.
- (BOOL) - DTDevices::emsrSetEncryption:params:error:

  Selects the prefered encryption algorithm.
- (BOOL) - DTDevices::emsrSetEncryption:keyID:params:error:

  Selects the prefered encryption algorithm.
- (BOOL) - DTDevices::emsrConfigMaskedDataShowExpiration:unmaskedDigitsAtStart:unmaskedDigitsAtEnd:error:

  Fine-tunes which part of the card data will be masked, and which will be sent in clear text for display/print purposes.
- (BOOL) - DTDevices::emsrConfigMaskedDataShowExpiration:unmaskedDigitsAtStart:unmaskedDigitsAtEnd:unmaskedDigitsAfter:error:

  Fine-tunes which part of the card data will be masked, and which will be sent in clear text for display/print purposes.
- (BOOL) - **DTDevices::emsrLoadRSAKeyPEM:version:error:**
- (EMSRDeviceInfo ∗) - DTDevices::emsrGetDeviceInfo:

  Returns general information about the encrypted head - firmware version, ident, serial number.
- (EMSRKeysInfo ∗) - DTDevices::emsrGetKeysInfo:

  Returns information about the loaded keys in the encrypted head and tampered status.

2.11.1 Detailed Description

Functions to work with encrypted magnetic head.

2.11.2 Macro Definition Documentation

2.11.2.1 #define LN_EMSR_ECARD LN_EMSR_EBASE-0x03

Encrypted magnetic head card error.

2.11.2.2 #define LN_EMSR_EHARDWARE LN_EMSR_EBASE-0x17

Encrypted magnetic head hardware failure.

2.11.2.3 #define LN_EMSR_EINVALID_COMMAND LN_EMSR_EBASE-0x01

Encrypted magnetic head invalid command sent.

2.11.2.4 #define LN_EMSR_EINVALID_LENGTH LN_EMSR_EBASE-0x14

Encrypted magnetic head invalid data length.

2.11.2.5 #define LN_EMSR_EINVALID_SIGNATURE LN_EMSR_EBASE-0x16

Encrypted magnetic head invalid signature.

2.11.2.6 #define LN_EMSR_ENO_DATA LN_EMSR_EBASE-0x06

Encrypted magnetic head no data available.

2.11.2.7 #define LN_EMSR_ENO_PERMISSION LN_EMSR_EBASE-0x02

Encrypted magnetic head no permission error.

2.11.2.8 #define LN_EMSR_ENO_RESPONSE LN_EMSR_EBASE-0x05

Encrypted magnetic head command no response from the magnetic chip.

2.11.2.9 #define LN_EMSR_ESYNTAX LN_EMSR_EBASE-0x04

Encrypted magnetic head command syntax error.

2.11.2.10 #define LN_EMSR_ETAMPERED LN_EMSR_EBASE-0x15

Encrypted magnetic head is tampered.

2.11.3 Function Documentation

2.11.3.1 - (BOOL) emsrConfigMaskedDataShowExpiration: (BOOL) *showExpiration* unmaskedDigitsAtStart:(int) *unmaskedDigitsAtStart* unmaskedDigitsAtEnd:(int) *unmaskedDigitsAtEnd* error:(NSError ∗∗) *error*

Fine-tunes which part of the card data will be masked, and which will be sent in clear text for display/print purposes.

Parameters

showExpiration	if set to TRUE, expiration date will be shown in clear text, otherwise will be masked
unmaskedDigits-AtStart	the number of digits to show in clear text at the start of the PAN, range from 0 to 6 (default is 4)
unmaskedDigits-AtEnd	the number of digits to show in clear text at the end of the PAN, range from 0, to 4 (default is 4)
error	pointer to NSError object, where error information is stored in case function fails. You can pass nil if you don't want that information

Returns

TRUE if function succeeded, FALSE otherwise

2.11.3.2 - (BOOL) emsrConfigMaskedDataShowExpiration: (BOOL) *showExpiration* unmaskedDigitsAtStart:(int) *unmaskedDigitsAtStart* unmaskedDigitsAtEnd:(int) *unmaskedDigitsAtEnd* unmaskedDigitsAfter:(int) *unmaskedDigitsAfter* error:(NSError ∗∗) *error*

Fine-tunes which part of the card data will be masked, and which will be sent in clear text for display/print purposes.

Parameters

showExpiration	if set to TRUE, expiration date will be shown in clear text, otherwise will be masked
unmaskedDigits-AtStart	the number of digits to show in clear text at the start of the PAN, range from 0 to 6 (default is 4)
unmaskedDigits-AtEnd	the number of digits to show in clear text at the end of the PAN, range from 0, to 4 (default is 4)

unmaskedDigits-After	the number of digits to show in clear after the PAN (starting with expiration date), range from 0 to 6 (default is 0). The first 4 digits are the expiration date, if the showExpiration parameter is enabled, then at least 4 digits will be unmasked. Supported only on pinpads.
error	pointer to NSError object, where error information is stored in case function fails. You can pass nil if you don't want that information

Returns

TRUE if function succeeded, FALSE otherwise

2.11.3.3 - (EMSRDeviceInfo ∗) emsrGetDeviceInfo: (NSError ∗∗) *error*

Returns general information about the encrypted head - firmware version, ident, serial number.

Parameters

error	pointer to NSError object, where error information is stored in case function fails. You can pass nil if you don't want that information

Returns

EMSRDeviceInfo object if function succeeded, nil otherwise

2.11.3.4 - (NSString ∗) emsrGetDeviceModel: (NSError ∗∗) *error*

Returns head's model.

Returns

head's model as string

Parameters

error	pointer to NSError object, where error information is stored in case function fails. You can pass nil if you don't want that information

Returns

TRUE if function succeeded, FALSE otherwise

2.11.3.5 - (NSData ∗) emsrGetDUKPTSerial: (NSError ∗∗) *error*

Returns DUKPT serial number, if DUKPT key is set.

Parameters

error	pointer to NSError object, where error information is stored in case function fails. You can pass nil if you don't want that information

Returns

serial number or nil if an error occured

2.11.3.6 - (NSDictionary ∗) emsrGetFirmwareInformation: (NSData ∗) *data* error:(NSError ∗∗) *error*

Returns information about the specified head firmware data.

Based on it, and the current head's name and firmware version you can chose to update or not the head's firmware

Parameters

data	- firmware data

Returns

dictionary containing extracted data or nil if the data is invalid. Keys contained are:

"deviceModel"	Head's model, for example "EMSR-DEA"
"firmwareRevision"	Firmware revision as string, for example 1.07
"firmwareRevisionNumber"	Firmware revision as number MAJOR∗100+MINOR, i.e. 1.07 will be returned as 107

Parameters

error	pointer to NSError object, where error information is stored in case function fails. You can pass nil if you don't want that information

Returns

TRUE if function succeeded, FALSE otherwise

2.11.3.7 - (BOOL) emsrGetFirmwareVersion: (int ∗) *version* error:(NSError ∗∗) *error*

Returns head's firmware version as number MAJOR∗100+MINOR, i.e.

version 1.05 will be sent as 105

Parameters

version	integer, where firmware version is stored upon success
error	pointer to NSError object, where error information is stored in case function fails. You can pass nil if you don't want that information

Returns

TRUE if function succeeded, FALSE otherwise

2.11.3.8 - (EMSRKeysInfo ∗) emsrGetKeysInfo: (NSError ∗∗) *error*

Returns information about the loaded keys in the encrypted head and tampered status.

Parameters

error	pointer to NSError object, where error information is stored in case function fails. You can pass nil if you don't want that information

Returns

EMSRKeysInfo object if function succeeded, nil otherwise

2.11.3.9 - (BOOL) emsrGetKeyVersion: (int) *keyID* keyVersion:(int ∗) *keyVersion* error:(NSError ∗∗) *error*

Retrieves the key version (if any) of a loaded key.

Parameters

keyID	the ID of the key to get the version, one of the KEY_∗ constants
keyVersion	- pointer to integer, where key version will be returned upon success. Key version can be 0.
error	pointer to NSError object, where error information is stored in case function fails. You can pass nil if you don't want that information

Returns

TRUE if function succeeded, FALSE otherwise

2.11.3.10 - (BOOL) emsrGetSecurityVersion: (int ∗) *version* error:(NSError ∗∗) *error*

Returns head's security version as number MAJOR∗100+MINOR, i.e.

version 1.05 will be sent as 105. Security version is the version of the certificated security kernel.

Parameters

version	integer, where firmware version is stored upon success
error	pointer to NSError object, where error information is stored in case function fails. You can pass nil if you don't want that information

Returns

TRUE if function succeeded, FALSE otherwise

2.11.3.11 - (NSData ∗) emsrGetSerialNumber: (NSError ∗∗) *error*

Return head's unique serial number as byte array.

Parameters

error	pointer to NSError object, where error information is stored in case function fails. You can pass nil if you don't want that information

Returns

serial number or nil if an error occured

2.11.3.12 - (NSArray ∗) emsrGetSupportedEncryptions: (NSError ∗∗) *error*

Returns supported encryption algorhtms by the encrypted head.

Parameters

error	pointer to NSError object, where error information is stored in case function fails. You can pass nil if you don't want that information

Returns

an array of supported algorithms or nil if an error occured

2.11.3.13 - (BOOL) emsrIsTampered: (BOOL $*$) *tampered* error:(NSError $**$) *error*

Checks if the head was tampered or not.

If the head's tamper protection have activated, the device should be sent to service for checks

Returns

true if the head was tampered and not operational

2.11.3.14 - (BOOL) emsrLoadInitialKey: (NSData $*$) *keyData* error:(NSError $**$) *error*

Loads Terminal Master Key (TMK) or reenable after tampering.

This command is enabled only if the device is in tamper mode or there is no TMK key yet. If the command is executed in normal mode an error will be returned. To reenable the device after tampering the old TMK key must be passed as an argument. If the keys do not match error will be returned.

Parameters

keyData	an array, that consists of: • BLOCK IDENT - 1 byte, set to 0x29 • KEY ID - the ID of the key to set, put KEY_TMK_AES (0x10) • KEY VERSION - the version of the key in high to low order, 4 bytes, cannot be 0 • KEY - the key data, 16 bytes • HASH - SHA256 of the previous bytes (BLOCK IDENT, KEY ID, KEY VERSION and KEY)
error	pointer to NSError object, where error information is stored in case function fails. You can pass nil if you don't want that information

Returns

TRUE if function succeeded, FALSE otherwise

2.11.3.15 - (BOOL) emsrLoadKey: (NSData $*$) *keyData* error:(NSError $**$) *error*

Loads new key, in plain or encrypted with already loaded AES256 Key Encryption Key (KEK).

Plain text loading works only the first time the specified key is loaded and is recommended only in secure environment. For normal usage the new key should be encrypted with the Key Encryption Key (KEK). The command is unavailable if the device is tampred.

Parameters

keyData	an array, that consists of: • MAGIC NUMBER - (1 byte) 0x2b • ENCRYPTION KEY ID - (1 byte) the ID of the already existing key, used to encrypt the new key data. Set it to KEY_EH_AES256_LOADING (0x02) if you want to set the key in encrypted state or 0xFF for plain state. • KEY ID - (1 byte) the ID of the key to set, one of the KEY_ constants. The TMK cannot be changed with this command. • KEY VERSION - (4 bytes) the version of the key in high to low order, 4 bytes, cannot be 0 • KEY - (variable) the key data, length depends on the key in question, AES256 keys are 32 bytes, DUKPT key is 16 bytes key, 10 bytes serial, 6 bytes for padding (zeroes) • HASH - SHA256 of the previous bytes (MAGIC NUMBER, ENCRYPTION KEY ID, KEY ID, KEY VERSION, KEY)

If using KEY_EH_AES256_LOADING, then KEY + HASH have to be put inside the packet encrypted with AES256 using key KEY_EH_AES256_LOADING. SHA256 is calculated on the unencrypted data. The head decrypts the data and then calculates and compares the hash. If the calculated SHA does not match the SHA sent with the command, the key excahnge is rejected and error is returned.

Parameters

error	pointer to NSError object, where error information is stored in case function fails. You can pass nil if you don't want that information

Returns

TRUE if function succeeded, FALSE otherwise

2.11.3.16 - (BOOL) emsrSetActiveHead: (int) *active* error:(NSError ∗∗) *error*

In case there are multiple encrypted heads on the device, sets the active one.

Currently second head, emulated, is present in EMV NFC Lineas only.

Parameters

active	the encrypted head to use with all other emsr functions - either EMSR_REAL or EMSR_EMUL
error	pointer to NSError object, where error information is stored in case function fails. You can pass nil if you don't want that information

Returns

TRUE if function succeeded, FALSE otherwise

2.11.3.17 - (BOOL) emsrSetEncryption: (int) *encryption* keyID:(int) *keyID* params:(NSDictionary ∗) *params* error:(NSError ∗∗) *error*

Selects the prefered encryption algorithm.

When card is swiped, it will be encrypted by it and sent via magneticCardEncryptedData delegate

Parameters

encryption	encryption algorhtm used, one o fthe ALG_∗ constants
keyID	the ID of the key to use, one of the KEY_∗ constants. The key needs to be suitable for the provided algorithm.
params	optional algorithm parameters, currently no algorithm supports these
error	pointer to NSError object, where error information is stored in case function fails. You can pass nil if you don't want that information

Returns

TRUE if function succeeded, FALSE otherwise

2.11.3.18 - (BOOL) emsrSetEncryption: (int) *encryption* params:(NSDictionary ∗) *params* error:(NSError ∗∗) *error*

Selects the prefered encryption algorithm.

When card is swiped, it will be encrypted by it and sent via magneticCardEncryptedData delegate

Parameters

encryption	encryption algorhtm used, one o fthe ALG_∗ constants
params	optional algorithm parameters, currently no algorithm supports these
error	pointer to NSError object, where error information is stored in case function fails. You can pass nil if you don't want that information

Returns

TRUE if function succeeded, FALSE otherwise

2.11.3.19 - (BOOL) emsrUpdateFirmware: (NSData ∗) *data* error:(NSError ∗∗) *error*

Performs firmware update on the encrypted head.

DO NOT INTERRUPT THE COMMUNICATION DURING THE FIRMWARE UPDATE!

Parameters

data	firmware file data
error	pointer to NSError object, where error information is stored in case function fails. You can pass nil if you don't want that information

Returns

TRUE if function succeeded, FALSE otherwise

2.12 Voltage Functions

Functions to work with voltage card endcryption.

Functions

- (DTVoltageInfo ∗) - DTDevices::voltageGetInfo:

  Returns various information about Voltage state.
- (BOOL) - DTDevices::voltageLoadConfiguration:error:

  Loads new configuration.
- (BOOL) - DTDevices::voltageGenerateNewKey:

  Forces generation of a new key.

2.12.1 Detailed Description

Functions to work with voltage card endcryption.

2.12.2 Function Documentation

2.12.2.1 - (BOOL) voltageGenerateNewKey: (NSError ∗∗) *error*

Forces generation of a new key.

This is asynchronous process, you can query the current state of key generation via voltageGetInfo

Parameters

error	pointer to NSError object, where error information is stored in case function fails. You can pass nil if you don't want that information

Returns

TRUE if function succeeded, FALSE otherwise

2.12.2.2 - (DTVoltageInfo ∗) voltageGetInfo: (NSError ∗∗) *error*

Returns various information about Voltage state.

Parameters

error	pointer to NSError object, where error information is stored in case function fails. You can pass nil if you don't want that information

Returns

DTVoltageInfo class if function succeeded, nil otherwise

2.12.2.3 - (BOOL) voltageLoadConfiguration: (NSData ∗) *configuration* error:(NSError ∗∗) *error*

Loads new configuration.

The configuration is signed with RSA2048 to prevent unauthorized modification and contains all parameters, needed for the algorithm to work.

The format of the data is series of blocks, following the structure:

- length of the field (2 bytes, big endian)
- content of the field (variable, can be missing if the length is 0)

If any field is missing, its length is 0 and content - empty. The final block looks something like:

[RSA CHECKSUM][LenHi,LenLo][Field0][LenHi,LenLo][Field1][LenHi,LenLo][Field2]...

Based on the position, the fields are as follows:

0	Required	Configuration version, 4 bytes in big endian format, can be read with voltageGetInfo
1	Required	Identity string, variable length, it is used during ETB generation
2	Required	Public parameters, variable length binary data block, used during ETB generation
3	Required	Encryption type, 1 byte, either 0 for full track encryption or 1 for structure preserving encryption (see below)
4	Optional	Merchant ID string, variable length, zero sized means no MID will be present in the packet (see below)
5	Optional	Key rollover days, 4 bytes in big endian format, 0 or zero-sized length disables the feature (see below)
6	Optional	Key rollover number of transactions, 4 bytes in big endian format, 0 or zero-sized length disables the feature (see below)

Encryption type, it can be either 0 (FULL, whole track) or 1 (SPE, structure preserving). They both differ on the way track 1 & 2 data and PAN is encrypted, there is no difference in track 3 and merchant ID.

An example of PAN data encryption using both methods:

- PAN : 5105105105105100
- FULL: 5105102433775100
- SPE : +++++++X0oDMHFSj

An example of track 1 data encryption using both methods:

- Track1: B5105105105105100^840PUBLIC/JOHN Q^120422212345?
- FULL : B5105103065100^840PUBLIC/JOHN Q^1204222kzKsspG8?
- SPE : 9o6OY2VmftqV69ZoYqxZ0cusnnDr1oQtiTIVGDaIQrnbSrHql

An example of track 2 data encryption using both methods:

- Track2: 5105105105105100=120422212345
- FULL : 5105103065100=1204222kzKsspG8
- SPE : 3Ep5uEIE1Ov7JEEM1IBRGjgQKGT

For PAN data, VOLTAGE_ENCRYPTION_SPE guarantees the following:

- The leading 6 digits of the original PAN are maintained in the clear.
- The trailing 4 digits of the original PAN are maintained in the clear.
- The middle digits are used for the ciphertext value, which is guaranteed to consist solely of digits.
- The Luhn check value is preserved so that a PAN with a valid (0) result, creates ciphertext that also checks as valid. For non-PAN data, such as MIDs, VOLTAGE_ENCRYPTION_SPE behaves the same way as VOLTAGE_ENCRYPTION_FULL.

Merchant ID value data length can be from 4 to 23 digits long. The input and output character sets are identical and length is always preserved. The following is an example of a MID encryption:

- Plaintext: 8888881000000000
- Ciphertext: 1234433247352418

Key rollover conditions will will force a new key generation when any of them triggers. When that happns, all conditions are reset to their start values, i.e. if you have set key rollover to 10 days or 100 transactions, if the 100 transactions happen first, then the day count is reset to 0.

Generating key is a long process (about 2 minutes), so if a card is swiped during the process, the device will use current key to encrypt and restart generation process. You can query the current state of key generation via voltageGetInfo.

Key rollover days - after the specified number of days elapsed, the key will be regenerated. Using value of 0, or not setting this field at all (length of 0) disables it. Key rollover will happen automatically after an year in any case. The actual time of key generation will differ each time by a random amount, i.e. if you set days to 1 and generated on 7am today, the next generation can happen at 6pm on next day. This is by design in order to disperse key generation requests to lower server load.

Key rollover number of transactions - after specified number of cards read, the key will be regenerated. Using value of 0, or not setting this field at all (length of 0) disables it.

Note

After changing configuration, be sure to call voltageGenerateNewKey in order for the new settings to take effect! The preferred way of handling the configuration is to send us the configuration values wanted (encryption type, public parameters, identity string, key rollover conditions - days/number of transactions) and we will return the required block. Then the program just checks the current parameters version and if different - reloads them.

Parameters

configuration	voltage configuration data
error	pointer to NSError object, where error information is stored in case function fails. You can pass nil if you don't want that information

Returns

TRUE if function succeeded, FALSE otherwise

2.13 RF Reader Functions

Functions to work with the RF cards reader.

Macros

- #define CARD_SUPPORT_TYPE_A 0x0001

  ISO14443 Type A (Mifare) cards will be detected.
- #define CARD_SUPPORT_TYPE_B 0x0002

  ISO14443 Type B cards will be detected.
- #define CARD_SUPPORT_FELICA 0x0004

  Felica cards will be detected.
- #define CARD_SUPPORT_NFC 0x0008

  NFC cards will be detected.
- #define CARD_SUPPORT_JEWEL 0x0010

  Jewel cards will be detected.
- #define CARD_SUPPORT_ISO15 0x0020

  ISO15693 cards will be detected.
- #define CARD_SUPPORT_STSRI 0x0040

  ST SRI cards will be detected.
- #define CARD_SUPPORT_PICOPASS_ISO14 0x0080

  PicoPass ISO14443-A.
- #define CARD_SUPPORT_PICOPASS_ISO15 0x0100

  PicoPass ISO15693.

Functions

- (BOOL) - DTDevices::rfInit:error:

  Initializes and powers on the RF card reader module.
- (BOOL) - DTDevices::rfClose:

  Powers down RF card reader module.
- (BOOL) - DTDevices::rfRemoveCard:error:

  Call this function once you are done with the card, a delegate call rfCardRemoved will be called when the card leaves the RF field and new card is ready to be detected.
- (BOOL) - DTDevices::mfAuthByKey:type:address:key:error:

  Authenticate mifare card block with direct key data.
- (BOOL) - DTDevices::mfStoreKeyIndex:type:key:error:

  Store key in the internal module memory for later use.
- (BOOL) - DTDevices::mfAuthByStoredKey:type:address:keyIndex:error:

  Authenticate mifare card block with previously stored key.
- (NSData $*$) - DTDevices::mfRead:address:length:error:

  Reads one more more blocks of data from Mifare Classic/Ultralight cards.
- (int) - DTDevices::mfWrite:address:data:error:

  Writes one more more blocks of data to Mifare Classic/Ultralight cards.
- (BOOL) - DTDevices::mfUlcSetKey:key:error:

  Sets the 3DES key of Mifare Ultralight C cards.
- (BOOL) - DTDevices::mfUlcAuthByKey:key:error:

  Performs 3DES authentication of Mifare Ultralight C card using the given key.
- (NSData $*$) - DTDevices::iso15693Read:startBlock:length:error:

  Reads one more more blocks of data from ISO 15693 card.

- (int) - DTDevices::iso15693Write:startBlock:data:error:

  Writes one more more blocks of data to ISO 15693 card.
- (NSData ∗) - DTDevices::iso15693GetBlocksSecurityStatus:startBlock:nBlocks:error:

  Reads the security status of one more more blocks from ISO 15693 card.
- (BOOL) - DTDevices::iso15693LockBlock:block:error:

  Locks a single ISO 15693 card block.
- (BOOL) - DTDevices::iso15693WriteAFI:afi:error:

  Changes ISO 15693 card AFI.
- (BOOL) - DTDevices::iso15693LockAFI:error:

  Locks ISO 15693 AFI preventing further changes.
- (BOOL) - DTDevices::iso15693WriteDSFID:dsfid:error:

  Changes ISO 15693 card DSFID.
- (BOOL) - DTDevices::iso15693LockDSFID:error:

  Locks ISO 15693 card DSFID preventing further changes.
- (NSData ∗) - DTDevices::iso14GetATS:error:

  Initializes ISO1443B card and returns Answer To Select.
- (NSData ∗) - DTDevices::iso14APDU:cla:ins:p1:p2:data:apduResult:error:

  Executes APDU command on ISO1443B compatible card.
- (BOOL) - DTDevices::felicaSetPollingParamsRequestCode:systemCode:error:

  Sets polling parameters of FeliCa card.
- (NSData ∗) - **DTDevices::felicaSendCommand:command:data:error:**
- (NSData ∗) - DTDevices::felicaRead:serviceCode:startBlock:length:error:

  Reads one more more blocks of data from FeliCa card.
- (int) - DTDevices::felicaWrite:serviceCode:startBlock:data:error:

  Writes one more more blocks of data to FeliCa card.
- (BOOL) - DTDevices::felicaSmartTagGetBatteryStatus:status:error:

  Returns FeliCa SmartTag battery status.
- (BOOL) - DTDevices::felicaSmartTagClearScreen:error:

  Clears the screen of FeliCa SmartTag.
- (BOOL) - DTDevices::felicaSmartTagDrawImage:image:topLeftX:topLeftY:drawMode:layout:error:

  Draws image on FeliCa SmartTag's screen.
- (BOOL) - DTDevices::felicaSmartTagSaveLayout:layout:error:

  Saves the current display as layout number.
- (BOOL) - DTDevices::felicaSmartTagDisplayLayout:layout:error:

  Displays previously stored layout.
- (int) - DTDevices::felicaSmartTagWrite:address:data:error:

  Writes data in FeliCa SmartTag.
- (NSData ∗) - DTDevices::felicaSmartTagRead:address:length:error:

  Writes data in FeliCa SmartTag.
- (BOOL) - DTDevices::felicaSmartTagWaitCompletion:error:

  Waits for FeliCa SmartTag to complete current operation.
- (NSData ∗) - DTDevices::stSRIRead:address:length:error:

  Reads one more more blocks of data from ST SRI card.
- (int) - DTDevices::stSRIWrite:address:data:error:

  Writes one more more blocks of data to ST SRI card.
- (NSData ∗) - **DTDevices::hidGetVersionInfo:**
- (NSData ∗) - **DTDevices::hidGetSerialNumber:**
- (NSData ∗) - **DTDevices::hidGetContentElement:pin:rootSoOID:error:**

2.13.1 Detailed Description

Functions to work with the RF cards reader.

2.13.2 Macro Definition Documentation

2.13.2.1 #define CARD_SUPPORT_JEWEL 0x0010

Jewel cards will be detected.

Currently unsupported.

2.13.2.2 #define CARD_SUPPORT_NFC 0x0008

NFC cards will be detected.

Currently unsupported.

2.13.2.3 #define CARD_SUPPORT_TYPE_B 0x0002

ISO14443 Type B cards will be detected.

Currently unsupported.

2.13.3 Function Documentation

2.13.3.1 - (NSData ∗) felicaRead: (int) *cardIndex* serviceCode:(int) *serviceCode* startBlock:(int) *startBlock* length:(int) *length* error:(NSError ∗∗) *error*

Reads one more more blocks of data from FeliCa card.

Parameters

cardIndex	the index of the card as sent by rfCardDetected delegate call
serviceCode	the service code, default is 0x0900
startBlock	the starting block to read from
length	the number of bytes to read, this must be multiple of block size (can be taken from the card info that is coming with rfCardDetected call)
error	pointer to NSError object, where error information is stored in case function fails. You can pass nil if you don't want that information

Returns

NSData object containing the data received or nil if an error occured

2.13.3.2 - (BOOL) felicaSetPollingParamsRequestCode: (int) *requestCode* systemCode:(int) *systemCode* error:(NSError ∗∗) *error*

Sets polling parameters of FeliCa card.

Call this function before rfInit!

Parameters

requestCode	request code, refer to FeliCa documentation, default is 1
systemCode	system code, refer to FeliCa documentation, default is 0xFFFF
error	pointer to NSError object, where error information is stored in case function fails. You can pass nil if you don't want that information

Returns

TRUE if function succeeded, FALSE otherwise

2.13.3.3 - (BOOL) felicaSmartTagClearScreen: (int) *cardIndex* error:(NSError ∗∗) *error*

Clears the screen of FeliCa SmartTag.

Parameters

cardIndex	the index of the card as sent by rfCardDetected delegate call
status	upon successful execution, battery status will be returned here, one of FELICA_SMARTTAG-_BATTERY_∗ constants
error	pointer to NSError object, where error information is stored in case function fails. You can pass nil if you don't want that information

Returns

TRUE if function succeeded, FALSE otherwise

2.13.3.4 - (BOOL) felicaSmartTagDisplayLayout: (int) *cardIndex* layout:(int) *layout* error:(NSError ∗∗) *error*

Displays previously stored layout.

Parameters

cardIndex	the index of the card as sent by rfCardDetected delegate call
layout	layout index (1-12) of the previously stored image
error	pointer to NSError object, where error information is stored in case function fails. You can pass nil if you don't want that information

Returns

TRUE if function succeeded, FALSE otherwise

2.13.3.5 - (BOOL) felicaSmartTagDrawImage: (int) *cardIndex* image:(UIImage ∗) *image* topLeftX:(int) *topLeftX* topLeftY:(int) *topLeftY* drawMode:(int) *drawMode* layout:(int) *layout* error:(NSError ∗∗) *error*

Draws image on FeliCa SmartTag's screen.

The screen is 200x96 pixels.

Parameters

cardIndex	the index of the card as sent by rfCardDetected delegate call
image	image to draw
topLeftX	- topleft X coordinate in pixels
topLeftY	- topleft Y coordinate in pixels
drawMode	draw mode, one of the FELICA_SMARTTAG_DRAW_∗ constants
layout	only used when drawMode is FELICA_SMARTTAG_DRAW_USE_LAYOUT, it specifies the index of the layout (1-12) of the previously stored image
error	pointer to NSError object, where error information is stored in case function fails. You can pass nil if you don't want that information

Returns

TRUE if function succeeded, FALSE otherwise

2.13.3.6 - (BOOL) felicaSmartTagGetBatteryStatus: (int) *cardIndex* status:(int ∗) *status* error:(NSError ∗∗) *error*

Returns FeliCa SmartTag battery status.

Note

Call this function before any other SmartTag

Parameters

cardIndex	the index of the card as sent by rfCardDetected delegate call
status	upon successful execution, battery status will be returned here, one of FELICA_SMARTTAG-_BATTERY_∗ constants
error	pointer to NSError object, where error information is stored in case function fails. You can pass nil if you don't want that information

Returns

TRUE if function succeeded, FALSE otherwise

2.13.3.7 - (NSData ∗) felicaSmartTagRead: (int) *cardIndex* address:(int) *address* length:(int) *length* error:(NSError ∗∗) *error*

Writes data in FeliCa SmartTag.

Parameters

cardIndex	the index of the card as sent by rfCardDetected delegate call
address	the address of the card to read from, refer to SmartTag documentation
length	of the data to read, note that the data does not need to be aligned to block size
error	pointer to NSError object, where error information is stored in case function fails. You can pass nil if you don't want that information

Returns

NSData object containing the data received or nil if an error occured

2.13.3.8 - (BOOL) felicaSmartTagSaveLayout: (int) *cardIndex* layout:(int) *layout* error:(NSError ∗∗) *error*

Saves the current display as layout number.

Parameters

cardIndex	the index of the card as sent by rfCardDetected delegate call
layout	layout index (1-12) to which the currently displayed image will be saved
error	pointer to NSError object, where error information is stored in case function fails. You can pass nil if you don't want that information

Returns

TRUE if function succeeded, FALSE otherwise

2.13.3.9 - (BOOL) felicaSmartTagWaitCompletion: (int) *cardIndex* error:(NSError ∗∗) *error*

Waits for FeliCa SmartTag to complete current operation.

Waiting is generally not needed, but needed in case for example drawing an image and then saving the layout, you need to wait for the image to be drawn. Write operation forces waiting internally.

Parameters

cardIndex	the index of the card as sent by rfCardDetected delegate call
error	pointer to NSError object, where error information is stored in case function fails. You can pass nil if you don't want that information

Returns

TRUE if function succeeded, FALSE otherwise

2.13.3.10 - (int) felicaSmartTagWrite: (int) *cardIndex* address:(int) *address* data:(NSData ∗) *data* error:(NSError ∗∗) *error*

Writes data in FeliCa SmartTag.

Parameters

cardIndex	the index of the card as sent by rfCardDetected delegate call
address	the address of the card to write to, refer to SmartTag documentation
data	data to write, note that the data does not need to be aligned to block size
error	pointer to NSError object, where error information is stored in case function fails. You can pass nil if you don't want that information

Returns

number of bytes actually written or 0 if an error occured

2.13.3.11 - (int) felicaWrite: (int) *cardIndex* serviceCode:(int) *serviceCode* startBlock:(int) *startBlock* data:(NSData ∗) *data* error:(NSError ∗∗) *error*

Writes one more more blocks of data to FeliCa card.

Parameters

cardIndex	the index of the card as sent by rfCardDetected delegate call
serviceCode	the service code, default is 0x0900
startBlock	the starting block to write to
data	the data to write, it must be multiple of block size (can be taken from the card info that is coming with rfCardDetected call)
error	pointer to NSError object, where error information is stored in case function fails. You can pass nil if you don't want that information

Returns

number of bytes actually written or 0 if an error occured

2.13.3.12 - (NSData $*$) iso14APDU: (int) *cardIndex* cla:(uint8_t) *cla* ins:(uint8_t) *ins* p1:(uint8_t) *p1* p2:(uint8_t) *p2* data:(NSData $*$) *data* apduResult:(uint16_t $*$) *apduResult* error:(NSError $**$) *error*

Executes APDU command on ISO1443B compatible card.

The card must be initialized with iso14GetATS first

Parameters

cardIndex	the index of the card as sent by rfCardDetected delegate call
cla	CLA parameter, refer to card documentation
ins	INS parameter, refer to card documentation
p1	P1 parameter, refer to card documentation
p2	P2 parameter, refer to card documentation
data	optional data with the command
apduResult	every APDU command sends 2 bytes result code, refer to card documentation
error	pointer to NSError object, where error information is stored in case function fails. You can pass nil if you don't want that information

Returns

APDU response data or empty object, or nil if command failed

2.13.3.13 - (NSData $*$) iso14GetATS: (int) *cardIndex* error:(NSError $**$) *error*

Initializes ISO1443B card and returns Answer To Select.

Call this function before further communication with the card.

Parameters

cardIndex	the index of the card as sent by rfCardDetected delegate call
error	pointer to NSError object, where error information is stored in case function fails. You can pass nil if you don't want that information

Returns

Answer To Select data, or nil if command failed

2.13.3.14 - (NSData $*$) iso15693GetBlocksSecurityStatus: (int) *cardIndex* startBlock:(int) *startBlock* nBlocks:(int) *nBlocks* error:(NSError $**$) *error*

Reads the security status of one more more blocks from ISO 15693 card.

Parameters

cardIndex	the index of the card as sent by rfCardDetected delegate call
startBlock	the starting block to read from
nBlocks	the number of blocks to get the security status
error	pointer to NSError object, where error information is stored in case function fails. You can pass nil if you don't want that information

Returns

NSData object containing the data received or nil if an error occured

2.13.3.15 - (BOOL) iso15693LockAFI: (int) *cardIndex* error:(NSError ∗∗) *error*

Locks ISO 15693 AFI preventing further changes.

Parameters

cardIndex	the index of the card as sent by rfCardDetected delegate call
error	pointer to NSError object, where error information is stored in case function fails. You can pass nil if you don't want that information

Returns

TRUE if function succeeded, FALSE otherwise

2.13.3.16 - (BOOL) iso15693LockBlock: (int) *cardIndex* block:(int) *block* error:(NSError ∗∗) *error*

Locks a single ISO 15693 card block.

Locked blocks cannot be written upon anymore.

Parameters

cardIndex	the index of the card as sent by rfCardDetected delegate call
block	the block index to lock
error	pointer to NSError object, where error information is stored in case function fails. You can pass nil if you don't want that information

Returns

TRUE if function succeeded, FALSE otherwise

2.13.3.17 - (BOOL) iso15693LockDSFID: (int) *cardIndex* error:(NSError ∗∗) *error*

Locks ISO 15693 card DSFID preventing further changes.

Parameters

cardIndex	the index of the card as sent by rfCardDetected delegate call
error	pointer to NSError object, where error information is stored in case function fails. You can pass nil if you don't want that information

Returns

TRUE if function succeeded, FALSE otherwise

2.13.3.18 - (NSData ∗) iso15693Read: (int) *cardIndex* startBlock:(int) *startBlock* length:(int) *length* error:(NSError ∗∗) *error*

Reads one more more blocks of data from ISO 15693 card.

Parameters

cardIndex	the index of the card as sent by rfCardDetected delegate call
startBlock	the starting block to read from
length	the number of bytes to read, this must be multiple of block size (can be taken from the card info that is coming with rfCardDetected call)
error	pointer to NSError object, where error information is stored in case function fails. You can pass nil if you don't want that information

Returns

NSData object containing the data received or nil if an error occured

2.13.3.19 - (int) iso15693Write: (int) *cardIndex* startBlock:(int) *startBlock* data:(NSData ∗) *data* error:(NSError ∗∗) *error*

Writes one more more blocks of data to ISO 15693 card.

Parameters

cardIndex	the index of the card as sent by rfCardDetected delegate call
startBlock	the starting block to write to
data	the data to write, it must be multiple of block size (can be taken from the card info that is coming with rfCardDetected call)
error	pointer to NSError object, where error information is stored in case function fails. You can pass nil if you don't want that information

Returns

number of bytes actually written or 0 if an error occured

2.13.3.20 - (BOOL) iso15693WriteAFI: (int) *cardIndex* afi:(uint8_t) *afi* error:(NSError ∗∗) *error*

Changes ISO 15693 card AFI.

Parameters

cardIndex	the index of the card as sent by rfCardDetected delegate call
afi	new AFI value
error	pointer to NSError object, where error information is stored in case function fails. You can pass nil if you don't want that information

Returns

TRUE if function succeeded, FALSE otherwise

2.13.3.21 - (BOOL) iso15693WriteDSFID: (int) *cardIndex* dsfid:(uint8_t) *dsfid* error:(NSError ∗∗) *error*

Changes ISO 15693 card DSFID.

Parameters

cardIndex	the index of the card as sent by rfCardDetected delegate call
dsfid	new DSFID value
error	pointer to NSError object, where error information is stored in case function fails. You can pass nil if you don't want that information

Returns

TRUE if function succeeded, FALSE otherwise

2.13.3.22 - (BOOL) mfAuthByKey: (int) *cardIndex* type:(char) *type* address:(int) *address* key:(NSData ∗) *key* error:(NSError ∗∗) *error*

Authenticate mifare card block with direct key data.

This is less secure method, as it requires the key to be present in the program, the prefered way is to store a key once in a secure environment and then authenticate using the stored key.

Parameters

cardIndex	the index of the card as sent by rfCardDetected delegate call
type	key type, either 'A' or 'B'
address	the address of the block to authenticate
key	6 bytes key
error	pointer to NSError object, where error information is stored in case function fails. You can pass nil if you don't want that information

Returns

TRUE if function succeeded, FALSE otherwise

2.13.3.23 - (BOOL) mfAuthByStoredKey: (int) *cardIndex* type:(char) *type* address:(int) *address* keyIndex:(int) *keyIndex* error:(NSError ∗∗) *error*

Authenticate mifare card block with previously stored key.

This the prefered method, as no key needs to reside in application.

Parameters

cardIndex	the index of the card as sent by rfCardDetected delegate call
type	key type, either 'A' or 'B'
address	the address of the block to authenticate
keyIndex	the index of the stored key, you can have up to 8 keys stored (0-7)
error	pointer to NSError object, where error information is stored in case function fails. You can pass nil if you don't want that information

Returns

TRUE if function succeeded, FALSE otherwise

2.13.3.24 - (NSData ∗) mfRead: (int) *cardIndex* address:(int) *address* length:(int) *length* error:(NSError ∗∗) *error*

Reads one more more blocks of data from Mifare Classic/Ultralight cards.

A single read operation gets 16 bytes of data, so you can pass 32 on length to read 2 blocks, etc

Parameters

cardIndex	the index of the card as sent by rfCardDetected delegate call
address	the address of the block to read
length	the number of bytes to read, this must be multiple of block size (16 bytes)
error	pointer to NSError object, where error information is stored in case function fails. You can pass nil if you don't want that information

Returns

NSData object containing the data received or nil if an error occured

2.13.3.25 - (BOOL) mfStoreKeyIndex: (int) *keyIndex* type:(char) *type* key:(NSData $*$) *key* error:(NSError $**$) *error*

Store key in the internal module memory for later use.

Parameters

keyIndex	the index of the key, you can have up to 8 keys stored (0-7)
type	key type, either 'A' or 'B'
key	6 bytes key
error	pointer to NSError object, where error information is stored in case function fails. You can pass nil if you don't want that information

Returns

TRUE if function succeeded, FALSE otherwise

2.13.3.26 - (BOOL) mfUlcAuthByKey: (int) *cardIndex* key:(NSData $*$) *key* error:(NSError $**$) *error*

Performs 3DES authentication of Mifare Ultralight C card using the given key.

Parameters

cardIndex	the index of the card as sent by rfCardDetected delegate call
key	16 bytes 3DES key to authenticate with
error	pointer to NSError object, where error information is stored in case function fails. You can pass nil if you don't want that information

Returns

TRUE if function succeeded, FALSE otherwise

2.13.3.27 - (BOOL) mfUlcSetKey: (int) *cardIndex* key:(NSData $*$) *key* error:(NSError $**$) *error*

Sets the 3DES key of Mifare Ultralight C cards.

Parameters

cardIndex	the index of the card as sent by rfCardDetected delegate call
key	16 bytes 3DES key to set
error	pointer to NSError object, where error information is stored in case function fails. You can pass nil if you don't want that information

Returns

TRUE if function succeeded, FALSE otherwise

2.13.3.28 - (int) mfWrite: (int) *cardIndex* address:(int) *address* data:(NSData $*$) *data* error:(NSError $**$) *error*

Writes one more more blocks of data to Mifare Classic/Ultralight cards.

A single write operation stores 16 bytes of data, so you can pass 32 on length to write 2 blocks, etc

Parameters

cardIndex	the index of the card as sent by rfCardDetected delegate call
address	the address of the block to write
data	the data to write, must be multiple of the block size (16 bytes)
error	pointer to NSError object, where error information is stored in case function fails. You can pass nil if you don't want that information

Returns

number of bytes actually written or 0 if an error occured

2.13.3.29 - (BOOL) rfClose: (NSError ∗∗) *error*

Powers down RF card reader module.

Call this function after you are done with the reader.

Parameters

error	pointer to NSError object, where error information is stored in case function fails. You can pass nil if you don't want that information

Returns

TRUE if function succeeded, FALSE otherwise

2.13.3.30 - (BOOL) rfInit: (int) *supportedCards* error:(NSError ∗∗) *error*

Initializes and powers on the RF card reader module.

Call this function before any other RF card functions. The module power consumption is highly optimized, so it can be left on for extended periods of time.

Parameters

supportedCards	any combination of CARD_SUPPORT_∗ flags to mark which card types to be active. Enable only cards you actually plan to work with, this has high implication on power usage and detection speed.
error	pointer to NSError object, where error information is stored in case function fails. You can pass nil if you don't want that information

Returns

TRUE if function succeeded, FALSE otherwise

2.13.3.31 - (BOOL) rfRemoveCard: (int) *cardIndex* error:(NSError ∗∗) *error*

Call this function once you are done with the card, a delegate call rfCardRemoved will be called when the card leaves the RF field and new card is ready to be detected.

Parameters

cardIndex	the index of the card as sent by rfCardDetected delegate call
error	pointer to NSError object, where error information is stored in case function fails. You can pass nil if you don't want that information

Returns

TRUE if function succeeded, FALSE otherwise

2.13.3.32 - (NSData ∗) stSRIRead: (int) *cardIndex* address:(int) *address* length:(int) *length* error:(NSError ∗∗) *error*

Reads one more more blocks of data from ST SRI card.

Parameters

cardIndex	the index of the card as sent by rfCardDetected delegate call
address	the starting block to read from
length	the number of bytes to read, this must be multiple of block size (can be taken from the card info that is coming with rfCardDetected call)
error	pointer to NSError object, where error information is stored in case function fails. You can pass nil if you don't want that information

Returns

NSData object containing the data received or nil if an error occured

2.13.3.33 - (int) stSRIWrite: (int) *cardIndex* address:(int) *address* data:(NSData ∗) *data* error:(NSError ∗∗) *error*

Writes one more more blocks of data to ST SRI card.

Parameters

cardIndex	the index of the card as sent by rfCardDetected delegate call
address	the starting block to write to
data	the data to write, it must be multiple of block size (can be taken from the card info that is coming with rfCardDetected call)
error	pointer to NSError object, where error information is stored in case function fails. You can pass nil if you don't want that information

Returns

number of bytes actually written or 0 if an error occured

2.14 SmartCard Functions

This section includes functions to access SmartCard module and operate with SmartCards.

Functions

- (BOOL) - DTDevices::scInit:error:

    Initializes SmartCard module.
- (NSData ∗) - DTDevices::scCardPowerOn:error:

    Powers on the SmartCard, resets it and returns ATR (Answer To Reset).
- (BOOL) - DTDevices::scCardPowerOff:error:

    Powers off SmartCard, call this function when you are done with the card.
- (BOOL) - DTDevices::scIsCardPresent:error:

    Manually checks if there is a card in the reader.
- (NSData ∗) - DTDevices::scCAPDU:apdu:error:

    Performs APDU command in the card.
- (BOOL) - DTDevices::scClose:error:

    Shuts down SmartCard module.

2.14.1 Detailed Description

This section includes functions to access SmartCard module and operate with SmartCards.

2.14.2 Function Documentation

2.14.2.1 - (NSData ∗) scCAPDU: (SC_SLOTS) *slot* apdu:(NSData ∗) *apdu* error:(NSError ∗∗) *error*

Performs APDU command in the card.

Parameters

slot	- which slot you want to operate with, one of: SLOT_MAIN — main SmartCard slot SLOT_SAM — SAM module slot
apdu	- the APDU command
error	pointer to NSError object, where error information is stored in case function fails. You can pass nil if you don't want that information

Returns

APDU response data if function succeeded, nil otherwise

2.14.2.2 - (BOOL) scCardPowerOff: (SC_SLOTS) *slot* error:(NSError ∗∗) *error*

Powers off SmartCard, call this function when you are done with the card.

Parameters

slot	- which slot you want to operate with, one of: SLOT_MAIN — main SmartCard slot SLOT_SAM — SAM module slot
error	pointer to NSError object, where error information is stored in case function fails. You can pass nil if you don't want that information

Returns

TRUE if function succeeded, FALSE otherwise

2.14.2.3 - (NSData ∗) scCardPowerOn: (SC_SLOTS) *slot* error:(NSError ∗∗) *error*

Powers on the SmartCard, resets it and returns ATR (Answer To Reset).

Call this function before you perform any APDU commands

Parameters

slot	- which slot you want to operate with, one of:	
	SLOT_MAIN	main SmartCard slot
	SLOT_SAM	SAM module slot
error	pointer to NSError object, where error information is stored in case function fails. You can pass nil if you don't want that information	

Returns

ATR response data if function succeeded or nil otherwise

2.14.2.4 - (BOOL) scClose: (SC_SLOTS) *slot* error:(NSError ∗∗) *error*

Shuts down SmartCard module.

Parameters

slot	- which slot you want to operate with, one of:	
	SLOT_MAIN	main SmartCard slot
	SLOT_SAM	SAM module slot
error	pointer to NSError object, where error information is stored in case function fails. You can pass nil if you don't want that information	

Returns

TRUE if function succeeded, FALSE otherwise

2.14.2.5 - (BOOL) scInit: (SC_SLOTS) *slot* error:(NSError ∗∗) *error*

Initializes SmartCard module.

Call this function before any other SmartCard related one. Without initialization, no SmartCard events will be fired.

Parameters

slot	- which slot you want to operate with, one of:	
	SLOT_MAIN	main SmartCard slot
	SLOT_SAM	SAM module slot
error	pointer to NSError object, where error information is stored in case function fails. You can pass nil if you don't want that information	

Returns

TRUE if function succeeded, FALSE otherwise

2.14.2.6 - (BOOL) scIsCardPresent: (SC_SLOTS) *slot* error:(NSError $**$) *error*

Manually checks if there is a card in the reader.

Parameters

slot	- which slot you want to operate with, one of:	
	SLOT_MAIN	main SmartCard slot
	SLOT_SAM	SAM module slot
error	pointer to NSError object, where error information is stored in case function fails. You can pass nil if you don't want that information	

Returns

TRUE if card is present, FALSE otherwise

2.15 Pinpad functions

Specific functions to work with the pinpad - entering and getting pin data, managing keys.

Functions

- (BOOL) - DTDevices::ppadStartPINEntry:startY:timeout:echoChar:message:error:

    Initiates asynchronous PIN entry procedure.
- (BOOL) - DTDevices::ppadCancelPINEntry:

    Tries to cancel asynchronous PIN entry procedure.
- (BOOL) - DTDevices::ppadMagneticCardEntry:timeout:error:

    Initiates synchronous magnetic card entry procedure.
- (NSData ∗) - DTDevices::ppadGetPINBlockUsingFixedKey:keyVariant:pinFormat:error:

    Gets encrypted pin data using pre-loaded 3DES key The returned data consists of:
- (NSData ∗) - DTDevices::pinGetPINBlockUsingDUKPT:keyVariant:pinFormat:error:

    Gets encrypted pin data using DUKPT.
- (DTKeyInfo ∗) - DTDevices::ppadGetKeyInfo:error:

    Gets information about some of the keys loaded in the pinpad.
- (NSData ∗) - **DTDevices::ppadGetDUKPTKeyKSN:error:**
- (NSData ∗) - DTDevices::ppadCryptoExchangeKeyID:kekID:usage:version:data:cbc:error:

    Loads/changes 3DES key into the pinpad.
- (NSData ∗) - DTDevices::ppadCryptoTR31ExchangeKeyID:kekID:tr31:error:

    Loads/changes 3DES key into the pinpad.
- (NSData ∗) - DTDevices::ppadCrypto3DESECBEncryptKeyID:inData:error:

    Encrypts a data on the pinpad using 3DES ECB.
- (NSData ∗) - DTDevices::ppadCrypto3DESECBDecryptKeyID:inData:error:

    Decrypts a data on the pinpad using 3DES ECB.
- (NSData ∗) - DTDevices::ppadCrypto3DESCBCEncryptKeyID:initVector:inData:error:

    Encrypts a data on the pinpad using 3DES CBC.
- (NSData ∗) - DTDevices::ppadCrypto3DESCBCDecryptKeyID:initVector:inData:error:

    Decrypts a data on the pinpad using 3DES CBC.
- (BOOL) - DTDevices::ppadSetButtonCaption:caption:error:

    Sets the text that is drawn above functional buttons in MPED400.
- (DTPinpadInfo ∗) - DTDevices::ppadGetSystemInfo:

    Returns pinpad specific information.
- (BOOL) - DTDevices::ppadKeyboardControl:error:

    Captures or releases keyboard.
- (BOOL) - DTDevices::ppadReadKey:error:

    Reads key from the pinpad.

2.15.1 Detailed Description

Specific functions to work with the pinpad - entering and getting pin data, managing keys.

2.15.2 Function Documentation

2.15.2.1 - (NSData ∗) pinGetPINBlockUsingDUKPT: (int) *dukptKeyID* keyVariant:(NSData ∗) *keyVariant* pinFormat:(int) *pinFormat* error:(NSError ∗∗) *error*

Gets encrypted pin data using DUKPT.

The returned data consists of:

- DUKPT/3DES Encrypted PIN code, according to the selected format (8 bytes)
- Current Key Serial Number (10 bytes)

Parameters

dukptKeyID	- DUKPT key ID (0-1)
keyVariant	16 bytes of data, that is XOR-ed with the key before encrypting. Pass nil if you don't want that.
pinFormat	PIN format, one of the PIN_FORMAT_∗ constants, according to ISO 9564
error	returns error information, you can pass nil if you don't want it

Returns

key information object upon success, nil otherwise

2.15.2.2 - (BOOL) ppadCancelPINEntry: (NSError ∗∗) *error*

Tries to cancel asynchronous PIN entry procedure.

Current pinpad versions do not have native support for async PIN, so this function always returns an error, but it will be implemented in the future.

Parameters

error	returns error information, you can pass nil if you don't want it

Returns

TRUE if function succeeded, FALSE otherwise

2.15.2.3 - (NSData ∗) ppadCrypto3DESCBCDecryptKeyID: (int) *keyID* initVector:(NSData ∗) *initVector* inData:(NSData ∗) *inData* error:(NSError ∗∗) *error*

Decrypts a data on the pinpad using 3DES CBC.

Parameters

keyID	the index of the 3DES key (1-49) to use for decryption
initVector	the initialization vector for the CBC, pass nil or 8 zeroes if you want to use empty IV
inData	the data to decrypt
error	returns error information, you can pass nil if you don't want it

Returns

decrypted data block

2.15.2.4 - (NSData ∗) ppadCrypto3DESCBCEncryptKeyID: (int) *keyID* initVector:(NSData ∗) *initVector* inData:(NSData ∗) *inData* error:(NSError ∗∗) *error*

Encrypts a data on the pinpad using 3DES CBC.

Parameters

keyID	the index of the 3DES key (1-49) to use for encryption
initVector	the initialization vector for the CBC, pass nil or 8 zeroes if you want to use empty IV
inData	the data to encrypt
error	returns error information, you can pass nil if you don't want it

Returns

encrypted data block

2.15.2.5 - (NSData ∗) ppadCrypto3DESECBDecryptKeyID: (int) *keyID* inData:(NSData ∗) *inData* error:(NSError ∗∗) *error*

Decrypts a data on the pinpad using 3DES ECB.

Parameters

keyID	the index of the 3DES key (1-49) to use for decryption
inData	the data to decrypt
error	returns error information, you can pass nil if you don't want it

Returns

decrypted data block

2.15.2.6 - (NSData ∗) ppadCrypto3DESECBEncryptKeyID: (int) *keyID* inData:(NSData ∗) *inData* error:(NSError ∗∗) *error*

Encrypts a data on the pinpad using 3DES ECB.

Parameters

keyID	the index of the 3DES key (1-49) to use for encryption
inData	the data to encrypt
error	returns error information, you can pass nil if you don't want it

Returns

encrypted data block

2.15.2.7 - (NSData ∗) ppadCryptoExchangeKeyID: (int) *keyID* kekID:(int) *kekID* usage:(int) *usage* version:(int) *version* data:(NSData ∗) *data* cbc:(BOOL) *cbc* error:(NSError ∗∗) *error*

Loads/changes 3DES key into the pinpad.

The key is encrypted via 3DES (ECB or CBC) by a Key Encryption Key already loaded. If KBPK type is used as KEK, then only other KEK (data encrypt, decrypt, pin) can be loadead, not the data key itself.

Parameters

keyID	- key the index where key shall be saved. For DUKPT keys this value can be between 0 and 1. For other keys the value can be between 1 and 49
kekID	- key the index of key, used to decrypt the encrypted key data when loading. The value can be between 0 and 49, where on index 0 resides the HMK key
usage	the key usage (type of key) attributes. See the KEY_USAGE_∗ constant field values.
version	the key version. Not used if key usage is KEY_USAGE_DUKPT
data	the 16 or 26 byte of input data to be processed. The first 16 bytes must contains encrypted key and next 10 bytes (if presents) are key serial number.
error	returns error information, you can pass nil if you don't want it

Returns

key check value upon success, nil otherwise

2.15.2.8 - (NSData ∗) ppadCryptoTR31ExchangeKeyID: (int) *keyID* kekID:(int) *kekID* tr31:(NSString ∗) *tr31* error:(NSError ∗∗) *error*

Loads/changes 3DES key into the pinpad.

The key is encrypted with TR31 by an already loaded KEK, KBPK or HMK If KBPK type is used as KEK, then all key types can be loaded.

Parameters

keyID	- the index where key shall be saved. For DUKPT keys this value can be between 0 and 1. For other keys the value can be between 1 and 49
kekID	- the index of the key, used to decrypt the encrypted key data when loading. The value can be between 0 and 49, where on index 0 resides the HMK key
tr31	the TR31 data block
error	returns error information, you can pass nil if you don't want it

Returns

key check value upon success, nil otherwise

2.15.2.9 - (DTKeyInfo ∗) ppadGetKeyInfo: (int) *keyID* error:(NSError ∗∗) *error*

Gets information about some of the keys loaded in the pinpad.

Parameters

keyID	- key ID (1-49)
error	returns error information, you can pass nil if you don't want it

Returns

key information object upon success, nil otherwise

2.15.2.10 - (NSData ∗) ppadGetPINBlockUsingFixedKey: (int) *fixedKeyID* keyVariant:(NSData ∗) *keyVariant* pinFormat:(int) *pinFormat* error:(NSError ∗∗) *error*

Gets encrypted pin data using pre-loaded 3DES key The returned data consists of:

- 3DES Encrypted PIN code, according to the selected format (8 bytes)

Parameters

fixedKeyID	- key ID (1-49)
keyVariant	16 bytes of data, that is XOR-ed with the key before encrypting. Pass nil if you don't want that.
pinFormat	PIN format, one of the PIN_FORMAT_∗ constants, according to ISO 9564
error	returns error information, you can pass nil if you don't want it

Returns

key information object upon success, nil otherwise

2.15.2.11 - (DTPinpadInfo ∗) ppadGetSystemInfo: (NSError ∗∗) *error*

Returns pinpad specific information.

Parameters

error	returns error information, you can pass nil if you don't want it

Returns

class containing pinpad information or nil if function failed

2.15.2.12 - (BOOL) ppadKeyboardControl: (BOOL) *capture* error:(NSError ∗∗) *error*

Captures or releases keyboard.

PinPad internally reads the keyboard to operate menus and such, if you want to be able to read keys, then you have to capture it before that, and release after.

Parameters

capture	- capture the keyboard if TRUE, release if FALSE
error	returns error information, you can pass nil if you don't want it

Returns

TRUE if function succeeded, FALSE otherwise

2.15.2.13 - (BOOL) ppadMagneticCardEntry: (int) *language* timeout:(int) *timeout* error:(NSError ∗∗) *error*

Initiates synchronous magnetic card entry procedure.

The magnetic card data is stored encrypted and protected inside the pinpad. After successful operation card data is sent like any other card read operation - via magneticCardEncryptedData with the encryption selected via emsr-SetEncryption. This function is blocking and can take up to timeout seconds, so make sure to call it on a thread or dispatch_async

Parameters

language	- the language to display promt on, one of the LANG_∗ constants
startY	- Y coordinate in characters from the top of the defined window where the PIN entry prompt will be drawn
timeout	- timeout in seconds waiting for the user to enter the card data (10-180)
error	returns error information, you can pass nil if you don't want it

Returns

TRUE if function succeeded, FALSE otherwise

2.15.2.14 - (BOOL) ppadReadKey: (char ∗) *key* error:(NSError ∗∗) *error*

Reads key from the pinpad.

y codes are:

0x00	No key have been pressed
0x01/0x03	Numeric key have been pressed, but no numeric mode is enabled
’0’-’9’	Number keys 0-9, available only in numeric mode
’A’	Accept button have been pressed
’C’	Cancel button have been pressed
’a’,’b’,’c’	Functional buttons 1-3

Parameters

key	- stores key upon return
error	returns error information, you can pass nil if you don’t want it

Returns

TRUE if function succeeded, FALSE otherwise

2.15.2.15 - (BOOL) ppadSetButtonCaption: (int) *buttonIndex* caption:(NSString ∗) *caption* error:(NSError ∗∗) *error*

Sets the text that is drawn above functional buttons in MPED400.

Parameters

buttonIndex	- functional button index (1-3)
caption	- text to display
error	returns error information, you can pass nil if you don’t want it

Returns

TRUE if function succeeded, FALSE otherwise

2.15.2.16 - (BOOL) ppadStartPINEntry: (int) *startX* startY:(int) *startY* timeout:(int) *timeout* echoChar:(char) *echoChar* message:(NSString ∗) *message* error:(NSError ∗∗) *error*

Initiates asynchronous PIN entry procedure.

The PIN is stored encrypted and protected inside the pinpad. This function is not blocking, it passes the answer via delegate. Currently this function calls internal synchronous function from a thread and notifies about the result, but future firmware versions will have native support where you can cancel pin entry too.

Parameters

startX	- X coordinate in characters from the left end of the defined window where the PIN entry prompt will be drawn
startY	- Y coordinate in characters from the top of the defined window where the PIN entry prompt will be drawn
timeout	- timeout in seconds waiting for the user to enter the pin (10-180)

echoChar	- symbol used to mark the pin digits, allowed are '∗', '+' or '-'
message	- text to be displayed to the user. You can use <CR> to move the cursor to the next line.
error	returns error information, you can pass nil if you don't want it

Returns

TRUE if function succeeded, FALSE otherwise

2.16 Certification Authority Functions

This section includes functions for managing CA keys.

Functions

- (BOOL) - DTDevices::caImportKeyNumber:RIDI:module:exponent:error:

  Import CA key.
- (BOOL) - DTDevices::caWriteKeysToFlash:

  Writes CA keys to flash.
- (NSArray ∗) - DTDevices::caGetKeysData:

  Returns keys data.
- (NSData ∗) - DTDevices::caImportIssuerKeyNumber:exponent:remainder:certificate:error:

  Import issuer key.
- (NSData ∗) - DTDevices::caImportICCKeyType:exponent:remainder:certificate:error:

  Import ICC key.
- (NSData ∗) - DTDevices::caRSAVerify:inData:error:

  RSA verify.

2.16.1 Detailed Description

This section includes functions for managing CA keys.

2.16.2 Function Documentation

2.16.2.1 - (NSArray ∗) caGetKeysData: (NSError ∗∗) *error*

Returns keys data.

Parameters

error	returns error information, you can pass nil if you don't want it

Returns

array of RFCAKeyData objects or nil if function failed

2.16.2.2 - (NSData ∗) caImportICCKeyType: (ICC_TYPES) *keyType* exponent:(NSData ∗) *exponent* remainder:(NSData ∗) *remainder* certificate:(NSData ∗) *certificate* error:(NSError ∗∗) *error*

Import ICC key.

Parameters

keyType	- key type, one of: TYPE_ICC — ICC TYPE_PIN — PIN
exponent	- exponent
remainder	- remainder
certificate	- certificate
error	returns error information, you can pass nil if you don't want it

Returns

decrypted certificate or nil if function failed

2.16.2.3 - (NSData ∗) caImportIssuerKeyNumber: (int) *keyNumber* exponent:(NSData ∗) *exponent* remainder:(NSData ∗) *remainder* certificate:(NSData ∗) *certificate* error:(NSError ∗∗) *error*

Import issuer key.

Parameters

keyNumber	- key number to decrypt issuer key (0-29)
exponent	- exponent
remainder	- remainder
certificate	- certificate
error	returns error information, you can pass nil if you don't want it

Returns

decrypted certificate or nil if function failed

2.16.2.4 - (BOOL) caImportKeyNumber: (int) *keyNumber* RIDI:(NSData ∗) *RIDI* module:(NSData ∗) *module* exponent:(NSData ∗) *exponent* error:(NSError ∗∗) *error*

Import CA key.

Parameters

keyNumber	- key number (0-29)
RIDI	- DIR+index (6 bytes)
module	- key module (32-248 bytes)
exponent	- key exponent (1-3 bytes typical)
error	returns error information, you can pass nil if you don't want it

Returns

TRUE if function succeeded, FALSE otherwise

2.16.2.5 - (NSData ∗) caRSAVerify: (RSA_VERIFY_KEY) *keyType* inData:(NSData ∗) *inData* error:(NSError ∗∗) *error*

RSA verify.

Parameters

keyType	- key type, one of: KEY_ISSUER — Issuer key; KEY_ICC — ICC key
inData	- input data
error	returns error information, you can pass nil if you don't want it

KEY_ISSUER	Issuer key
KEY_ICC	ICC key

Returns

output data or nil if function failed

2.16.2.6 - (BOOL) caWriteKeysToFlash: (NSError $**$) *error*

Writes CA keys to flash.

It is important to call this function after changing the keys if you don't want to lose them after device is turned off.

Parameters

error	returns error information, you can pass nil if you don't want it

Returns

TRUE if function succeeded, FALSE otherwise

2.17 Universal EMV2 Kernel

Universal EMV Level 2 kernel functions, structures and defnitions.

Universal EMV Level 2 kernel functions, structures and defnitions.

2.18 PinPad EMV Kernel

EMV Level 2 kernel functions, structures and defnitions.

Modules

- EMV Operation Workflow

    Description and block diagrams of various operations with the pinpad.
- EMV TAGs

    EMV TAGs you can use with their properties.
- EMV Status Codes

    These status codes are returned from every EMV function to indicate the result of it.
- Transaction Start

    This section includes the command used to start the transaction: ATR validation and application selection.
- Transaction Processing

    This section covers the different phases of the transaction:
    Initial process
    Data reading
    Card data authentication
    Restrictions processing
    Risk Control
    Cardholder authentication
    Certificate generation
    Make Transaction decision
    Make default decision.
- Issuer Authentication

    The commands listed here are intended to process the data coming from the issuer as part of the response to the online authorization request.
- General Commands

    These commands are not part of the basic transaction management but provide the kernel with more flexibility, and can be used by the application for its own particular requirements.
- Data Access

    The commands described below are used to access the data items used by the kernel.

Macros

- #define **EMV_STRUCTURES_DEFINED**
- #define TVR_DEFAULT_TDOL_USED 0x0508

    This is the list of the bits of the TVR that can be checked or updated.
- #define **TVR_ISSUER_AUTH_FAILED** 0x0507
- #define **TVR_SCRIPT_FAIL_BEFORE_AC** 0x0506
- #define **TVR_SCRIPT_FAIL_AFTER_AC** 0x0505
- #define **TVR_TERMINAL_LIMIT_EXCEEDED** 0x0408
- #define **TVR_LOWER_OFF_LIMIT_EXCEEDED** 0x0407
- #define **TVR_UPPER_OFF_LIMIT_EXCEEDED** 0x0406
- #define **TVR_RANDOM_SELECTION_ONLINE** 0x0405
- #define **TVR_MERCHANT_FORCE_ONLINE** 0x0404
- #define **TVR_CARDHOLDER_VERIF_FAILURE** 0x0308
- #define **TVR_VERIF_METHOD_UNKNOWN** 0x0307
- #define **TVR_PIN_LIMIT_EXCEEDED** 0x0306
- #define **TVR_PIN_ASKED_PINPAD_FAILURE** 0x0305
- #define **TVR_PIN_ASKED_BUT_NOT_ENTERED** 0x0304
- #define **TVR_ONLINE_PIN_ENTERED** 0x0303

- #define **TVR_SOFTWARE_VERSIONS** 0x0208
- #define **TVR_APPLICATION_EXPIRED** 0x0207
- #define **TVR_APPLICATION_NOT_EFFECTIVE** 0x0206
- #define **TVR_REQ_SERVICE_NOT_ALLOWED** 0x0205
- #define **TVR_NEW_CARD** 0x0204
- #define **TVR_OFFDATA_AUTH_NOT_DONE** 0x0108
- #define **TVR_STATIC_AUTH_FAILED** 0x0107
- #define **TVR_DATA_NOT_FOUND** 0x0106
- #define **TVR_CARD_IN_HOT_LIST** 0x0105
- #define **TVR_DYNAMIC_AUTH_FAILED** 0x0104
- #define **TVR_COMBINED_DDA_FAILED** 0x0103
- #define TSI_OFFDATA_AUTH_DONE 0x0108

  This is the list of the bits of the TSI that can be checked or updated.
- #define **TSI_CARDHOLDER_VERIF_DONE** 0x0107
- #define **TSI_CARD_RISK_DONE** 0x0106
- #define **TSI_ISSUER_AUTH_DONE** 0x0105
- #define **TSI_TERMINAL_RISK_DONE** 0x0104
- #define **TSI_SCRIPT_PROCESS_DONE** 0x0103

Enumerations

- enum **APP_SELECTION_METHODS** { **SELECTION_PSE** =0, **SELECTION_AIDLIST** }
- enum **APP_MATCH_CRITERIAS** { **MATCH_FULL** =1, **MATCH_PARTIAL_VISA**, **MATCH_PARTIAL_EUR-OPAY** }
- enum **AUTH_RESULTS** { **AUTH_RESULT_SUCCESS** =1, **AUTH_RESULT_FAILURE**, **AUTH_FAIL_PIN_-ENTRY_NOT_DONE**, **AUTH_FAIL_USER_CANCELLATION** }
- enum **BYPASS_MODES** { **BYPASS_CURRENT_METHOD_MODE** =0, **BYPASS_ALL_METHODS_MODE** }
- enum **CERTIFICATE_AC_TYPES** { **CERTIFICATE_AAC** =0, **CERTIFICATE_TC**, **CERTIFICATE_ARQC** }
- enum **CARD_RISK_TYPES** { **CDOL_1** =1, **CDOL_2** }
- enum **TAG_TYPES** { **TAG_TYPE_BINARY** =0, **TAG_TYPE_BCD**, **TAG_TYPE_STRING** }

2.18.1 Detailed Description

EMV Level 2 kernel functions, structures and defnitions. **Kernel initialisation and version verification**

Firstly the application will have to initialise the library, this will only have to be done once at the unit power up. At the same time it will be convenient also to check the version info provided by the kernel to make sure that is the expected one.

After the initialisation, the first thing to do will be to set all the data items needed to start the transaction, mainly these items correspond to configuration issues:

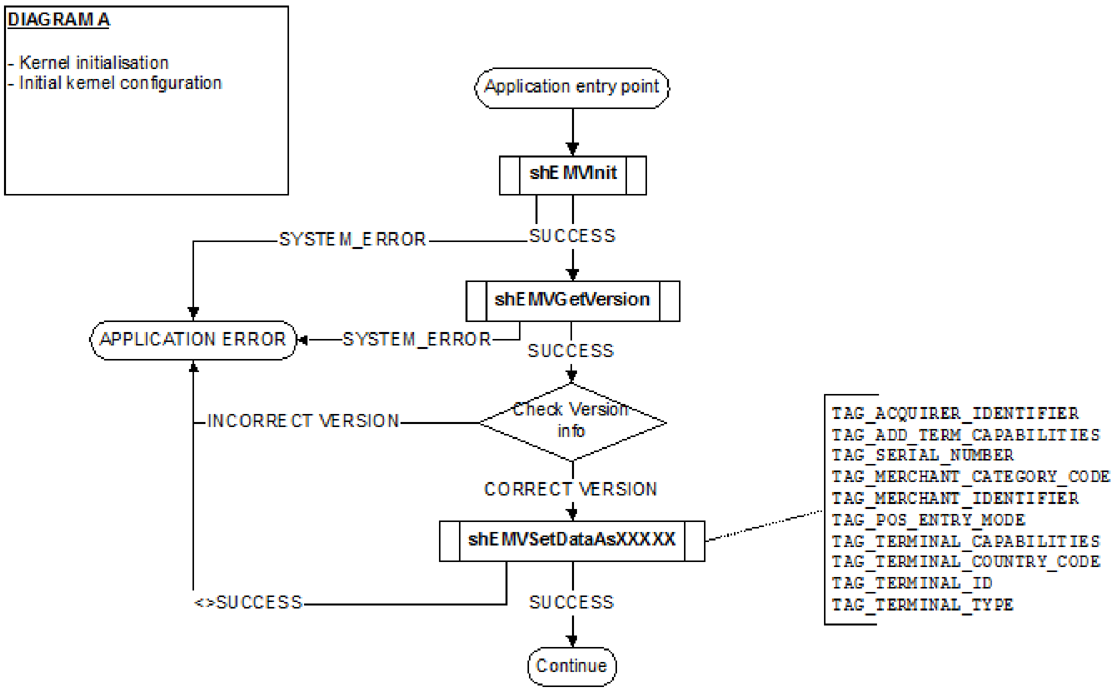

Figure 2.1: Kernel initialisation and version verification

Card recognition and ATR validation

The application will be in charge of detecting the presence of the smart card in the reader using the corresponding firmware function call, the application must power on the card also, the kernel is used in this phase to validate the ATR got from the card.

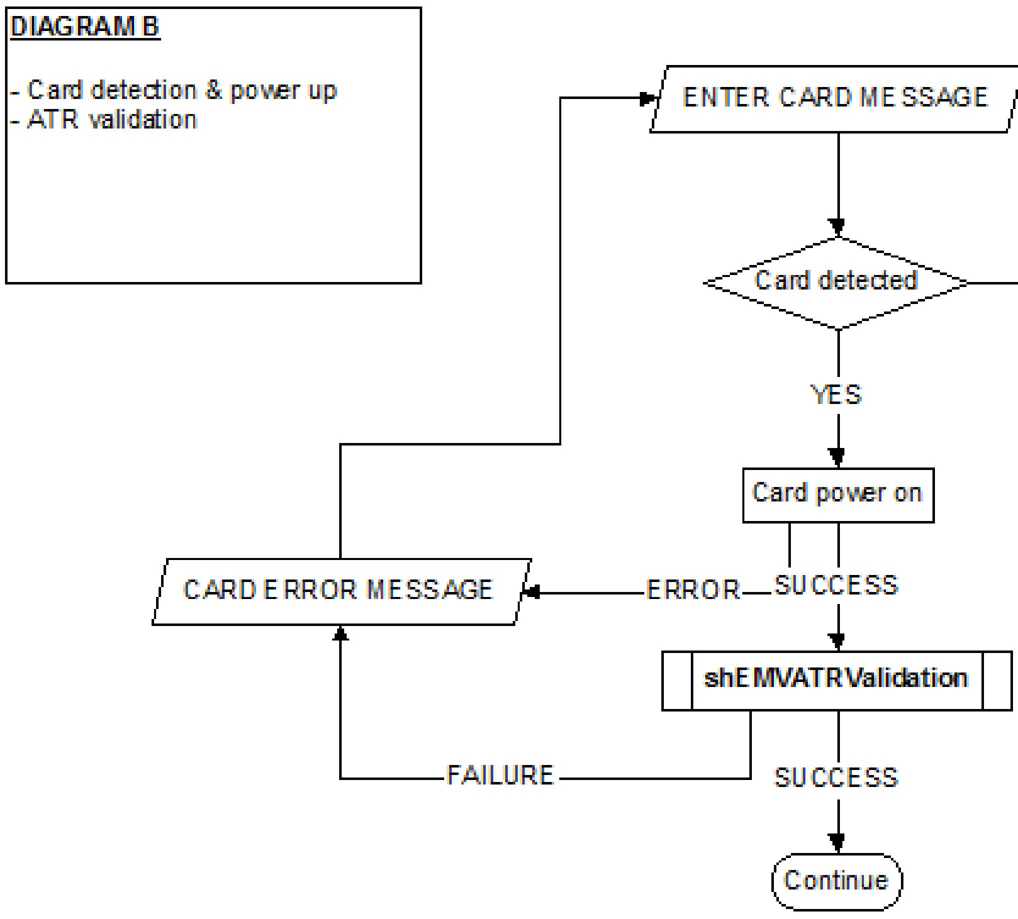

Figure 2.2: Card recognition and ATR validation

Application selection & initiation

Once the card has been powered on and the ATR validated to ensure that is a valid EMV card, the next step is to proceed with the application selection and initiation.

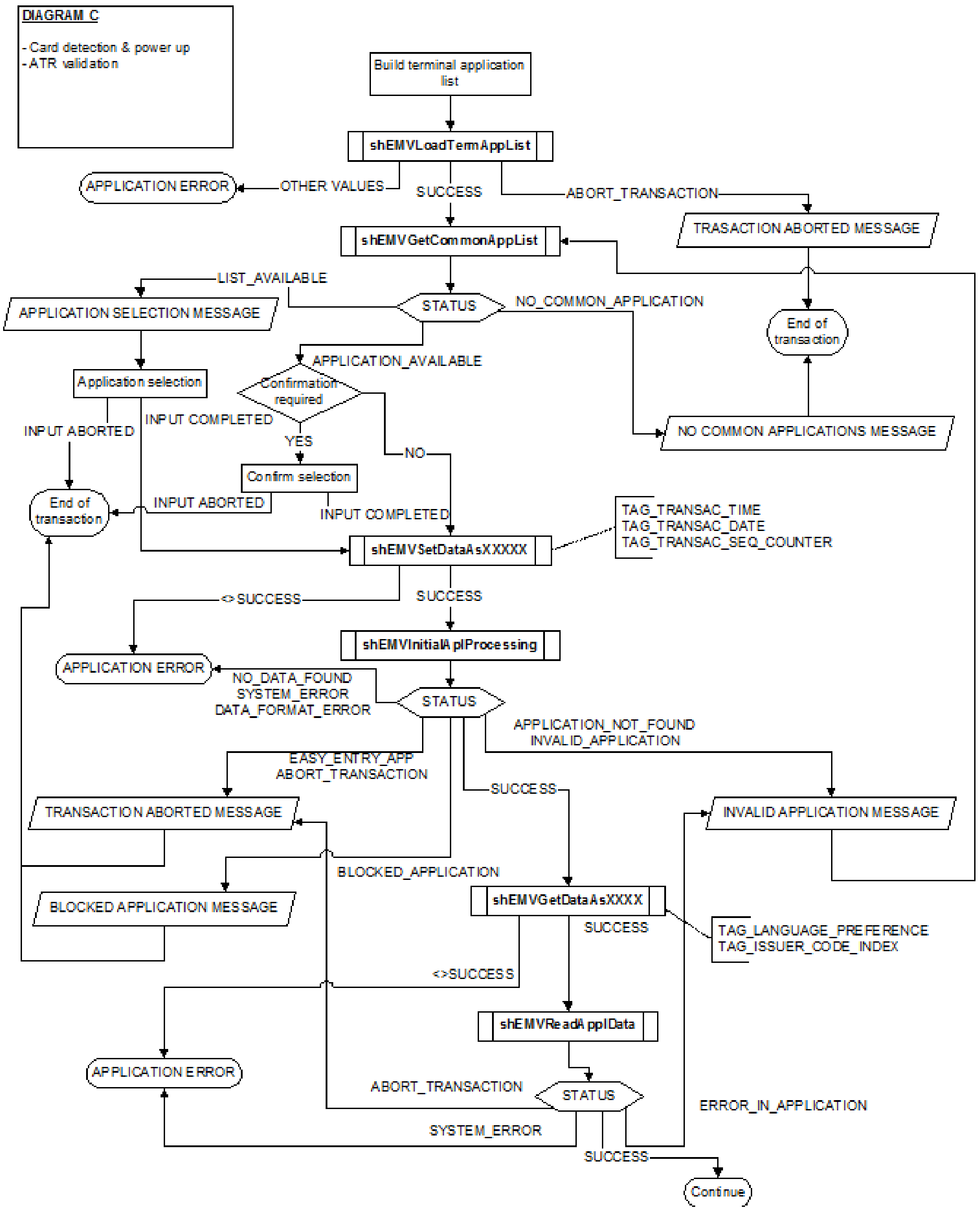

Figure 2.3: Application selection & initiation

Transaction data processing

Next the rest of the EMV transaction phases will be completed prior to the transaction decision, this includes:

Card data authentication.

Restrictions processing.

Risk control.

Cardholder verification.

For the card data authentication process the function shEMVAuthentication is called with the amount detection flag set to FALSE because it's assumed that the amount was already entered and is available for the application, if that's not the case if the application wants to use the actual value for the amount can enable this flag and provide the amount if requested during the dynamic authentication.

If the application is not offline enabled the call to the function shEMVTerminalRisk can be made without setting the data previously as shown in the diagam.

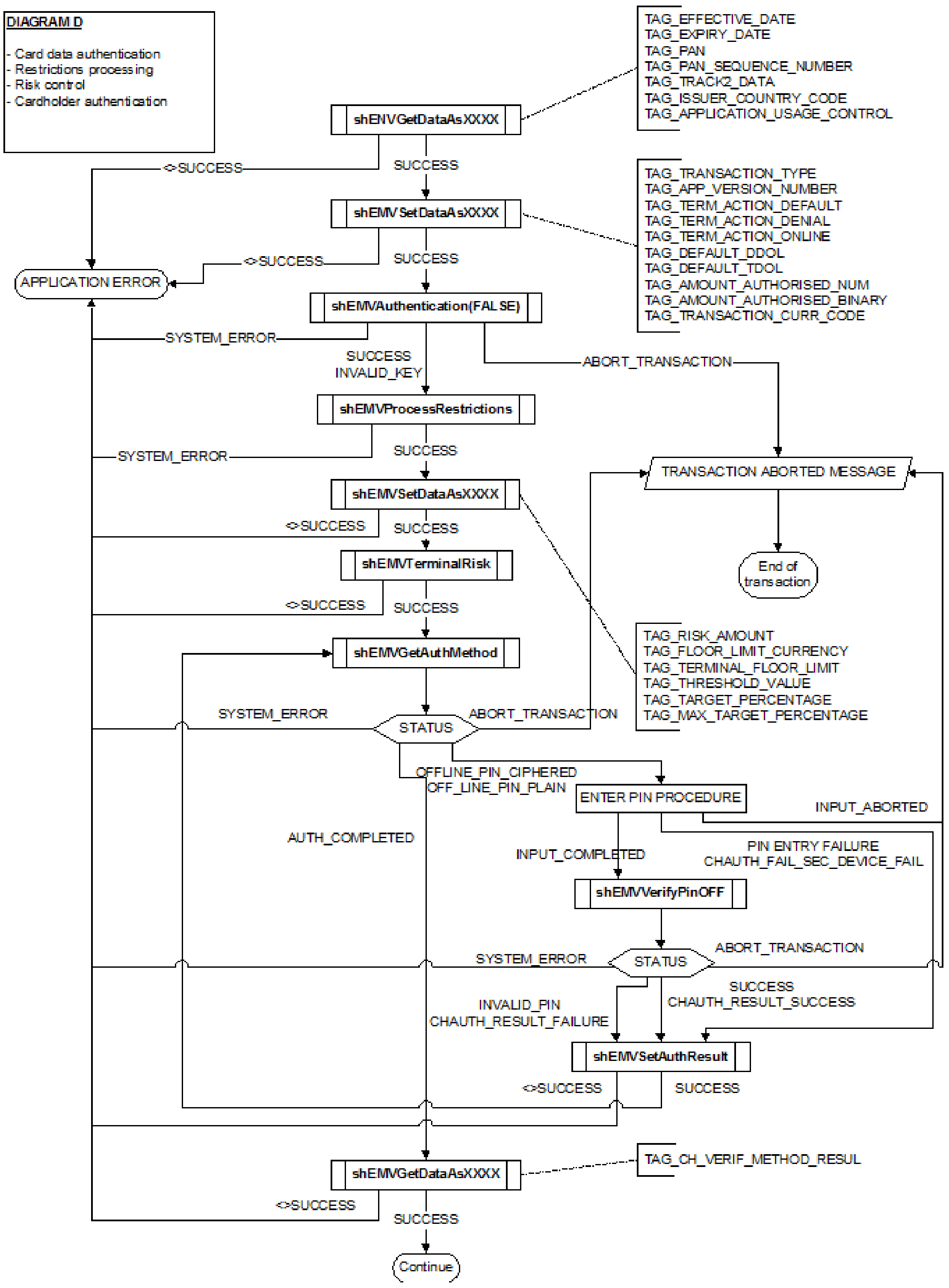

Figure 2.4: Data authentication process

Application transaction decision

At this point of the transaction, it's where the first decision is made. All the previous procedures results have been reflected on the TVR & TSI, and in this case the former is used to determine what type of transaction will be carried out from here.

Additionally for offline applications it will be necessary to check if the card is in the host list, if so the appropiate TVR bit must be updated.

The “offline possible” verification normally consists of a validation of the transactions log to ensure that the application can store the transaction data properly as well as any additional validation such as BIN control.

If the application has online only capabilities the result TRANSACTION_APPROVED should never be received as the response to the shEMVMakeTransDecision call, anyhow if this happens the transaction should be considered denied.

Once the cryptogram has been generated, it’s necessary to check its type according to the original requested type. So, it’s not acceptable to get a TC when requesting an AAC or ARQC, for that reason the verification types “AC Requested $<$ XX” appear on the flow diagram.

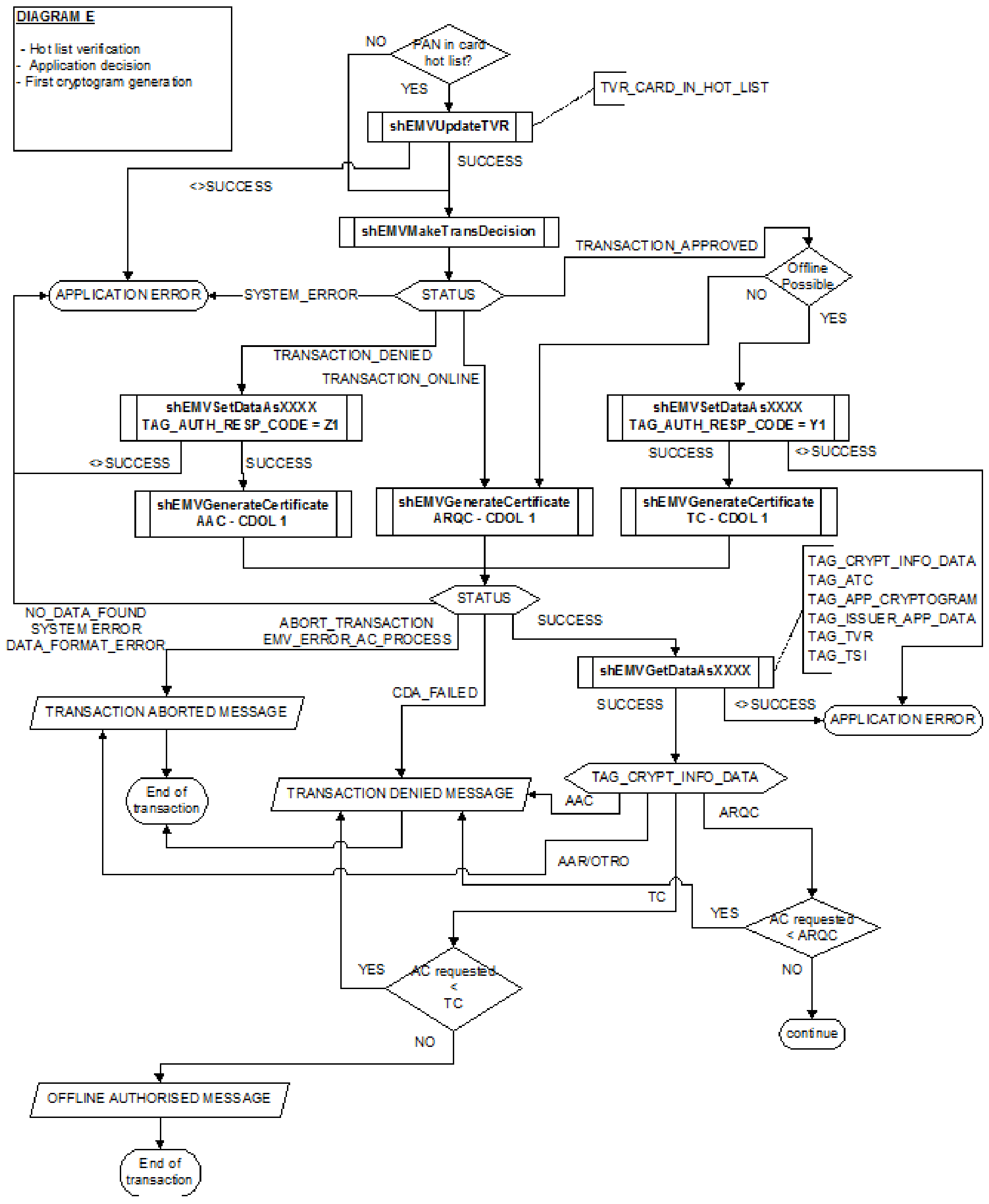

Figure 2.5: Diagram

Transaction card decision

When the issuer decision is known, it must be informed to the card requesting the appropiate cryptogram type, so that it's the card the one who has the final decision regarding the transaction. The refund/reversal procedure is out of the scope of the kernel, anyway all the data items needed can be accessed through the shEMVGetDatAsXXXX functions.

Additionally the storage of the scripts results, second cryptogram for further report to the issuer is also out of the scope of this specification and will have to be determined by the particular payment system.

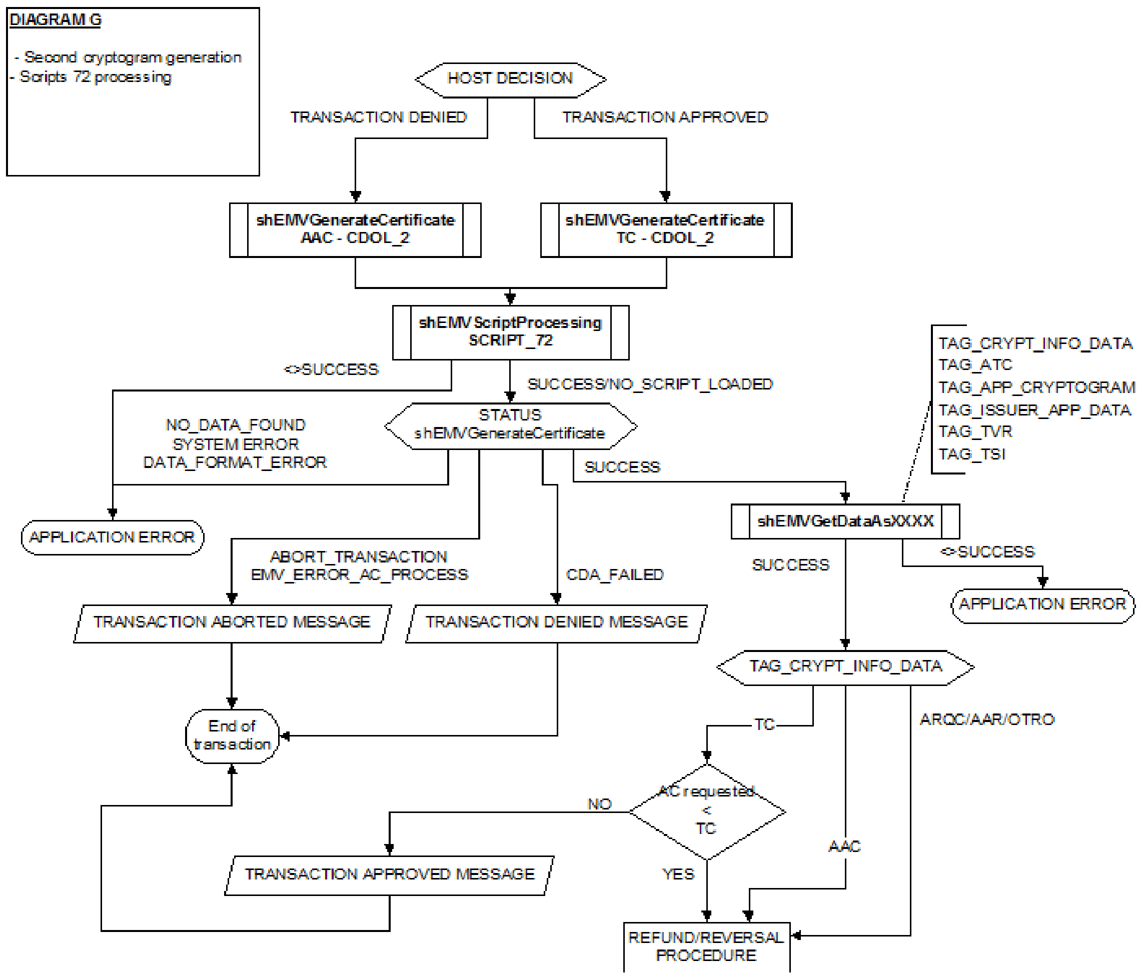

Figure 2.6: Diagram

Default processing

If the transaction cannot be completed online due to problems with the communication channel the default processing must be applied. In this case, if the application has no offline capabilities the transaction must be declined inmediately without any further processing.

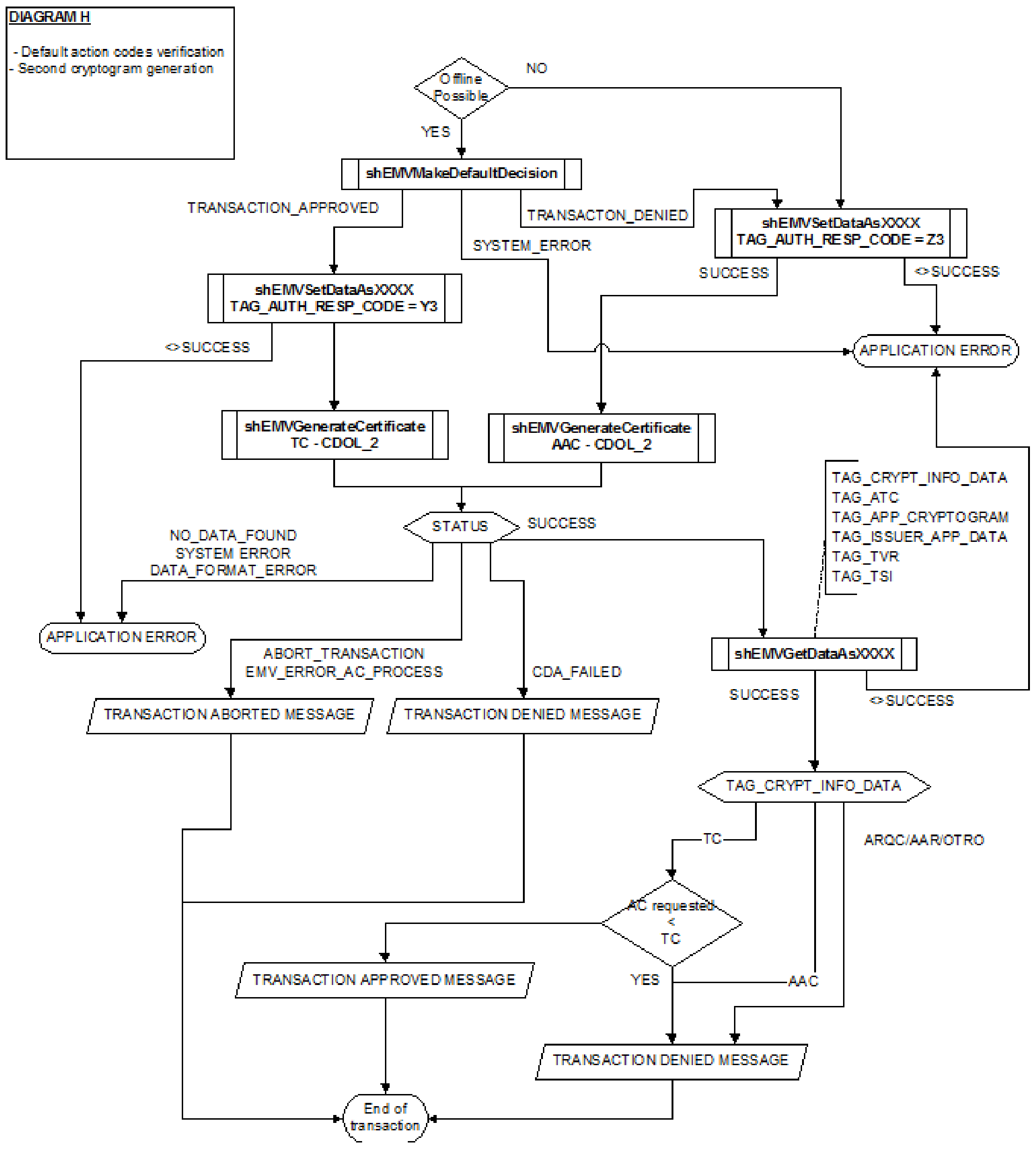

Figure 2.7: Default processing

2.19 EMV Operation Workflow

Description and block diagrams of various operations with the pinpad.

Description and block diagrams of various operations with the pinpad. **Kernel initialisation and version verification**

Firstly the application will have to initialise the library, this will only have to be done once at the unit power up. At the same time it will be convenient also to check the version info provided by the kernel to make sure that is the expected one.

After the initialisation, the first thing to do will be to set all the data items needed to start the transaction, mainly these items correspond to configuration issues:

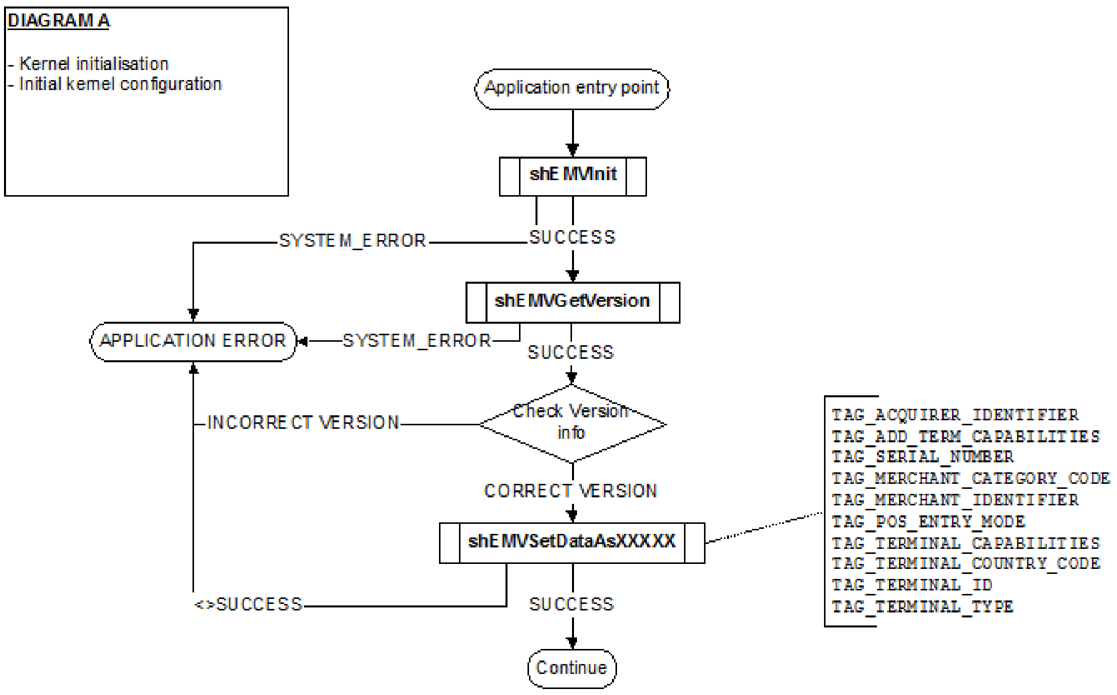

Figure 2.8: Kernel initialisation and version verification

Card recognition and ATR validation

The application will be in charge of detecting the presence of the smart card in the reader using the corresponding firmware function call, the application must power on the card also, the kernel is used in this phase to validate the ATR got from the card.

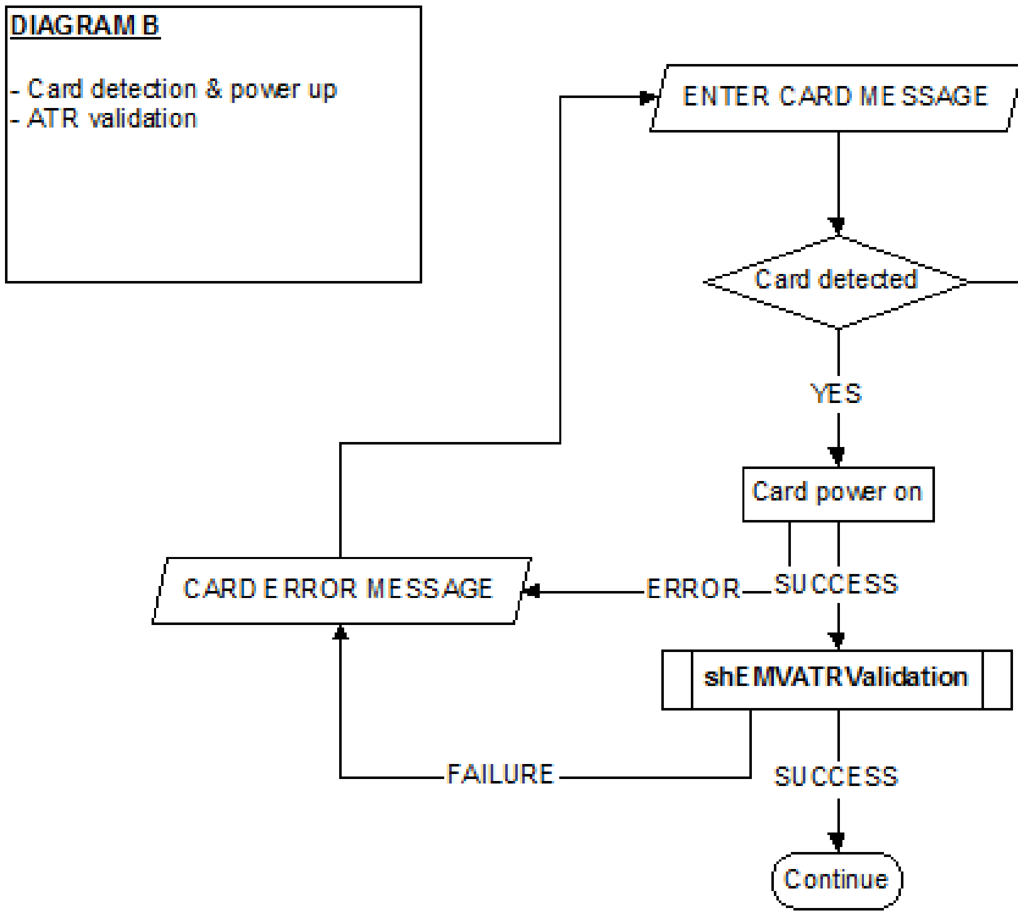

Figure 2.9: Card recognition and ATR validation

Application selection & initiation

Once the card has been powered on and the ATR validated to ensure that is a valid EMV card, the next step is to proceed with the application selection and initiation.

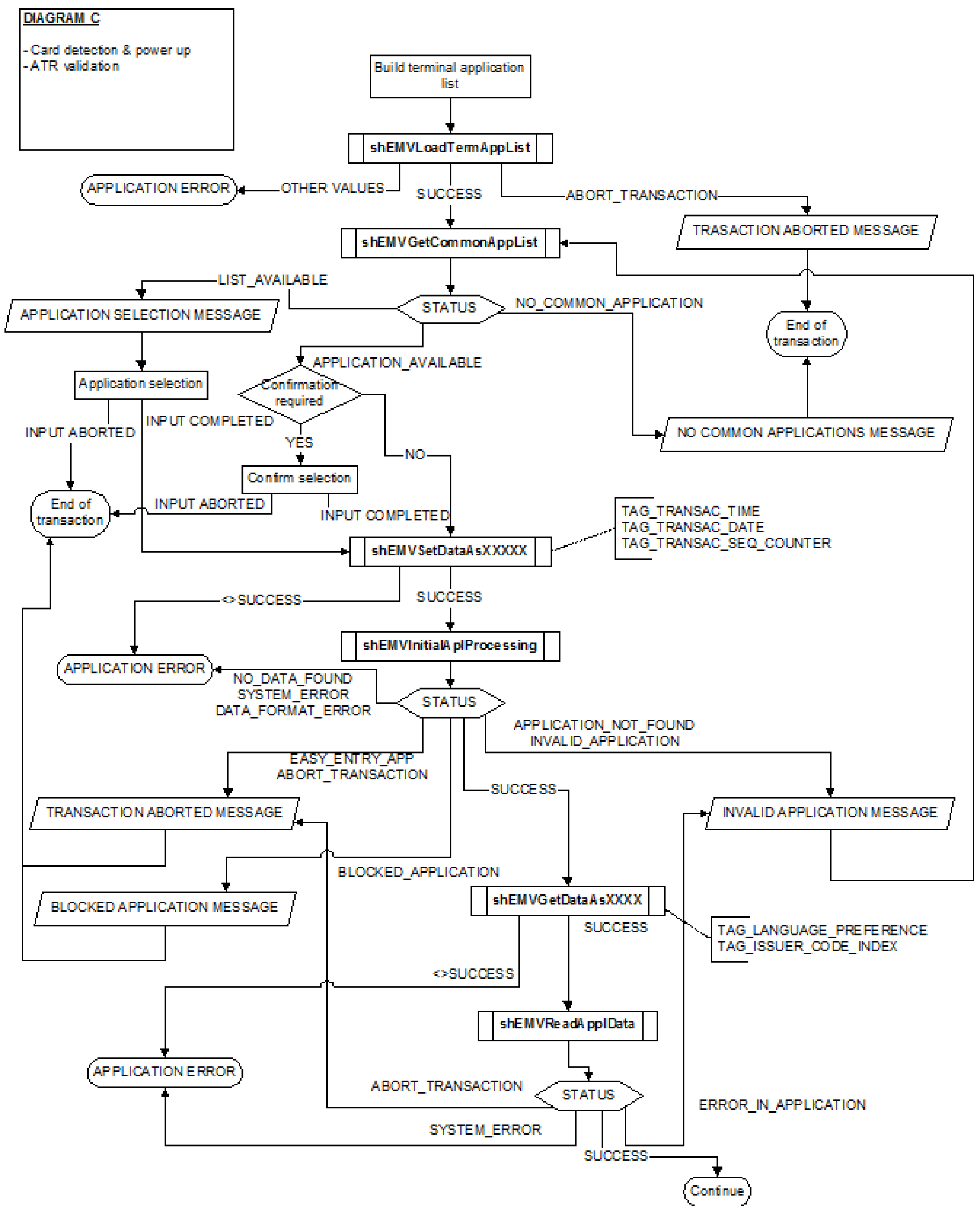

Figure 2.10: Application selection & initiation

Transaction data processing

Next the rest of the EMV transaction phases will be completed prior to the transaction decision, this includes:

Card data authentication.

Restrictions processing.

Risk control.

Cardholder verification.

For the card data authentication process the function shEMVAuthentication is called with the amount detection flag set to FALSE because it's assumed that the amount was already entered and is available for the application, if that's not the case if the application wants to use the actual value for the amount can enable this flag and provide the amount if requested during the dynamic authentication.

If the application is not offline enabled the call to the function shEMVTerminalRisk can be made without setting the data previously as shown in the diagam.

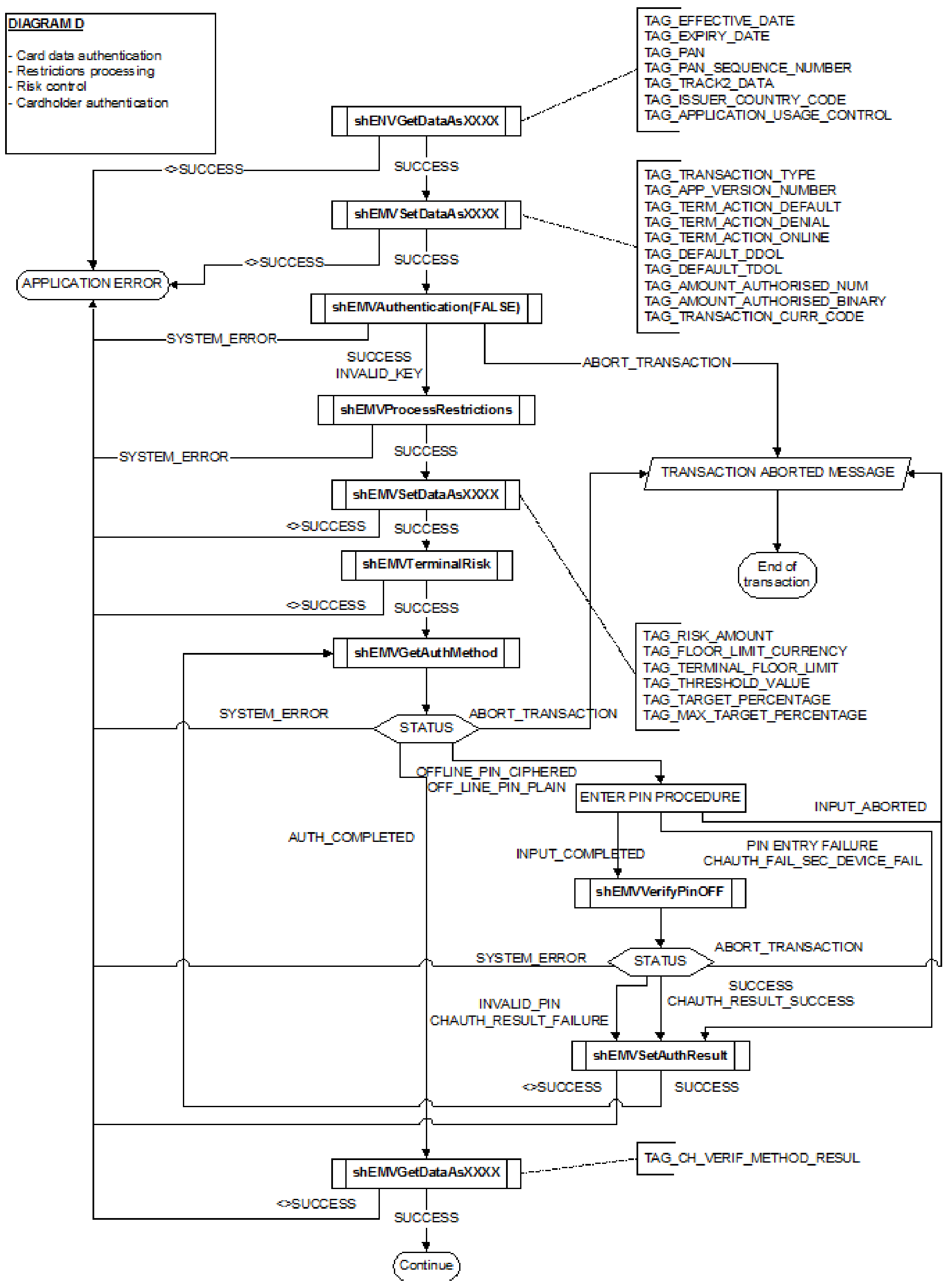

Figure 2.11: Data authentication process

Application transaction decision

At this point of the transaction, it's where the first decision is made. All the previous procedures results have been reflected on the TVR & TSI, and in this case the former is used to determine what type of transaction will be carried out from here.

Additionally for offline applications it will be necessary to check if the card is in the host list, if so the appropiate TVR bit must be updated.

The “offline possible” verification normally consists of a validation of the transactions log to ensure that the application can store the transaction data properly as well as any additional validation such as BIN control.

If the application has online only capabilities the result TRANSACTION_APPROVED should never be received as the response to the shEMVMakeTransDecision call, anyhow if this happens the transaction should be considered denied.

Once the cryptogram has been generated, it’s necessary to check its type according to the original requested type. So, it’s not acceptable to get a TC when requesting an AAC or ARQC, for that reason the verification types “AC Requested $<$ XX” appear on the flow diagram.

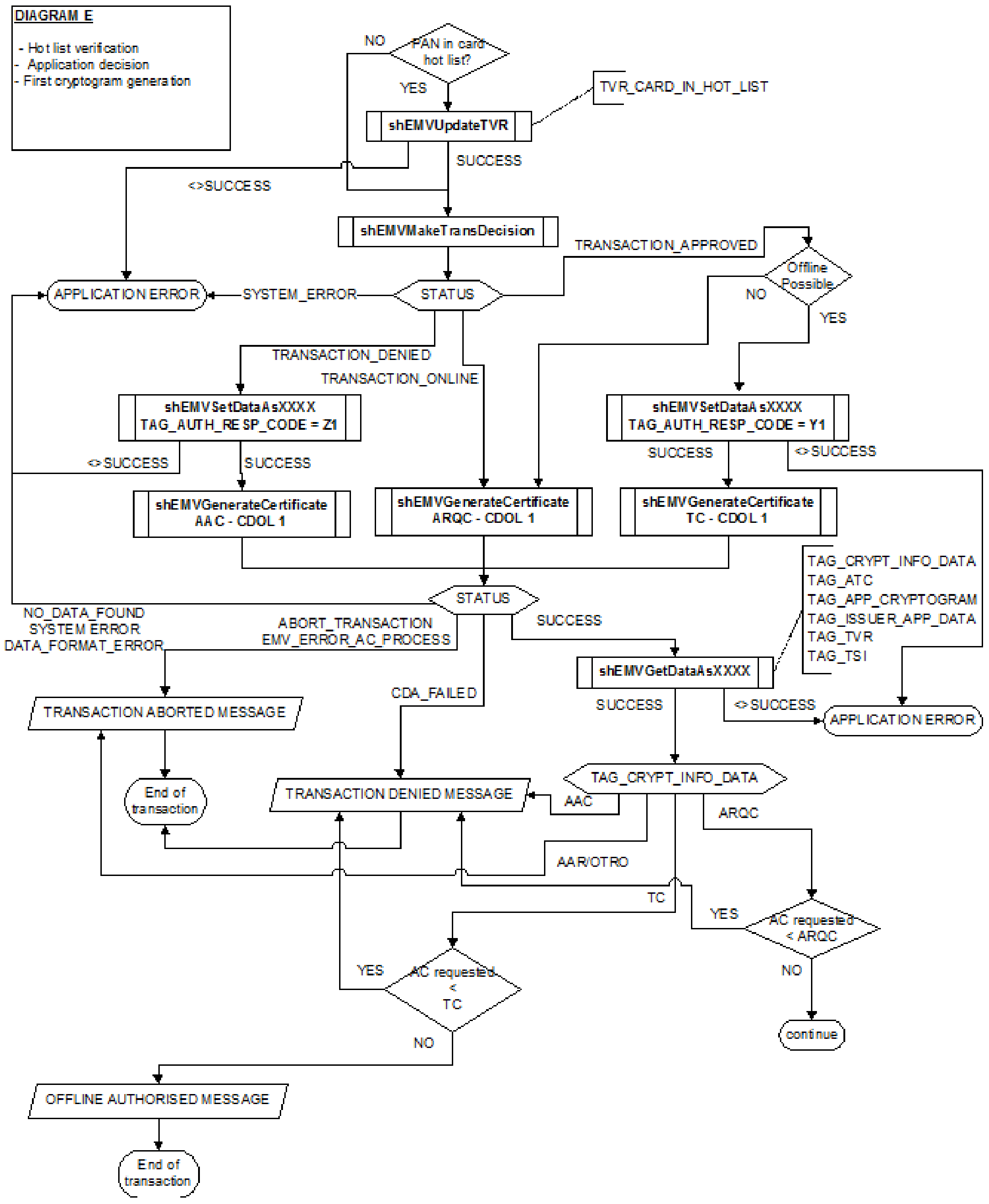

Figure 2.12: Diagram

Transaction card decision

When the issuer decision is known, it must be informed to the card requesting the appropiate cryptogram type, so that it's the card the one who has the final decision regarding the transaction. The refund/reversal procedure is out of the scope of the kernel, anyway all the data items needed can be accessed through the shEMVGetDatAsXXXX functions.

Additionally the storage of the scripts results, second cryptogram for further report to the issuer is also out of the scope of this specification and will have to be determined by the particular payment system.

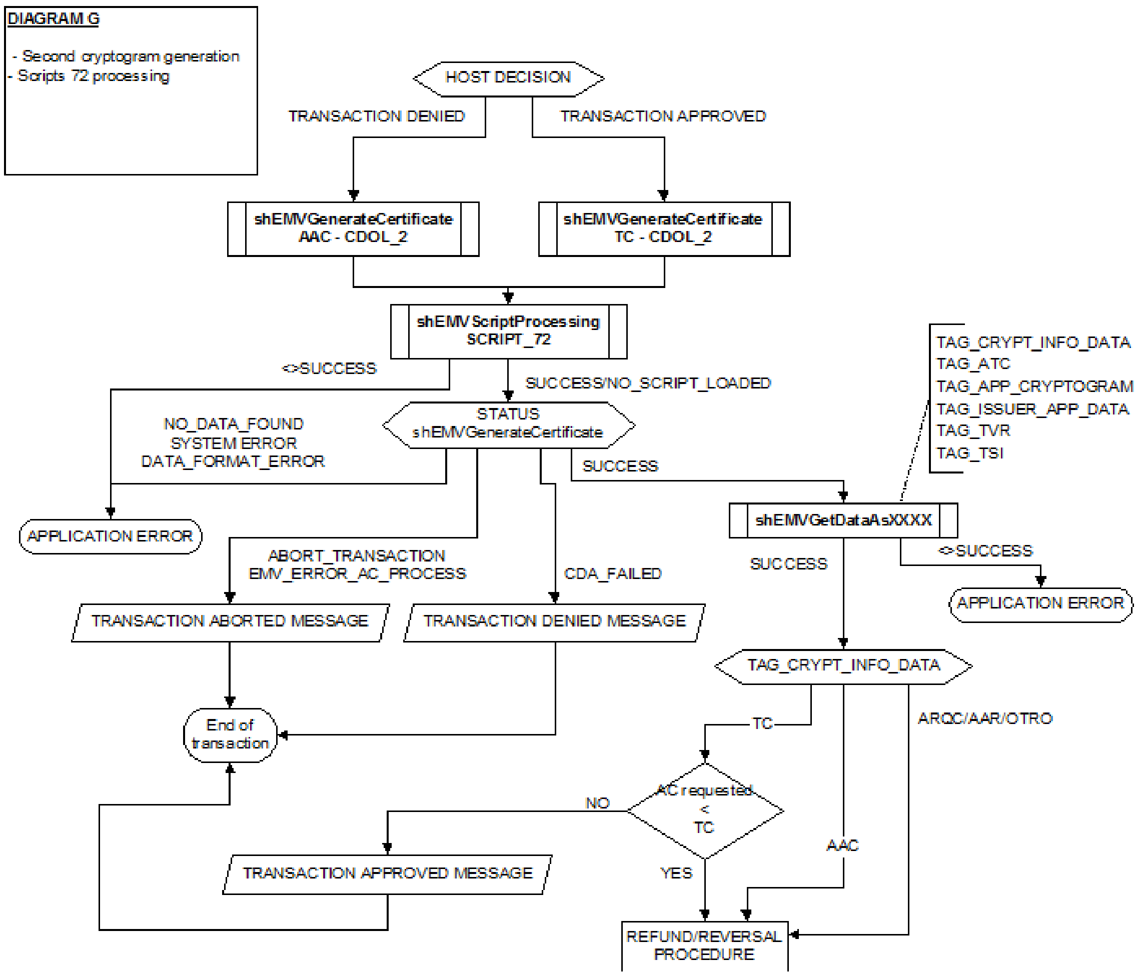

Figure 2.13: Diagram

Default processing

If the transaction cannot be completed online due to problems with the communication channel the default processing must be applied. In this case, if the application has no offline capabilities the transaction must be declined inmediately without any further processing.

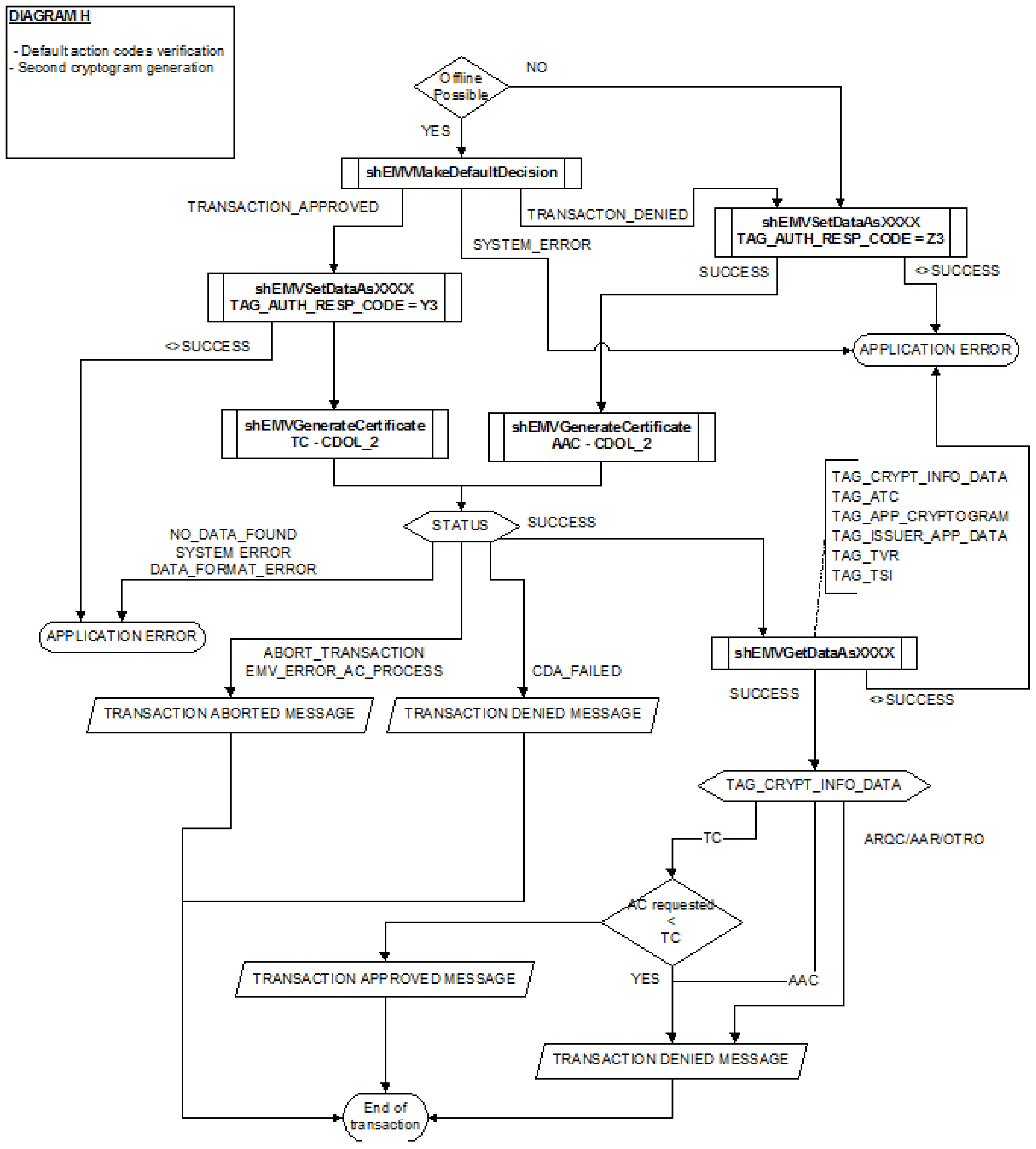

Figure 2.14: Default processing

2.20 EMV TAGs

EMV TAGs you can use with their properties.

Macros

- #define TAG_PAN 0x5A

  Source: ICC
  Length: ..10
  Format: N
  Read: YES
  Write: NO

- #define TAG_CDOL_1 0x8C

  Source: ICC
  Length: ..252
  Format: B
  Read: YES
  Write: NO

- #define TAG_CDOL_2 0x8D

  Source: ICC
  Length: ..252
  Format: B
  Read: YES
  Write: NO

- #define TAG_CVM_LIST 0x8E

  Source: ICC
  Length: ..252
  Format: B
  Read: YES
  Write: NO

- #define TAG_TDOL 0x97

  Source: ICC
  Length: ..252
  Format: B
  Read: YES
  Write: NO

- #define TAG_ISSUER_PK_CERTIFICATE 0x90

  Source: ICC
  Length: ..248
  Format: B
  Read: YES
  Write: NO

- #define TAG_SIGNED_STA_APP_DAT 0x93

  Source: ICC
  Length: ..248
  Format: B
  Read: YES
  Write: NO

- #define TAG_ISSUER_PK_REMAINDER 0x92

  Source: ICC
  Length: ..248
  Format: B
  Read: YES
  Write: NO

- #define TAG_CA_PK_INDEX 0x8F

Source: ICC
Length: 1
Format: B
Read: YES
Write: NO

- #define TAG_CARDHOLDER_NAME 0x5F20

  Source: ICC
  Length: 2-26
  Format: A
  Read: YES
  Write: NO

- #define TAG_SERVICE_CODE 0x5F30

  Source: ICC
  Length: 2
  Format: B
  Read: YES
  Write: NO

- #define TAG_CARDHOLDER_NAME_EXTEN 0x9F0B

  Source: ICC
  Length: 27-45
  Format: A
  Read: YES
  Write: NO

- #define TAG_EXPIRY_DATE 0x5F24

  Source: ICC
  Length: 3
  Format: B
  Read: YES
  Write: NO

- #define TAG_EFFECTIVE_DATE 0x5F25

  Source: ICC
  Length: 3
  Format: B
  Read: YES
  Write: NO

- #define TAG_ISSUER_COUNTRY_CODE 0x5F28

  Source: ICC
  Length: 2
  Format: B
  Read: YES
  Write: NO

- #define TAG_ISSUER_COUNTRY_CODE_A2 0x5F55

  Source: ICC
  Length: 2
  Format: A
  Read: YES
  Write: NO

- #define TAG_ISSUER_COUNTRY_CODE_A3 0x5F56

  Source: ICC
  Length: 3
  Format: A
  Read: YES
  Write: NO

- #define TAG_PAN_SEQUENCE_NUMBER 0x5F34

  Source: ICC
  Length: 1
  Format: B
  Read: YES
  Write: NO

- #define TAG_APP_DISCRETION_DAT 0x9F05

  Source: ICC
  Length: 1-32
  Format: B
  Read: YES
  Write: NO
- #define TAG_APP_USAGE_CONTROL 0x9F07

  Source: ICC
  Length: 2
  Format: B
  Read: YES
  Write: NO
- #define TAG_ICC_APP_VERSION_NUMBER 0x9F08

  Source: ICC
  Length: 2
  Format: B
  Read: YES
  Write: NO
- #define TAG_ISSUER_ACTION_DEFAULT 0x9F0D

  Source: ICC
  Length: 5
  Format: B
  Read: YES
  Write: NO
- #define TAG_ISSUER_ACTION_DENIAL 0x9F0E

  Source: ICC
  Length: 5
  Format: B
  Read: YES
  Write: NO
- #define TAG_ISSUER_ACTION_ONLINE 0x9F0F

  Source: ICC
  Length: 5
  Format: B
  Read: YES
  Write: NO
- #define TAG_APPL_REF_CURRENCY 0x9F3B

  Source: ICC
  Length: 2-8
  Format: N
  Read: YES
  Write: NO
- #define TAG_APPL_CURRENCY_CODE 0x9F42

  Source: ICC
  Length: 2
  Format: N
  Read: YES
  Write: NO
- #define TAG_APPL_REF_CURRENCY_EXP 0x9F43

  Source: ICC
  Length: 1-4
  Format: N
  Read: YES
  Write: NO
- #define TAG_APPL_CURRENCY_EXP 0x9F44

  Source: ICC
  Length: 1
  Format: N
  Read: YES
  Write: NO

- #define TAG_ICC_PK_CERTIFICATE 0x9F46

  Source: ICC
  Length: 248
  Format: B
  Read: YES
  Write: NO
- #define TAG_ICC_PIN_PK_CERTIFICATE 0x9F2D

  Source: ICC
  Length: 248
  Format: B
  Read: YES
  Write: NO
- #define TAG_ICC_PK_EXP 0x9F47

  Source: ICC
  Length: 1-3
  Format: B
  Read: YES
  Write: NO
- #define TAG_ICC_PIN_PK_EXP 0x9F2E

  Source: ICC
  Length: 1-3
  Format: B
  Read: YES
  Write: NO
- #define TAG_ICC_PK_REMAINDER 0x9F48

  Source: ICC
  Length: 248
  Format: B
  Read: YES
  Write: NO
- #define TAG_ICC_PIN_PK_REMAINDER 0x9F2F

  Source: ICC
  Length: 248
  Format: B
  Read: YES
  Write: NO
- #define TAG_STA_DAT_AUTH_TAG_LIST 0x9F4A

  Source: ICC
  Length: ..252
  Format: B
  Read: YES
  Write: NO
- #define TAG_DDOL 0x9F49

  Source: ICC
  Length: ..252
  Format: B
  Read: YES
  Write: NO
- #define TAG_ISSUER_PK_EXP 0x9F32

  Source: ICC
  Length: 1-3
  Format: B
  Read: YES
  Write: NO
- #define TAG_LOW_CONSEC_OFFLINE_LIMIT 0x9F14

  Source: ICC
  Length: 1
  Format: B
  Read: YES
  Write: NO

- #define TAG_UPP_CONSEC_OFFLINE_LIMIT 0x9F23

  Source: ICC
  Length: 1
  Format: B
  Read: YES
  Write: NO
- #define TAG_TRACK2_DISCRETION_DAT 0x9F20

  Source: ICC
  Length: ..22
  Format: N
  Read: YES
  Write: NO
- #define TAG_TRACK1_DISCRETION_DAT 0x9F1F

  Source: ICC
  Length: ..52
  Format: A
  Read: YES
  Write: NO
- #define TAG_TRACK2_EQUIVALENT_DATA 0x57

  Source: ICC
  Length: ..19
  Format: B
  Read: YES
  Write: NO
- #define TAG_UNPREDICTABLE_NUMBER 0x9F37

  Source: KER
  Length: 4
  Format: B
  Read: YES
  Write: NO
- #define TAG_ACQUIRER_IDENTIFIER 0x9F01

  Source: APP
  Length: 6
  Format: N
  Read: YES
  Write: YES
- #define TAG_ADD_TERM_CAPABILITIES 0x9F40

  Source: APP
  Length: 5
  Format: B
  Read: YES
  Write: YES
- #define TAG_AMOUNT_AUTHORISED_BINARY 0x81

  Source: APP
  Length: 4
  Format: B
  Read: YES
  Write: YES
- #define TAG_AMOUNT_AUTHORISED_NUM 0x9F02

  Source: APP
  Length: 6
  Format: N
  Read: YES
  Write: YES
- #define TAG_AMOUNT_OTHER_BINARY 0x9F04

  Source: APP
  Length: 4
  Format: B
  Read: YES
  Write: YES

- #define TAG_AMOUNT_OTHER_NUM 0x9F03

  Source: APP
  Length: 6
  Format: N
  Read: YES
  Write: YES
- #define TAG_AMOUNT_REF_CURR 0x9F3A

  Source: APP
  Length: 4
  Format: B
  Read: YES
  Write: YES
- #define TAG_APP_CRYPTOGRAM 0x9F26

  Source: ICC
  Length: 8
  Format: B
  Read: YES
  Write: NO
- #define TAG_AFL 0x94

  Source: ICC
  Length: ...252
  Format: B
  Read: YES
  Write: NO
- #define TAG_ICC_AID 0x4F

  Source: ICC
  Length: 5-16
  Format: B
  Read: YES
  Write: NO
- #define TAG_TERM_AID 0x9F06

  Source: APP
  Length: 5-16
  Format: B
  Read: YES
  Write: YES
- #define TAG_AIP 0x82

  Source: ICC
  Length: 2
  Format: B
  Read: YES
  Write: NO
- #define TAG_APP_LABEL 0x50

  Source: ICC
  Length: 1-16
  Format: AN
  Read: YES
  Write: NO
- #define TAG_APP_PREFERRED_NAME 0x9F12

  Source: ICC
  Length: 1-16
  Format: AN
  Read: YES
  Write: NO
- #define TAG_APP_PRIORITY_INDICATOR 0x87

  Source: ICC
  Length: 1
  Format: B
  Read: YES
  Write: NO

- #define TAG_ATC 0x9F36

  Source: ICC
  Length: 2
  Format: B
  Read: YES
  Write: NO
- #define TAG_APP_VERSION_NUMBER 0x9F09

  Source: APP
  Length: 2
  Format: B
  Read: YES
  Write: YES
- #define TAG_AUTH_CODE 0x89

  Source: APP
  Length: 6
  Format: AN
  Read: YES
  Write: YES
- #define TAG_AUTH_RESP_CODE 0x8A

  Source: APP
  Length: 2
  Format: AN
  Read: YES
  Write: YES
- #define TAG_CH_VERIF_METHOD_RESULT 0x9F34

  Source: KER
  Length: 3
  Format: B
  Read: YES
  Write: NO
- #define TAG_CA_PUBLIC_KEY_INDEX 0x9F22

  Source: APP
  Length: 1
  Format: B
  Read: YES
  Write: YES
- #define TAG_CRYPT_INFO_DATA 0x9F27

  Source: ICC
  Length: 1
  Format: B
  Read: YES
  Write: NO
- #define TAG_DAT_AUTH_CODE 0x9F45

  Source: ICC
  Length: 2
  Format: B
  Read: YES
  Write: NO
- #define TAG_ICC_DYN_NUMBER 0x9F4C

  Source: ICC
  Length: 2-8
  Format: B
  Read: YES
  Write: NO
- #define TAG_SERIAL_NUMBER 0x9F1E

  Source: APP
  Length: 8
  Format: AN
  Read: YES
  Write: YES

- #define TAG_ISSUER_APP_DAT 0x9F10

  Source: ICC
  Length: ..32
  Format: B
  Read: YES
  Write: NO
- #define TAG_ISSUER_AUTH_DAT 0x91

  Source: APP
  Length: 8-16
  Format: B
  Read: YES
  Write: YES
- #define TAG_ISSUER_CODE_INDEX 0x9F11

  Source: ICC
  Length: 1
  Format: N
  Read: YES
  Write: NO
- #define TAG_LANGUAGE_PREFERENCE 0x5F2D

  Source: ICC
  Length: 2-8
  Format: AN
  Read: YES
  Write: NO
- #define TAG_LATC 0x9F13

  Source: ICC
  Length: 2
  Format: B
  Read: YES
  Write: NO
- #define TAG_MERCHANT_CATEGORY_CODE 0x9F15

  Source: APP
  Length: 2
  Format: N
  Read: YES
  Write: YES
- #define TAG_MERCHANT_IDENTIFIER 0x9F16

  Source: APP
  Length: 15
  Format: AN
  Read: YES
  Write: YES
- #define TAG_PIN_TRY_COUNTER 0x9F17

  Source: ICC
  Length: 1
  Format: B
  Read: YES
  Write: NO
- #define TAG_POS_ENTRY_MODE 0x9F39

  Source: APP
  Length: 1
  Format: N
  Read: YES
  Write: YES
- #define TAG_PDOL 0x9F38

  Source: ICC
  Length: ..252
  Format: B
  Read: YES
  Write: NO

- #define TAG_TERMINAL_CAPABILITIES 0x9F33

  Source: APP
  Length: 3
  Format: B
  Read: YES
  Write: YES
- #define TAG_TERMINAL_COUNTRY_CODE 0x9F1A

  Source: APP
  Length: 2
  Format: N
  Read: YES
  Write: YES
- #define TAG_TERMINAL_FLOOR_LIMIT 0x9F1B

  Source: APP
  Length: 4
  Format: B
  Read: YES
  Write: YES
- #define TAG_TERMINAL_ID 0x9F1C

  Source: APP
  Length: 8
  Format: AN
  Read: YES
  Write: YES
- #define TAG_TERMINAL_RISK_DAT 0x9F1D

  Source: APP
  Length: 1-8
  Format: B
  Read: YES
  Write: YES
- #define TAG_TERMINAL_TYPE 0x9F35

  Source: APP
  Length: 1
  Format: N
  Read: YES
  Write: YES
- #define TAG_TVR 0x95

  Source: KER
  Length: 5
  Format: B
  Read: YES
  Write: NO
- #define TAG_TRANSACTION_CURR_CODE 0x5F2A

  Source: APP
  Length: 2
  Format: N
  Read: YES
  Write: YES
- #define TAG_TRANSACTION_CURR_EXP 0x5F36

  Source: APP
  Length: 1
  Format: N
  Read: YES
  Write: YES
- #define TAG_TRANSACTION_DATE 0x9A

  Source: APP
  Length: 3
  Format: N
  Read: YES
  Write: YES

- #define TAG_TRANSACTION_REF_CURR_CODE 0x9F3C

  Source: APP
  Length: 2
  Format: N
  Read: YES
  Write: YES
- #define TAG_TRANSACTION_REF_CURR_EXP 0x9F3D

  Source: APP
  Length: 1
  Format: N
  Read: YES
  Write: YES
- #define TAG_TRANSACTION_SEQ_COUNTER 0x9F41

  Source: APP
  Length: 2-4
  Format: N
  Read: YES
  Write: YES
- #define TAG_TSI 0x9B

  Source: KER
  Length: 2
  Format: B
  Read: YES
  Write: NO
- #define TAG_TRANSACTION_TIME 0x9F21

  Source: APP
  Length: 3
  Format: N
  Read: YES
  Write: YES
- #define TAG_TRANSACTION_TYPE 0x9C

  Source: APP
  Length: 1
  Format: N
  Read: YES
  Write: YES
- #define TAG_SIGNED_DYN_APP_DAT 0x9F4B

  Source: ICC
  Length: ..248
  Format: B
  Read: YES
  Write: NO
- #define TAG_TC_HASH_VALUE 0x98

  Source: APP
  Length: 20
  Format: B
  Read: YES
  Write: YES
- #define TAG_ACCOUNT_TYPE 0x5F37

  Source: APP
  Length: 1
  Format: N
  Read: YES
  Write: YES
- #define TAG_BANK_IDENTIFIER_CODE 0x5F54

  Source: ICC
  Length: 8-11
  Format: AN
  Read: YES
  Write: NO

- #define TAG_IBAN 0x5F53

  Source: ICC
  Length: ..34
  Format: AN
  Read: YES
  Write: NO
- #define TAG_ISSUER_IDENTIFICATION_NUMBER 0x42

  Source: ICC
  Length: 3
  Format: N
  Read: YES
  Write: NO
- #define TAG_ISSUER_URL 0x5F50

  Source: ICC
  Length: ..255
  Format: AN
  Read: YES
  Write: NO
- #define TAG_LOG_ENTRY 0x9F4D

  Source: ICC
  Length: 2
  Format: B
  Read: YES
  Write: NO
- #define TAG_TRANSACTION_CATEGORY_CODE 0x9F53

  Source: APP
  Length: 1
  Format: B
  Read: YES
  Write: YES
- #define TAG_RISK_AMOUNT 0xDF02

  Source: APP
  Length: 4
  Format: B
  Read: YES
  Write: YES
- #define TAG_TERM_ACTION_DEFAULT 0xDF03

  Source: APP
  Length: 5
  Format: B
  Read: YES
  Write: YES
- #define TAG_TERM_ACTION_DENIAL 0xDF04

  Source: APP
  Length: 5
  Format: B
  Read: YES
  Write: YES
- #define TAG_TERM_ACTION_ONLINE 0xDF05

  Source: APP
  Length: 5
  Format: B
  Read: YES
  Write: YES
- #define TAG_THRESHOLD_VALUE 0xDF07

  Source: APP
  Length: 5
  Format: B
  Read: YES
  Write: YES

- #define TAG_TARGET_PERCENTAGE 0xDF08

  Source: APP
  Length: 1
  Format: B
  Read: YES
  Write: YES
- #define TAG_MAX_TARGET_PERCENTAGE 0xDF09

  Source: APP
  Length: 1
  Format: B
  Read: YES
  Write: YES
- #define TAG_DEFAULT_DDOL 0xDF15

  Source: APP
  Length: ...252
  Format: B
  Read: YES
  Write: YES
- #define TAG_DEFAULT_TDOL 0xDF18

  Source: APP
  Length: ..252
  Format: B
  Read: YES
  Write: YES
- #define TAG_FLOOR_LIMIT_CURRENCY 0xDF19

  Source: APP
  Length: 2
  Format: B
  Read: YES
  Write: YES
- #define TAG_OFF_AUTH_DAT 0xDF23

  Source: APP
  Length: ..2048
  Format: B
  Read: YES
  Write: NO
- #define TAG_ISSUER_SCRIPTS 0xDF24

  Source: APP
  Length: ..256
  Format: B
  Read: YES
  Write: YES
- #define TAG_ISSUER_SCRIPTS_RESULT 0xDF25

  Source: APP
  Length: ..256
  Format: B
  Read: YES
  Write: NO

2.20.1 Detailed Description

EMV TAGs you can use with their properties.

2.21 EMV Status Codes

These status codes are returned from every EMV function to indicate the result of it.

Macros

- #define EMV_SUCCESS 0

  Operation successful.
- #define EMV_LIST_AVAILABLE 1

  More than one matching applications found.
- #define EMV_APPLICATION_AVAILABLE 2

  Only one matching application found.
- #define EMV_NO_COMMON_APPLICATION 3

  No matching applications found.
- #define EMV_EASY_ENTRY_APP 4

  Easy Entry application.
- #define EMV_AMOUNT_NEEDED 5

  Amount is requested by the dynamic data authentication.
- #define EMV_RESULT_NEEDED 6

  Result needed.
- #define EMV_AUTH_COMPLETED 7

  Authentication is completed.
- #define EMV_AUTH_NOT_DONE 8

  Authentication was not performed.
- #define EMV_OFFLINE_PIN_PLAIN 9

  OFFLINE plain text pin is required.
- #define EMV_ONLINE_PIN 10

  ONLINE pin is required.
- #define EMV_OFFLINE_PIN_CIPHERED 11

  OFFLINE ciphered pin is required.
- #define EMV_BLOCKED_APPLICATION 12

  Explicit selection was done and blocked AIDs were found.
- #define EMV_TRANSACTION_ONLINE 13

  An online request should be done.
- #define EMV_TRANSACTION_APPROVED 14

  Transaction can be accepted offline.
- #define EMV_TRANSACTION_DENIED 15

  Transaction must be declined.
- #define EMV_CDA_FAILED 16

  CDA failed and the cryptogram got is not an AAC or the data handed for DDA was not found.
- #define EMV_INVALID_PIN 17

  Incorrect PIN.
- #define EMV_INVALID_PIN_LAST_ATTEMPT 18

  Incorrect PIN, last attempt available only.
- #define EMV_FAILURE 50

  Command failed, possibly due wrong imput parameters - wrong ATR, bit values, etc.
- #define EMV_NO_DATA_FOUND 51

  Incoming data pointer is null or empty.
- #define EMV_SYSTEM_ERROR 52

  Internal system error occurred.

- #define EMV_DATA_FORMAT_ERROR 53

  Incorrect format found in the input parameters.
- #define EMV_INVALID_ATR 54

  Invalid ATR sequence, not according to specs.
- #define EMV_ABORT_TRANSACTION 55

  Severe error occurred transaction must be aborted.
- #define EMV_APPLICATION_NOT_FOUND 56

  AID not found in the card.
- #define EMV_INVALID_APPLICATION 57

  Application is not correct.
- #define EMV_ERROR_IN_APPLICATION 58

  Some error during read process.
- #define EMV_CARD_BLOCKED 59

  Status word got from the PSE selection indicates that the card is blocked.
- #define EMV_NO_SCRIPT_LOADED 61

  No script loaded.
- #define EMV_INVALID_TAG 62

  Tag cannot be read.
- #define EMV_INVALID_LENGTH 63

  Length of the buffer is incorrect.
- #define EMV_INVALID_HASH 64

  Error in the HASH verification.
- #define EMV_INVALID_KEY 65

  No key was found to do the verification.
- #define EMV_NO_MORE_KEYS 66

  No more available locations for keys.
- #define EMV_ERROR_AC_PROCESS 67

  Error processing the AC generation.
- #define EMV_ERROR_AC_DENIED 68

  Status word got from the card is 6985.
- #define EMV_NO_CURRENT_METHOD 69

  No method is currently applicable.
- #define EMV_RESULT_ALREADY_LOADED 70

  Result already loaded for the current method.
- #define **EMV_LAST_EMVKERNEL_ERR_CODE** 70
- #define **EMV_INVALID_REMAINDER** 80
- #define EMV_INVALID_HEADER 81

  Invalid header.
- #define EMV_INVALID_FOOTER 82

  Invalid footer.
- #define EMV_INVALID_FORMAT 83

  Invalid format.
- #define EMV_INVALID_CERTIFICATE 84

  Invalid certificate.
- #define EMV_INVALID_SIGNATURE 85

  Invalid signature.

2.21.1 Detailed Description

These status codes are returned from every EMV function to indicate the result of it.

2.22 Transaction Start

This section includes the command used to start the transaction: ATR validation and application selection.

Functions

- (BOOL) - DTDevices::emvInitialise:

  This command initializes the emv kernel, call it before calling any other EMV function.
- (BOOL) - DTDevices::emvDeinitialise:

  This command deinitializes the emv kernel and frees the allocated resources, call it after you are done with the EMV transaction.
- (BOOL) - DTDevices::emvATRValidation:warmReset:error:

  The command is in charge of validating the ATR sequence got from the card to ensure that is fully EMV compliant and that obeys the rules stated in the specification.
- (BOOL) - DTDevices::emvLoadAppList:selectionMethod:includeBlockedAIDs:error:

  The command initiates the application selection process, loading the application list supported by the terminal.
- (NSArray ∗) - DTDevices::emvGetCommonAppList:error:

  The command gets back the list of common applications supported by the terminal and the card, actually this commands will end or resume the selection procedure.

2.22.1 Detailed Description

This section includes the command used to start the transaction: ATR validation and application selection.

2.22.2 Function Documentation

2.22.2.1 - (BOOL) emvATRValidation: (NSData ∗) *ATR* warmReset:(BOOL) *warmReset* error:(NSError ∗∗) *error*

The command is in charge of validating the ATR sequence got from the card to ensure that is fully EMV compliant and that obeys the rules stated in the specification.

Note

Upon successful execution, EMV kernel status is stored in emvLastStatus property.

Parameters

warmReset	- holds the type of power up applied if cold or warm power up.
ATR	- ATR sequence received form the card: TS+T0+TB1+TC1+TS+T0+TB1+TC1+TD1+TD2+T-A3+TB3+TCK
error	returns error information, you can pass nil if you don't want it

Returns

TRUE upon success, FALSE otherwise

2.22.2.2 - (BOOL) emvDeinitialise: (NSError ∗∗) *error*

This command deinitializes the emv kernel and frees the allocated resources, call it after you are done with the EMV transaction.

Note

Upon successful execution, EMV kernel status is stored in emvLastStatus property.

Parameters

error	returns error information, you can pass nil if you don't want it

Returns

TRUE upon success, FALSE otherwise

2.22.2.3 - (NSArray ∗) emvGetCommonAppList: (BOOL ∗) *confirmationRequired* error:(NSError ∗∗) *error*

The command gets back the list of common applications supported by the terminal and the card, actually this commands will end or resume the selection procedure.

Initially the command will check the provided data, if it's empty string, the status NO_DATA_FOUND will be returned, if during the procedure any internal error occurs the status will be EMV_SYSTEM_ERROR. On the other hand, if the process and be completed correctly the possible status returned will be: EMV_LIST_AVAILABLE, EMV_APPLICATION_AVAILABLE, EMV_NO_COMMON_APPLICATION according to the number of common applications found.

Note

The application may know beforehand the number of common applications by retrieving the value of the data item TAG_COMMON_APP_NUMBER.
In the event of an application error that doesn't force to abort the transaction, this command will be called again as many times as necessary while the list won't be empty. Internally the Kernel will remove the wrong application so that the selection could be resumed.
Upon successful execution, EMV kernel status is stored in emvLastStatus property.

Parameters

confirmationRequired	- defines if USER Confirmation is required
error	returns error information, you can pass nil if you don't want it

Returns

Array of DTEMVApplication upon success, nil otherwise

2.22.2.4 - (BOOL) emvInitialise: (NSError ∗∗) *error*

This command initializes the emv kernel, call it before calling any other EMV function.

Note

Upon successful execution, EMV kernel status is stored in emvLastStatus property.

Parameters

error	returns error information, you can pass nil if you don't want it

Returns

TRUE upon success, FALSE otherwise

2.22.2.5 - (BOOL) emvLoadAppList: (NSArray ∗) *appList* selectionMethod:(APP_SELECTION_METHODS) *selectionMethod* includeBlockedAIDs:(BOOL) *includeBlockedAIDs* error:(NSError ∗∗) *error*

The command initiates the application selection process, loading the application list supported by the terminal.

The maximum number of application that can be loaded into the kernel is up to 75. This number is only constrained by the max packet size that can be exchanged on the port (2Kb).

Initially the command will inspect the incoming data to make sure that if data are provided and that all the data related to terminal applications is valid. If no data has been provided (the list is empty) the status EMV_NO_DATA_FOUND will be returned, in the event of a format failure of the applications data the result got will be EMV_DATA_FORMA-T_ERROR. If during the internal procedure of the commands a system error occurs the command will return with the status EMV_SYSTEM_ERROR, on the other hand if the error occurs dealing with the card or with the data got and the transaction must be aborted according to EMV specs, the result will be EMV_ABORT_TRANSACTION. If the process can be completed correctly and the list is properly parsed and managed the status SUCCESS will be returned.

Note

Upon successful execution, EMV kernel status is stored in emvLastStatus property.

Parameters

appList	- an array of application DTEMVApplication
selectionMethod	- defines the selection preferred method: SELECTION_PSE — Selection by PSE SELECTION_AIDLIST — Selection by AID list
includeBlocked-AIDs	- indicates if blocked AIDs should be included
error	returns error information, you can pass nil if you don't want it

Returns

TRUE upon success, FALSE otherwise

2.23 Transaction Processing

This section covers the different phases of the transaction:

Initial process

Data reading

Card data authentication

Restrictions processing

Risk Control

Cardholder authentication

Certificate generation

Make Transaction decision

Make default decision.

Functions

- (BOOL) - DTDevices::emvInitialAppProcessing:error:

  Once an application has been selected, the next phase is to start the transaction with it by issuing the GET PROCESSING ommand and analyzing the information got.
- (BOOL) - DTDevices::emvReadAppData:error:

  The command reads and validates the data informed in the AFL and that will be used along the transaction.
- (BOOL) - DTDevices::emvAuthentication:error:

  Through this command the card data is authenticated depending on the capabilities of the card and the kernel.
- (BOOL) - DTDevices::emvProcessRestrictions:

  The command performs the restrictions processing related to application version, application usage control and effective and expiry dates.
- (BOOL) - DTDevices::emvTerminalRisk:error:

  The application risk control is done by this command, including Floorlimit checking, Random selection (only if offline is enabled) and Velocity checking.
- (BOOL) - DTDevices::emvGetAuthenticationMethod:

  The command starts or resumes the cardholder authentication procedure, the current verification method is communicated to the application.
- (BOOL) - DTDevices::emvSetAuthenticationResult:error:

  Using this command the kernel gets the result of the previously informed verification method.
- (BOOL) - DTDevices::emvVerifyPinOffline:

  The command allows the application to apply the offline PIN verification (plaintext or encrypted) method.
- (BOOL) - DTDevices::emvGenerateCertificate:risk:error:

  Using this command the application will be able to generate an application cryptogram, the first or the second one, as required by the transaction.
- (BOOL) - DTDevices::emvMakeTransactionDecision:

  The command checks the action codes (provided by the application and read from the card), the TVR and will determine how the transaction is resolved.
- (BOOL) - DTDevices::emvMakeDefaultDecision:

  The command checks the default action code (provided by the application and read from the card), the TVR and will determine how the transaction is resolved by default.

2.23.1 Detailed Description

This section covers the different phases of the transaction:

Initial process

Data reading

Card data authentication

Restrictions processing

Risk Control

Cardholder authentication

Certificate generation

Make Transaction decision

Make default decision.

2.23.2 Function Documentation

2.23.2.1 - (BOOL) emvAuthentication: (BOOL) *checkAmount* error:(NSError $**$) *error*

Through this command the card data is authenticated depending on the capabilities of the card and the kernel.

The method could be static or dynamic, in this case is completed here, or combined that will be carried out later at the application cryptogram generation stage.

If the authentication can be performed (successfully or not) the command will return EMV_SUCCESS. If an internal error occurs the status got will be EMV_SYSTEM_ERROR. EMV_ABORT_TRANSACTION will be returned if the transaction must be immediately terminated due to a severe error in the processing. If the check amount flag was enabled and the amount is one of the data items requested by the dynamic data authentication the status EMV_A-MOUNT_NEEDED will be returned. If the authentication cannot be completed due to a missing CA public key, the status returned will be EMV_INVALID_KEY.

Note

EMV_INVALID_KEY status code will let the application to detect and invalid configuration concerning the CA RSA public keys.
If the selected authentication method is the CDA, the verification of the CA public key presence and the recovery of the issuer public key is done here prior to the actual CDA verification to be done at the AC generation.
The reason for setting the checkAmount parameter to TRUE is to allow the application to know if the amount is required as part of the dynamic data used for the authentication. This can be useful if the application plans to be sure that the actual amount will be used in the process rather than a default value set to zero.
Upon successful execution, EMV kernel status is stored in emvLastStatus property.

Parameters

checkAmount	- determine whether the amount is checked for the dynamic authentication
error	returns error information, you can pass nil if you don't want it

Returns

TRUE upon success, FALSE otherwise

2.23.2.2 - (BOOL) emvGenerateCertificate: (CERTIFICATE_AC_TYPES) *type* risk:(CARD_RISK_TYPES) *risk* error:(NSError $**$) *error*

Using this command the application will be able to generate an application cryptogram, the first or the second one, as required by the transaction.

If the incoming pointer to the structure with the parameters is NULL, the result set will be EMV_NO_DATA_FOUND. If any of the incoming parameters value is incorrect the status EMV_DATA_FORMAT_ERROR will be returned. E-MV_SYSTEM_ERROR will be get by the application if any internal error occurs during the processing. If during the cryptogram generation an error occurs that requires the transaction termination, the status EMV_ABORT_TRANSA-CTION will be informed. If other kind of error occurs during the generation the status EMV_ERROR_AC_PROCESS will be got. If the combined authentication is enabled, EMV_CDA_FAILED will be returned to indicate that it failed. Finally if the certificate can be obtained with no error the status will be EMV_SUCCESS.

Note

If the CDA is the card data authentication mode the CDA will be always requested on the first cryptogram generation if the cryptogram type to be requested is a TC. It will be always disabled for AAC and for an ARQC depends on the CDA mode active.
If the CDA is the card data authentication mode the CDA will be disabled on the second cryptogram generation if the cryptogram type to be requested is an AAC, otherwise if the cryptogram type is a TC it will depend on the CDA mode active.
Upon successful execution, EMV kernel status is stored in emvLastStatus property.

Parameters

type	- specifies AC type AAC:	
	CERTIFICATE_AAC	AAC
	CERTIFICATE_TC	TC
	CERTIFICATE_ARQC	ARQC
risk	- card risk:	
	CDOL_1	CDOL_1
	CDOL_2	CDOL_2
error	returns error information, you can pass nil if you don't want it	

Returns

TRUE upon success, FALSE otherwise

2.23.2.3 - (BOOL) emvGetAuthenticationMethod: (NSError ∗∗) *error*

The command starts or resumes the cardholder authentication procedure, the current verification method is communicated to the application.

The lists of methods and conditions is parsed and processed to identify what are the valid ones according to the kernel capabilities the possible methods available are: EMV_OFF_LINE_PIN_PLAIN, EMV_ONLINE_PIN, EMV_-OFFLINE_PIN_CIPHERED.

If during the process an internal error occurs the status EMV_SYSTEM_ERROR is returned, if the transaction has to be terminated the status EMV_ABORT_TRANSACTION will be returned. If there are not more valid methods to be applied the status EMV_AUTH_COMPLETED is set.

Note

If a combination of methods is required by the card, pin verification plus signature, the kernel directly checks if the latter is possible according to the capabilities, if so the former is informed otherwise the next entry in the list will be processed.
Upon successful execution, EMV kernel status is stored in emvLastStatus property.

Parameters

error	returns error information, you can pass nil if you don't want it

Returns

TRUE upon success, FALSE otherwise

2.23.2.4 - (BOOL) emvInitialAppProcessing: (NSData $*$) *aid* error:(NSError $**$) *error*

Once an application has been selected, the next phase is to start the transaction with it by issuing the GET PROC-ESSING ommand and analyzing the information got.

First the input data are checked, if empty the status EMV_NO_DATA_FOUND is returned, if the length of the AID is incorrect (greater than AID max length or less than TAG min length) the status got will be EMV_DATA_FORMA-T_ERROR. If any internal error occurs during the processing the status returned will be EMV_SYSTEM_ERROR. Depending on the application type or status the codes EMV_EASY_ENTRY_APP, EMV_INVALID_APPLICATION or EMV_BLOCKED_APPLICATION could be returned. If the transaction must be aborted due to a processing error with the card or with the data got from it the status returned will be EMV_ABORT_TRANSACTION. EMV_APPLIC-ATION_NOT_FOUND will be the status got if the AID provided cannot be found in the card. If everything is correct and the application can be initiated properly the status will be EMV_SUCCESS.

Note

At this point of the transaction it could be possible to resume the application selection by calling the ppEmvGet-CommonAppList command again, this will depend on the status got, normally for EMV_EASY_ENTRY_APP, EMV_INVALID_APPLICATION or EMV_BLOCKED_APPLICATION the selection should be resumed.
Upon successful execution, EMV kernel status is stored in emvLastStatus property.

Parameters

aid	- indicates the selected application AID
error	returns error information, you can pass nil if you don't want it

Returns

TRUE upon success, FALSE otherwise

2.23.2.5 - (BOOL) emvMakeDefaultDecision: (NSError $**$) *error*

The command checks the default action code (provided by the application and read from the card), the TVR and will determine how the transaction is resolved by default.

EMV_SYSTEM_ERROR will be returned if any internal error occurs during the processing. If any of the bits in the TVR match with the default action codes the status EMV_TRANSACTION_DENIED will be returned, otherwise the status will be EMV_TRANSACTION_APPROVED instead.

Note

Upon successful execution, EMV kernel status is stored in emvLastStatus property.

Parameters

error	returns error information, you can pass nil if you don't want it

Returns

TRUE upon success, FALSE otherwise

2.23.2.6 - (BOOL) emvMakeTransactionDecision: (NSError ∗∗) *error*

The command checks the action codes (provided by the application and read from the card), the TVR and will determine how the transaction is resolved.

EMV_SYSTEM_ERROR will be got by the application if any internal error occurs during the processing. First the denial action codes are checked, if any of the bits in the TVR match the status EMV_TRANSACTION_DENIED will be returned, otherwise if the terminal is both offline & online, the online action codes will be checked in the same way and if any of the bits match with the TVR data the status EMV_TRANSACTION_ONLINE will be set, if there's no match at all the status will be EMV_TRANSACTION_APPROVED instead. If the terminal is offline only the default action code is checked, if any of the bits in the TVR match the status EMV_TRANSACTION_DENIED will be returned, otherwise the status got will be EMV_TRANSACTION_APPROVED. If the terminal is online only the status EMV_TRANSACTION_ONLINE will be returned.

Note

According to the latest EMV recommendations concerning the CDA processing (Specification update bulletin No. 44) if the CDA is the card data authentication mechanism to be performed, the previous key recovery process will be accomplished prior to the transaction decision so that CDA errors could be detected in advance and reflected on the TVR.
The online/offline capability of the terminal is determined by the value of the tag TAG_TERMINAL_TYPE.
Upon successful execution, EMV kernel status is stored in emvLastStatus property.

Parameters

error	returns error information, you can pass nil if you don't want it

Returns

TRUE upon success, FALSE otherwise

2.23.2.7 - (BOOL) emvProcessRestrictions: (NSError ∗∗) *error*

The command performs the restrictions processing related to application version, application usage control and effective and expiry dates.

If the process can be completed correctly the returned status will be SUCCESS, if any internal error occurs the status will be EMV_SYSTEM_ERROR instead.

Note

To complete this process the kernel needs from the application the following data items to have been provided prior to this command: • TAG_APP_VERSION_NUMBER • TAG_TERMINAL_TYPE • TAG_ADD_TERM_CAPABILITIES • TAG_TERMINAL_COUNTRY_CODE • TAG_TRANSAC_DATE • TAG_TRANSAC_TYPE
Upon successful execution, EMV kernel status is stored in emvLastStatus property.

Parameters

error	returns error information, you can pass nil if you don't want it

Returns

TRUE upon success, FALSE otherwise

2.23.2.8 - (BOOL) emvReadAppData: (NSArray ∗) *tags* error:(NSError ∗∗) *error*

The command reads and validates the data informed in the AFL and that will be used along the transaction.

If during the AFL data reading and validating an error occurs that commits the transaction to be terminated, the status EMV_ABORT_TRANSACTION will be returned. If the error allows the application selection to be resumed, the status returned will be EMV_ERROR_IN_APPLICATION. If psrEMVManTagList is not NULL, the presence of the tags provided here will be checked. If during the procedure any internal error occurs, EMV_SYSTEM_ERROR will be returned. On the other hand, if everything is correct and the data can be extracted and validated, the status EMV_SUCCESS will be the value returned.

Note

At this point of the transaction it could be possible to resume the application selection by calling the ppEmvGet-CommonAppList command again, this will depend on the status got, normally for EMV_ERROR_IN_APPLIC-ATION the selection should be resumed.
Upon successful execution, EMV kernel status is stored in emvLastStatus property.

Parameters

tags	- an array of tags to return
error	returns error information, you can pass nil if you don't want it

Returns

TRUE upon success, FALSE otherwise

2.23.2.9 - (BOOL) emvSetAuthenticationResult: (AUTH_RESULTS) *result* error:(NSError ∗∗) *error*

Using this command the kernel gets the result of the previously informed verification method.

Firstly the value of the result informed must be checked, if its value is not a valid one the status EMV_DATA_FO-RMAT_ERROR will be returned. If the authentication process was not started and no method is currently active the status EMV_NO_CURRENT_METHOD will be got by the application, if the result for the current method was already provided the status will be EMV_RESULT_ALREADY_LOADED. EMV_SYSTEM_ERROR will be get by the application if any internal error occurs during the processing. When everything is ok and the result can be stored correctly the status sent back is EMV_SUCCESS.

Note

The actual verification method result according to EMV specs can be recovered by the application at later stage by accessing the data item TAG_CH_VERIF_METHOD_RESUL.
Upon successful execution, EMV kernel status is stored in emvLastStatus property.

Parameters

result	- result of the verification method previously informed:	
	AUTH_RESULT_SUCCESS	The method result was successful
	AUTH_RESULT_FAILURE	The method failed
	AUTH_FAIL_PIN_ENTRY_NOT_DONE	PIN entry was bypassed
	AUTH_FAIL_USER_CANCELLATION	PIN entry was cancelled
error	returns error information, you can pass nil if you don't want it	

Returns

TRUE upon success, FALSE otherwise

2.23.2.10 - (BOOL) emvTerminalRisk: (BOOL) *forceProcessing* error:(NSError ∗∗) *error*

The application risk control is done by this command, including Floorlimit checking, Random selection (only if offline is enabled) and Velocity checking.

If the process can be completed correctly the returned status will be SUCCESS, if any internal error occurs the status will be EMV_SYSTEM_ERROR instead.

Note

To complete this process the kernel needs from the application the following data items to have been provided previously: • TAG_RISK_AMOUNT (if offline enabled) • TAG_AMOUNT_AUTHORISED_BINARY (if online only) • TAG_FLOOR_LIMIT_CURRENCY (optional) • TAG_TERMINAL_FLOOR_LIMIT • TAG_THRESHOL-D_VALUE (if offline) • TAG_TARGET_PERCENTAGE (if offline) • TAG_MAX_TARGET_PERCENTAGE (if offline) • TAG_TRANSAC_CURR_CODE (optional)
Upon successful execution, EMV kernel status is stored in emvLastStatus property.

Parameters

forceProcessing	- determine whether the process should be carried out despite of the AIP configuration
error	returns error information, you can pass nil if you don't want it

Returns

TRUE upon success, FALSE otherwise

2.23.2.11 - (BOOL) emvVerifyPinOffline: (NSError ∗∗) *error*

The command allows the application to apply the offline PIN verification (plaintext or encrypted) method.

Depending on the current PIN entry type (plaintext or encrypted) is verified against the card, if the PIN is no valid and is rejected the status EMV_INVALID_PIN will be returned if more than one attempt is still available otherwise the status will be EMV_INVALID_PIN_LAST_ATTEMPT. If a severe error occurs so that the transaction should be terminated immediately, the status EMV_ABORT_TRANSACTION will be set. If any kind of internal error occurs during the processing, the status EMV_SYSTEM_ERROR will be returned. If the verification cannot be completed due to a missing CA public key, the status returned will be EMV_INVALID_KEY. Finally if the PIN is entered and verified correctly the status got will be EMV_SUCCESS.

Note

EMV_INVALID_KEY status code will let the application to detect and invalid configuration concerning the CA RSA public keys.
The PIN entry process will have to be accomplished by the application calling to the proper commands provided for that aim.
Upon successful execution, EMV kernel status is stored in emvLastStatus property.

Parameters

error	returns error information, you can pass nil if you don't want it

Returns

TRUE upon success, FALSE otherwise

2.24 Issuer Authentication

The commands listed here are intended to process the data coming from the issuer as part of the response to the online authorization request.

Functions

- (BOOL) - DTDevices::emvAuthenticateIssuer:

  The command is used to validate the cryptogram got from the issuer.
- (BOOL) - DTDevices::emvScriptProcessing:error:

  The script processing retrieved in the online authorization is handled by this command.

2.24.1 Detailed Description

The commands listed here are intended to process the data coming from the issuer as part of the response to the online authorization request.

2.24.2 Function Documentation

2.24.2.1 - (BOOL) emvAuthenticateIssuer: (NSError ∗∗) *error*

The command is used to validate the cryptogram got from the issuer.

If the issuer cryptogram was not set previously, the status EMV_NO_DATA_FOUND will be returned. If during the processing any internal error occurs, the status EMV_SYSTEM_ERROR will be set. If everything is ok and the cryptogram is verified, the result will be EMV_SUCCESS.

Note

The data item that the application has to provide to the kernel so that this command could be executed is: TAG_ISSUER_AUTH_DATA
Upon successful execution, EMV kernel status is stored in emvLastStatus property.

Parameters

error	returns error information, you can pass nil if you don't want it

Returns

TRUE upon success, FALSE otherwise

2.24.2.2 - (BOOL) emvScriptProcessing: (int) *type* error:(NSError ∗∗) *error*

The script processing retrieved in the online authorization is handled by this command.

First the presence of the script in the data repository is checked, if it's not present the status EMV_NO_SCRIPT_-LOADED is returned. If during the processing any internal error occurs the status EMV_SYSTEM_ERROR will be set. Once the script has been conveniently processed and issued to the card the status EMV_SUCCESS will be set.

Note

The script data should be provided to the kernel through the data item TAG_ISSUER_SCRIPTS, and after the processing is over the results can be recovered by accessing the data item TAG_ISSUER_SCRIPTS_RESULT.
The maximum length of the scripts supported is 256 bytes.
Upon successful execution, EMV kernel status is stored in emvLastStatus property.

Parameters

type	- script type to be processed:	
	0x71	SCRIPT_71
	0x72	SCRIPT_72
error	returns error information, you can pass nil if you don't want it	

Returns

TRUE upon success, FALSE otherwise

2.25 General Commands

These commands are not part of the basic transaction management but provide the kernel with more flexibility, and can be used by the application for its own particular requirements.

Functions

- (BOOL) - DTDevices::emvUpdateTVRByte:bit:value:error:

  The command allows modifying the TVR directly, setting or unsetting the desired bits.
- (BOOL) - DTDevices::emvUpdateTSIByte:bit:value:error:

  The command allows modifying the TSI directly, setting or unsetting the desired bits.
- (BOOL) - DTDevices::emvCheckTVRByte:bit:error:

  The command is intended to verify an individual bit within the TVR.
- (BOOL) - DTDevices::emvCheckTSIByte:bit:error:

  The command is intended to verify an individual bit within the TSI.
- (BOOL) - DTDevices::emvRemovePublicKey:RID:error:

  The command is intended to delete a given CA public key.

2.25.1 Detailed Description

These commands are not part of the basic transaction management but provide the kernel with more flexibility, and can be used by the application for its own particular requirements.

2.25.2 Function Documentation

2.25.2.1 - (BOOL) emvCheckTSIByte: (int) *byte* bit:(int) *bit* error:(NSError ∗∗) *error*

The command is intended to verify an individual bit within the TSI.

Initially the incoming parameters are validated to ensure that are pointing to a valid location within the TSI structure, if that's not the case the status EMV_DATA_FORMAT_ERROR will be returned. If during the processing any internal error occurs the status EMV_SYSTEM_ERROR will be set. EMV_SUCCESS will be returned if the given bit is set otherwise it will be EMV_FAILURE.

Note

The aim of this command is to let the application to achieve any additional procedure that could need as a particular requirement. Consult section List of TVR and TSI bits for a list of the bits.
Upon successful execution, EMV kernel status is stored in emvLastStatus property.

Parameters

byte	- defines the byte number. Accepted values are in the range [1..5]
bit	- defines the bit number. Accepted values are in the range [1..8]
error	returns error information, you can pass nil if you don't want it

Returns

TRUE upon success, FALSE otherwise

2.25.2.2 - (BOOL) emvCheckTVRByte: (int) *byte* bit:(int) *bit* error:(NSError ∗∗) *error*

The command is intended to verify an individual bit within the TVR.

Initially the incoming parameters are validated to ensure that are pointing to a valid location within the TVR structure, if that's not the case the status EMV_DATA_FORMAT_ERROR will be returned. If during the processing any internal error occurs the status EMV_SYSTEM_ERROR will be set. EMV_SUCCESS will be returned if the given bit is set otherwise it will be EMV_FAILURE.

Note

The aim of this command is to let the application to achieve any additional procedure that could need as a particular requirement. Consult section List of TVR and TSI bits for a list of the bits.
Upon successful execution, EMV kernel status is stored in emvLastStatus property.

Parameters

byte	- defines the byte number. Accepted values are in the range [1..5]
bit	- defines the bit number. Accepted values are in the range [1..8]
error	returns error information, you can pass nil if you don't want it

Returns

TRUE upon success, FALSE otherwise

2.25.2.3 - (BOOL) emvRemovePublicKey: (int) *caIndex* RID:(NSData ∗) *RID* error:(NSError ∗∗) *error*

The command is intended to delete a given CA public key.

If the input pointer is NULL the status returned will be EMV_NO_DATA_FOUND, if the key cannot be found the EMV_INVALID_KEY status will be got. If during the processing any internal error occurs the returned status will be EMV_SYSTEM_ERROR. Finally if the key can be deleted the status will be EMV_SUCCESS.

Note

Upon successful execution, EMV kernel status is stored in emvLastStatus property.

Parameters

RID	- holds the RID data (5 bytes)
caIndex	- certification authority public key index
error	returns error information, you can pass nil if you don't want it

Returns

TRUE upon success, FALSE otherwise

2.25.2.4 - (BOOL) emvUpdateTSIByte: (int) *byte* bit:(int) *bit* value:(int) *value* error:(NSError ∗∗) *error*

The command allows modifying the TSI directly, setting or unsetting the desired bits.

Initially the incoming parameters are validated to ensure that are pointing to a valid location within the TSI structure, if that's not the case the status EMV_DATA_FORMAT_ERROR will be returned. If during the processing any internal error occurs the status EMV_SYSTEM_ERROR will be set. EMV_SUCCESS will be returned if everything is correct and the TVR could be updated.

Note

The aim of this command is to let the application to achieve any additional procedure that could need as a particular requirement. Consult section List of TVR and TSI bits for a list of the bits.
Upon successful execution, EMV kernel status is stored in emvLastStatus property.

Parameters

byte	- defines the byte number to update. Accepted values are in the range [1..5]
bit	- defines the bit number to update. Accepted values are in the range [1..8]
value	- holds the new bit value [0..1]
error	returns error information, you can pass nil if you don't want it

Returns

TRUE upon success, FALSE otherwise

2.25.2.5 - (BOOL) emvUpdateTVRByte: (int) *byte* bit:(int) *bit* value:(int) *value* error:(NSError $**$) *error*

The command allows modifying the TVR directly, setting or unsetting the desired bits.

Initially the incoming parameters are validated to ensure that are pointing to a valid location within the TVR structure. If that's not the case, the status EMV_DATA_FORMAT_ERROR will be returned. If during the processing any internal error occurs, the status EMV_SYSTEM_ERROR will be set. EMV_SUCCESS will be returned if everything is correct and the TVR could be updated.

Note

The aim of this command is to let the application to achieve any additional procedure that could need as a particular requirement. Consult section List of TVR and TSI bits below for the complete list of the bits.
Upon successful execution, EMV kernel status is stored in emvLastStatus property.

Parameters

byte	- defines the byte number to update. Accepted values are in the range [1..5]
bit	- defines the bit number to update. Accepted values are in the range [1..8]
value	- holds the new bit value [0..1]
error	returns error information, you can pass nil if you don't want it

Returns

TRUE upon success, FALSE otherwise

2.26 Data Access

The commands described below are used to access the data items used by the kernel.

Functions

- (BOOL) - DTDevices::emvSetDataAsBinary:data:error:

  The command sets a data item with data in binary format (raw data).
- (BOOL) - DTDevices::emvSetDataAsString:data:error:

  The command sets a data item with data in string format.
- (NSData ∗) - DTDevices::emvGetDataAsBinary:error:

  The command gets a data item in binary format (raw data).
- (NSString ∗) - DTDevices::emvGetDataAsString:error:

  The command gets a data item in string format.
- (BOOL) - DTDevices::emvGetDataDetails:tagType:maxLen:currentLen:error:

  The command allows the application direct access to the data of a given item.
- (BOOL) - DTDevices::emvSetBypassMode:error:

  With this command is possible to setup the behavior of the KERNEL regarding the PIN based method bypass, so that only the current method will be bypassed or any other found later in the CVM list will be considered so as well.
- (BOOL) - DTDevices::emvSetTags:error:

  Loads multiple tags at the same time, this is much faster than calling them 1 by 1.
- (NSData ∗) - DTDevices::emvGetTags:error:

  Reads multiple tags at the same time, this is much faster than calling them 1 by 1.
- (NSData ∗) - DTDevices::emvGetTagsEncrypted3DES:keyID:uniqueID:error:

  Reads multiple tags at the same time and sends them encrypted, this is much faster than calling them 1 by 1.
- (NSData ∗) - DTDevices::emvGetTagsEncryptedDUKPT:keyID:uniqueID:error:

  Reads multiple tags at the same time and sends them encrypted, this is much faster than calling them 1 by 1.

2.26.1 Detailed Description

The commands described below are used to access the data items used by the kernel.

2.26.2 Function Documentation

2.26.2.1 - (NSData ∗) emvGetDataAsBinary: (uint32_t) *tagID* error:(NSError ∗∗) *error*

The command gets a data item in binary format (raw data).

If the length of the data item is greater than the length of the buffer requested the status EMV_INVALID_LENGTH will be set, in the case of not finding the requested item the status EMV_TAG_NOT_FOUND will be returned. After checking the item attributes, if the item cannot be read the returned status will be EMV_INVALID_TAG. If during the processing any internal error occurs the returned status will be EMV_SYSTEM_ERROR. Finally if everything is OK and the data can be extracted the status will be EMV_SUCCESS.

Note

Using this method there's no applicable conversion, so the data retrieved is in the format that corresponds to the data item. Consult section List of EMV tags for a list of the data items.
Upon successful execution, EMV kernel status is stored in emvLastStatus property.

Parameters

tagID	- holds the Tag Id of the data item
error	returns error information, you can pass nil if you don't want it

Returns

Tag value as data upon success, nil otherwise

2.26.2.2 - (NSString ∗) emvGetDataAsString: (uint32_t) *tagID* error:(NSError ∗∗) *error*

The command gets a data item in string format.

If the length of the data item is greater than the length of the buffer requested the status EMV_INVALID_LENGTH will be set, in the case of not finding the requested item the status EMV_TAG_NOT_FOUND will be returned. After checking the item attributes, if the item cannot be read the returned status will be EMV_INVALID_TAG. If during the processing any internal error occurs the returned status will be EMV_SYSTEM_ERROR. Finally if everything is OK and the data can be extracted the status will be EMV_SUCCESS.

Note

Using this method there's no applicable conversion, so the data retrieved is in the format that corresponds to the data item. Consult section List of EMV tags for a list of the data items.
Upon successful execution, EMV kernel status is stored in emvLastStatus property.

Parameters

tagID	- holds the Tag Id of the data item
error	returns error information, you can pass nil if you don't want it

Returns

Tag value as string upon success, nil otherwise

2.26.2.3 - (BOOL) emvGetDataDetails: (uint32_t) *tagID* tagType:(int ∗) *tagType* maxLen:(int ∗) *maxLen* currentLen:(int ∗) *currentLen* error:(NSError ∗∗) *error*

The command allows the application direct access to the data of a given item.

In the case of not finding the requested item the status EMV_TAG_NOT_FOUND will be returned. If during the processing any internal error occurs, the returned status will be EMV_SYSTEM_ERROR. Finally, if everything is OK and the attributes can be extracted, the status will be EMV_SUCCESS.

Warning

The aim of this command is to let the application a direct access to the already assigned buffers of the data items. This could be useful to save and to optimize memory usage. It can be also used to determine the presence of an item or to know its current length.

Note

Upon successful execution, EMV kernel status is stored in emvLastStatus property.

Parameters

tagID	- holds the Tag Id of the data item	
tagType	- returns the type of the tag:	
	TAG_TYPE_BINARY	Binary data
	TAG_TYPE_BCD	Numeric data (BCD)
	TAG_TYPE_STRING	String data
maxLen	- returns maximum length of the item	
currentLen	- returns current length of the item	
error	returns error information, you can pass nil if you don't want it	

Returns

TRUE upon success, FALSE otherwise

2.26.2.4 - (NSData ∗) emvGetTags: (NSData ∗) *tagList* error:(NSError ∗∗) *error*

Reads multiple tags at the same time, this is much faster than calling them 1 by 1.

Some sensitive tags can only be read encrypted.

Note

Upon successful execution, EMV kernel status is stored in emvLastStatus property.

Parameters

tagList	list of tags to read, the list follows the BER_TLV structure without having length and value, tags can be single or 2bytes
error	returns error information, you can pass nil if you don't want it

Returns

BER-TLV data containing tag-length-value or nil if function failed

2.26.2.5 - (NSData ∗) emvGetTagsEncrypted3DES: (NSData ∗) *tagList* keyID:(int) *keyID* uniqueID:(uint32_t) *uniqueID* error:(NSError ∗∗) *error*

Reads multiple tags at the same time and sends them encrypted, this is much faster than calling them 1 by 1.

Some sensitive tags can only be read encrypted.

Note

Upon successful execution, EMV kernel status is stored in emvLastStatus property.

Parameters

tagList	list of tags to read, the list follows the BER_TLV structure without having length and value, tags can be single or 2bytes
keyID	index of the key to use (1-49)
error	returns error information, you can pass nil if you don't want it

Returns

encrypted packet or nil if function failed. After decryption the data contains:

- random data (4 bytes)
- unique ID (4 bytes) - same ID you have sent to the function
- payload length (2 bytes) - length of the TLV block in BIG ENDIAN
- data (variable) - BER-TLV data, as per EMV books
- crc (2 bytes) - CRC16 CCIT on all the bytes before it
- padding (0-7 bytes) zeroes to pad the packet with

2.26.2.6 - (NSData ∗) emvGetTagsEncryptedDUKPT: (NSData ∗) *tagList* keyID:(int) *keyID* uniqueID:(uint32_t) *uniqueID* error:(NSError ∗∗) *error*

Reads multiple tags at the same time and sends them encrypted, this is much faster than calling them 1 by 1.

Some sensitive tags can only be read encrypted.

Note

Upon successful execution, EMV kernel status is stored in emvLastStatus property.

Parameters

tagList	list of tags to read, the list follows the BER_TLV structure without having length and value, tags can be single or 2bytes
keyID	index of the DUKPT key to use (0-1)
error	returns error information, you can pass nil if you don't want it

Returns

encrypted packet + DUKPT KSN (10 bytes) or nil if function failed. After decryption the data contains:

- random data (4 bytes)
- unique ID (4 bytes) - same ID you have sent to the function
- payload length (2 bytes) - length of the TLV block in BIG ENDIAN
- data (variable) - BER-TLV data, as per EMV books
- crc (2 bytes) - CRC16 CCIT on all the bytes before it
- padding (0-7 bytes) zeroes to pad the packet with

2.26.2.7 - (BOOL) emvSetBypassMode: (BYPASS_MODES) *mode* error:(NSError ∗∗) *error*

With this command is possible to setup the behavior of the KERNEL regarding the PIN based method bypass, so that only the current method will be bypassed or any other found later in the CVM list will be considered so as well.

If any kind of internal error occurs during the processing or the kernel was not initialized before the status EM-V_SYSTEM_ERROR will be returned. On the other hand if the value can be set correctly the status got will be EMV_SUCCESS.

Note

If this command is not used along the transaction the default value applied by the kernel will be BYPASS_CU-RRENT_METHOD_MODE. If the expected behavior is other than the default one the call to this command will have to be done prior to the cardholder authentication procedure and after application selection.
Upon successful execution, EMV kernel status is stored in emvLastStatus property.

Parameters

mode	- bypass mode, one of:
	BYPASS_CURRENT_METHOD_MODE: Bypass current method
	BYPASS_ALL_METHODS_MODE: Bypass all methods
error	returns error information, you can pass nil if you don't want it

Returns

TRUE upon success, FALSE otherwise

2.26.2.8 - (BOOL) emvSetDataAsBinary: (uint32_t) *tagID* data:(NSData ∗) *data* error:(NSError ∗∗) *error*

The command sets a data item with data in binary format (raw data).

Initially the input data is validated, if the buffer is NULL the status EMV_NO_DATA_FOUND will be returned, in case of not locating the tag EMV_TAG_NOT_FOUND will be set, if the length of the incoming data is not in the range accepted by the data item the status EMV_INVALID_LENGTH will be returned. The data item attributes are checked to determine whether the item can be written or not, if it's not the case the status returned will be EMV_INVALID_TAG. If during the processing any internal error occurs the returned status will be EMV_SYSTEM_ERROR. Once the data has been saved properly the status EMV_SUCCESS will be set.

Note

Using this method there's no applicable conversion, so the data provided should be in the format that corresponds to the data item to be set. So, in fact, it's like setting a given data item with raw data. Consult section List of EMV tags for a list of the data items.
Upon successful execution, EMV kernel status is stored in emvLastStatus property.

Parameters

tagID	- holds the Tag Id of the data item
data	- holds the Tag Data
error	returns error information, you can pass nil if you don't want it

Returns

TRUE upon success, FALSE otherwise

2.26.2.9 - (BOOL) emvSetDataAsString: (uint32_t) *tagID* data:(NSString ∗) *data* error:(NSError ∗∗) *error*

The command sets a data item with data in string format.

Initially the input data is validated, if the buffer is NULL the status EMV_NO_DATA_FOUND will be returned, in case of not locating the tag EMV_TAG_NOT_FOUND will be set, if the length of the incoming data is not in the range accepted by the data item the status EMV_INVALID_LENGTH will be returned. The data item attributes are checked to determine whether the item can be written or not, if it's not the case the status returned will be EMV_INVALID_TAG. If during the processing any internal error occurs the returned status will be EMV_SYSTEM_ERROR. Once the data has been saved properly the status EMV_SUCCESS will be set.

Note

Using this method there's no applicable conversion, so the data provided should be in the format that corresponds to the data item to be set. So, in fact, it's like setting a given data item with raw data. Consult section List of EMV tags for a list of the data items.
Upon successful execution, EMV kernel status is stored in emvLastStatus property.

Parameters

tagID	- holds the Tag Id of the data item
data	- holds the Tag Data
error	returns error information, you can pass nil if you don't want it

Returns

TRUE upon success, FALSE otherwise

2.26.2.10 - (BOOL) emvSetTags: (NSData ∗) *tlv* error:(NSError ∗∗) *error*

Loads multiple tags at the same time, this is much faster than calling them 1 by 1.

Note

Upon successful execution, EMV kernel status is stored in emvLastStatus property.

Parameters

tlv	BER-TLV lists ot tag-length-value, as described in EMV books
error	returns error information, you can pass nil if you don't want it

Returns

TRUE upon success, FALSE otherwise

2.27 User Interface Functions

This section includes functions for managing the display, reading PIN and keyboard.

Functions

- (BOOL) - DTDevices::uiGetScreenInfoWidth:height:colorMode:error:

    Returns screen properties.
- (BOOL) - DTDevices::uiDrawText:topLeftX:topLeftY:font:error:

    Disaplay some text, starting at a specified position.
- (BOOL) - DTDevices::uiFillRectangle:topLeftY:width:height:color:error:

    Fills rectangle on the screen with specified color.
- (BOOL) - DTDevices::uiSetContrast:error:

    Set display contrast.
- (BOOL) - DTDevices::uiPutPixel:y:color:error:

    Draws pixel on the screen with specified color.
- (BOOL) - DTDevices::uiDisplayImage:topLeftY:image:error:

    Displays image on the screen.
- (BOOL) - DTDevices::uiStartAnimation:topLeftX:topLeftY:animated:error:

    Draws predefined animation on the screen.
- (BOOL) - DTDevices::uiStopAnimation:error:

    Stops animation playback started with ppUiStartAnimation.
- (BOOL) - DTDevices::uiControlLEDsWithBitMask:error:

    Enables or disables controllable LEDs on the device based on bit mask.
- (BOOL) - DTDevices::uiEnableVibrationForTime:error:

    Activates vibration motor (if available) for a specific time.
- (BOOL) - DTDevices::uiEnableSpeaker:error:

    Enables or disables external speaker.
- (BOOL) - DTDevices::uiIsSpeakerEnabled:error:

    Returns the state of external speaker.

Properties

- int DTDevices::uiDisplayWidth

    Contains display width in pixels.
- int DTDevices::uiDisplayHeight

    Contains display height in pixels.
- BOOL DTDevices::uiDisplayAtBottom

    Contains display height in pixels.

2.27.1 Detailed Description

This section includes functions for managing the display, reading PIN and keyboard.

2.27.2 Function Documentation

2.27.2.1 - (BOOL) uiControlLEDsWithBitMask: (uint32_t) *mask* error:(NSError ∗∗) *error*

Enables or disables controllable LEDs on the device based on bit mask.

Parameters

mask	bit mask of the enabled LEDs, 1 means the bit will be lit, 0 - disabled
error	returns error information, you can pass nil if you don't want it

Returns

TRUE if function succeeded, FALSE otherwise

2.27.2.2 - (BOOL) uiDisplayImage: (int) *topLeftX* topLeftY:(int) *topLeftY* image:(UIImage ∗) *image* error:(NSError ∗∗) *error*

Displays image on the screen.

The image is dithered down to black and white before sending.

Parameters

topLeftX	- topleft X coordinate of the image in pixels
topLeftY	- topleft Y coordinate of the image in pixels
image	- image to draw
error	returns error information, you can pass nil if you don't want it

Returns

TRUE if function succeeded, FALSE otherwise

2.27.2.3 - (BOOL) uiDrawText: (NSString ∗) *text* topLeftX:(int) *topLeftX* topLeftY:(int) *topLeftY* font:(FONTS) *font* error:(NSError ∗∗) *error*

Disaplay some text, starting at a specified position.

The text can contain control symbols that alter cursor position, colors or whole window. Characters going outside the screen will not be drawn.

Parameters

text	- text string to write. Special codes that can be used are: 0x0A: newline (moves cursor at the beginning of the next line) 0x0B: turns on character inversion 0x0C: turns of character inversion
topLeftX	- topleft X coordinate in pixels
topLeftY	- topleft Y coordinate in pixels
font	font size, one of the FONT_∗ constants
error	returns error information, you can pass nil if you don't want it

0x0A	newline (moves cursor at the beginning of the next line)
0x0B	turns on character inversion
0x0C	turns of character inversion

Returns

TRUE if function succeeded, FALSE otherwise

2.27.2.4 - (BOOL) uiEnableSpeaker: (BOOL) *enabled* error:(NSError ∗∗) *error*

Enables or disables external speaker.

The speaker is active as long as the device controlling it is connected/awake, so if you want the speaker to be used in background, you have to set external accessory background mode in your application or use setAutoOffWhenIdle to set long standby time

Note

enabling external speaker consumes power for the amplifier, so in order to conserve battery, enable it only when needed

Parameters

enabled	TRUE if you want to enable the external speaker
error	returns error information, you can pass nil if you don’t want it

Returns

TRUE if function succeeded, FALSE otherwise

2.27.2.5 - (BOOL) uiEnableVibrationForTime: (float) *time* error:(NSError ∗∗) *error*

Activates vibration motor (if available) for a specific time.

Parameters

time	the maximum amount of time the vibration will be active
error	returns error information, you can pass nil if you don’t want it

Returns

TRUE if function succeeded, FALSE otherwise

2.27.2.6 - (BOOL) uiFillRectangle: (int) *topLeftX* topLeftY:(int) *topLeftY* width:(int) *width* height:(int) *height* color:(UIColor ∗) *color* error:(NSError ∗∗) *error*

Fills rectangle on the screen with specified color.

Parameters

topLeftX	- topleft X coordinate of the rectangle in pixels
topLeftY	- topleft Y coordinate of the rectangle in pixels
width	- rectangle width in pixels or 0 for automatic calculation
height	- rectangle height in pixels or 0 for automatic calculation
color	- the color to use, either COLOR_INVERT or custom UIColor
error	returns error information, you can pass nil if you don’t want it

Returns

TRUE if function succeeded, FALSE otherwise

2.27.2.7 - (BOOL) uiGetScreenInfoWidth: (int ∗) *width* height:(int ∗) *height* colorMode:(SCREEN_COLOR_MODES ∗) *colorMode* error:(NSError ∗∗) *error*

Returns screen properties.

Parameters

width	screen width in pixels will be returned here
height	screen height in pixels will be returned here
color	screen capability to display colors will be returned here, one of the COLOR_MODE_∗ constants
error	returns error information, you can pass nil if you don't want it

Returns

TRUE if function succeeded, FALSE otherwise

2.27.2.8 - (BOOL) uiIsSpeakerEnabled: (BOOL ∗) *enabled* error:(NSError ∗∗) *error*

Returns the state of external speaker.

Parameters

enabled	stores the current state of the external speaker, TRUE means it is enabled, FALSE - internal speaker is used
error	returns error information, you can pass nil if you don't want it

Returns

TRUE if function succeeded, FALSE otherwise

2.27.2.9 - (BOOL) uiPutPixel: (int) *x* y:(int) *y* color:(UIColor ∗) *color* error:(NSError ∗∗) *error*

Draws pixel on the screen with specified color.

Parameters

x	- X coordinate in pixels
y	- Y coordinate in pixels
color	- the color to use, either COLOR_INVERT or custom UIColor
error	returns error information, you can pass nil if you don't want it

Returns

TRUE if function succeeded, FALSE otherwise

2.27.2.10 - (BOOL) uiSetContrast: (int) *contrast* error:(NSError ∗∗) *error*

Set display contrast.

Parameters

contrast	- display contrast
error	returns error information, you can pass nil if you don't want it

Returns

TRUE if function succeeded, FALSE otherwise

2.27.2.11 - (BOOL) uiStartAnimation: (ANIMATIONS) *animationIndex* topLeftX:(int) *topLeftX* topLeftY:(int) *topLeftY* animated:(BOOL) *animated* error:(NSError $**$) *error*

Draws predefined animation on the screen.

You can have multiple animations active. Not all animations are present in every pinpad.

Parameters

animationIndex	- animation index, one of the ANIM_$*$ constants
topLeftX	- topleft X coordinate of the animation in pixels
topLeftY	- topleft Y coordinate of the animation in pixels
animated	- if TRUE, the animation will play continuous until stopped with ppUiStopAnimation
error	returns error information, you can pass nil if you don't want it

Returns

TRUE if function succeeded, FALSE otherwise

2.27.2.12 - (BOOL) uiStopAnimation: (ANIMATIONS) *animationIndex* error:(NSError $**$) *error*

Stops animation playback started with ppUiStartAnimation.

Parameters

animationIndex	- animation index, one of the ANIM_$*$ constants
error	returns error information, you can pass nil if you don't want it

Returns

TRUE if function succeeded, FALSE otherwise

2.28 Printing functions

Functions to print graphic, text and barcodes on supported printers.

Macros

- #define **CHANNEL_PRN** 1
- #define **CHANNEL_SMARTCARD** 2
- #define **CHANNEL_GPRS** 5
- #define **CHANNEL_ENCMSR** 14
- #define **CHANNEL_MIFARE** 16

Functions

- (BOOL) - DTDevices::prnFlushCache:

  Forces data still in the sdk buffers to be sent directly to the printer.
- (BOOL) - DTDevices::prnWriteDataToChannel:data:error:

  Sends data to the connected printer no matter the connection type.
- (NSData $*$) - DTDevices::prnReadDataFromChannel:length:timeout:error:

  Tries to read data from the connected remote device for specified timeout.
- (BOOL) - DTDevices::prnWaitPrintJob:error:

  Waits specified timeout for the printout to complete.
- (BOOL) - DTDevices::prnGetPrinterStatus:error:

  Retrieves current printer status.
- (BOOL) - DTDevices::prnSelfTest:error:

  Prints selftest.
- (BOOL) - DTDevices::prnTurnOff:

  Forces printer to turn off.
- (BOOL) - DTDevices::prnFeedPaper:error:

  Feeds the paper X lines (1/203 of the inch) or as needed (different length based on the printer model) so it allows paper to be teared.
- (BOOL) - DTDevices::prnPrintBarcode:barcode:error:

  Prints barcode.
- (BOOL) - DTDevices::prnPrintLogo:error:

  Prints the stored logo.
- (BOOL) - DTDevices::prnSetBarcodeSettings:height:hriPosition:align:error:

  Set various barcode parameters.
- (BOOL) - DTDevices::prnSetDensity:error:

  Sets printer density level.
- (BOOL) - DTDevices::prnSetLineSpace:error:

  Sets the line "height" in pixels If the characters are 16 pixelx high for example, setting the linespace to 20 will make the printer leave 4 blank lines before next line of text starts.
- (BOOL) - DTDevices::prnSetLeftMargin:error:

  Sets left margin.
- (BOOL) - DTDevices::prnPrintText:usingEncoding:error:

  Prints text with specified font/styles.
- (BOOL) - DTDevices::prnPrintText:error:

  Prints text with specified font/styles.
- (BOOL) - DTDevices::prnPrintDelimiter:error:

  Prints the delimiter character at the whole width of the paper, adjusting itself to the paper width.
- (BOOL) - DTDevices::prnGetBlackMarkTreshold:error:

Returns blackmark sensor treshold or UnsupportedOperationException if printer is not in blackmark mode.

- (BOOL) - DTDevices::prnSetBlackMarkTreshold:error:

  Sets blackmark sensor treshold or UnsupportedOperationException if printer is not in blackmark mode.
- (BOOL) - DTDevices::prnCalibrateBlackMark:error:

  Provides blackmark sensor calibration by scaning 200mm of paper for possible black marks and adjust the sensor treshold.
- (BOOL) - DTDevices::prnLoadLogo:align:error:

  Loads logo into printer's memory.
- (BOOL) - DTDevices::prnPrintImage:align:error:

  Prints Bitmap object using specified alignment.

2.28.1 Detailed Description

Functions to print graphic, text and barcodes on supported printers.

2.28.2 Function Documentation

2.28.2.1 - (BOOL) prnCalibrateBlackMark: (int ∗) *treshold* error:(NSError ∗∗) *error*

Provides blackmark sensor calibration by scaning 200mm of paper for possible black marks and adjust the sensor treshold.

Make sure you have put the right paper before calling this function.

Returns

returns new trashold value for the scanned paper. The trashold is already stored in printer's flash memory so no additional set is needed.

Parameters

treshold	upon sucess, the black mark treshold will be returned here
error	returns error information, you can pass nil if you don't want it

Returns

TRUE upon success, FALSE otherwise

2.28.2.2 - (BOOL) prnFeedPaper: (int) *lines* error:(NSError ∗∗) *error*

Feeds the paper X lines (1/203 of the inch) or as needed (different length based on the printer model) so it allows paper to be teared.

Note

If blackmark mode is active, this function searches for blackmark. If the paper is not blackmark one or the mark can not be found in 360mm, the printer will put itself into out of paper state and will need LF button to be pushed to continue.

Parameters

lines	the number of lines (1/203 of the inch) to feed or 0 to automatically feed the paper as much as needed to tear the paper.
error	returns error information, you can pass nil if you don't want it

Returns

TRUE upon success, FALSE otherwise

2.28.2.3 - (BOOL) prnFlushCache: (NSError ∗∗) *error*

Forces data still in the sdk buffers to be sent directly to the printer.

Parameters

error	returns error information, you can pass nil if you don't want it

Returns

TRUE upon success, FALSE otherwise

2.28.2.4 - (BOOL) prnGetBlackMarkTreshold: (int ∗) *treshold* error:(NSError ∗∗) *error*

Returns blackmark sensor treshold or UnsupportedOperationException if printer is not in blackmark mode.

This value tells the printer how dark a spot on the paper needs to be in order to be considered as blackmark.

Parameters

treshold	upon success stores the current blackmark treshold
error	returns error information, you can pass nil if you don't want it

Returns

TRUE upon success, FALSE otherwise

2.28.2.5 - (BOOL) prnGetPrinterStatus: (int ∗) *status* error:(NSError ∗∗) *error*

Retrieves current printer status.

This function is useful on printers having no automatic status notifications like DPP-250 and DPP-350.

Parameters

status	upon successful execution, printer status (one or more of the PRN_STAT_∗ constants) will be stored here
error	returns error information, you can pass nil if you don't want it

Returns

TRUE upon success, FALSE otherwise

2.28.2.6 - (BOOL) prnLoadLogo: (UIImage ∗) *logo* align:(int) *align* error:(NSError ∗∗) *error*

Loads logo into printer's memory.

The logo is persistent and can be deleted only if battery is removed

Parameters

logo	logo bitmap data
align	logo alignment, one of the ALIGN_∗ constants
error	returns error information, you can pass nil if you don't want it

Returns

TRUE upon success, FALSE otherwise

2.28.2.7 - (BOOL) prnPrintBarcode: (int) *bartype* barcode:(NSData ∗) *barcode* error:(NSError ∗∗) *error*

Prints barcode.

Parameters

bartype	Barcode type, one of the BAR_PRN_∗ constants
barcode	barcode data to be printed
error	returns error information, you can pass nil if you don't want it

Returns

TRUE upon success, FALSE otherwise

2.28.2.8 - (BOOL) prnPrintDelimiter: (char) *delimchar* error:(NSError ∗∗) *error*

Prints the delimiter character at the whole width of the paper, adjusting itself to the paper width.

The character is printed with font 12x24

Parameters

delimchar	character to print
error	returns error information, you can pass nil if you don't want it

Returns

TRUE upon success, FALSE otherwise

2.28.2.9 - (BOOL) prnPrintImage: (UIImage ∗) *image* align:(int) *align* error:(NSError ∗∗) *error*

Prints Bitmap object using specified alignment.

You can print color bitmaps, as they will be converted to black and white using error diffusion and dithering to achieve best results. On older devices this can take some time

Parameters

image	UIImage object
align	image alighment, one of the ALIGN_∗ constants
error	returns error information, you can pass nil if you don't want it

Returns

TRUE upon success, FALSE otherwise

2.28.2.10 - (BOOL) prnPrintLogo: (int) *mode* error:(NSError $**$) *error*

Prints the stored logo.

You can upload log with logo function

Parameters

mode	logo mode, one of the LOGO_$*$ constants
error	returns error information, you can pass nil if you don't want it

Returns

TRUE upon success, FALSE otherwise

2.28.2.11 - (BOOL) prnPrintText: (NSString $*$) *textString* error:(NSError $**$) *error*

Prints text with specified font/styles.

This function can act as both simple plain text printing and quite complex printing using internal tags to format the text. The function uses the currently font size and style (or default ones) as well as the aligning, however it allows modifications of them inside the text. Any modification of the settings using the tags will be reverted when function completes execution. For example if you have default font selected before using printText and set bold font inside, it will be reverted to plain when function completes. The tags are control commands used to modify the text printing parameters. They are surrounded by {} brackets. A list of all control tags follows:

- {==} - reverts all settings to their defaults. It includes font size, style, aligning
- {=Fx} - selects font size. x ranges from 0 to 1 as follows:
- 0: FONT_9X16 (hieroglyph characters are using the same width as height, i.e. 16x16)
- 1: FONT_12X24 (hieroglyph characters are using the same width as height, i.e. 24x24)
- {=L} - left text aligning
- {=C} - center text aligning
- {=R} - right text aligning
- {=Rx} - text rotation as follows:
- 0: not rotated
- 1: rotated 90 degrees
- 2: rotated 180 degrees
- {+/-B} - sets or unsets bold font style
- {+/-I} - sets or unsets italic font style
- {+/-U} - sets or unsets underline font style
- {+/-V} - sets or unsets inverse font style
- {+/-W} - sets or unsets text word-wrapping
- {+/-DW} - sets or unsets doubled font width
- {+/-DH} - sets or unsets doubled font height

An example of using tags "{=C}Plain centered text\n{=L}Left centered\n{+B}...bold...{-B}{+I}or ITALIC"

Parameters

textString	the text to print
error	returns error information, you can pass nil if you don't want it

Returns

TRUE upon success, FALSE otherwise

2.28.2.12 - (BOOL) prnPrintText: (NSString $*$) *textString* usingEncoding:(NSStringEncoding) *encoding* error:(NSError $**$) *error*

Prints text with specified font/styles.

This function can act as both simple plain text printing and quite complex printing using internal tags to format the text. The function uses the currently font size and style (or default ones) as well as the aligning, however it allows modifications of them inside the text. Any modification of the settings using the tags will be reverted when function completes execution. For example if you have default font selected before using printText and set bold font inside, it will be reverted to plain when function completes. The tags are control commands used to modify the text printing parameters. They are surrounded by {} brackets. A list of all control tags follows:

- {==} - reverts all settings to their defaults. It includes font size, style, aligning
- {=Fx} - selects font size. x ranges from 0 to 1 as follows:
- 0: FONT_9X16 (hieroglyph characters are using the same width as height, i.e. 16x16)
- 1: FONT_12X24 (hieroglyph characters are using the same width as height, i.e. 24x24)
- {=L} - left text aligning
- {=C} - center text aligning
- {=R} - right text aligning
- {=Rx} - text rotation as follows:
- 0: not rotated
- 1: rotated 90 degrees
- 2: rotated 180 degrees
- {+/-B} - sets or unsets bold font style
- {+/-I} - sets or unsets italic font style
- {+/-U} - sets or unsets underline font style
- {+/-V} - sets or unsets inverse font style
- {+/-W} - sets or unsets text word-wrapping
- {+/-DW} - sets or unsets doubled font width
- {+/-DH} - sets or unsets doubled font height

An example of using tags "{=C}Plain centered text\n{=L}Left centered\n{+B}...bold...{-B}{+I}or ITALIC"

Parameters

textString	the text to print
encoding	the encoding to use when converting the string to format suitable to the printer. Default encoding should be NSWindowsCP1252StringEncoding. Currently double-byte encodings like JIS are not supported.
error	returns error information, you can pass nil if you don't want it

Returns

TRUE upon success, FALSE otherwise

2.28.2.13 - (NSData ∗) prnReadDataFromChannel: (int) *channel* length:(int) *length* timeout:(double) *timeout* error:(NSError ∗∗) *error*

Tries to read data from the connected remote device for specified timeout.

Parameters

channel	one of the CHANNEL_∗ constants. Use CHANNEL_PRN for generic access to the printer. This parameter has only meaning in protocol mode.
length	maximum amount of bytes to wait for
timeout	maximim timeout in seconds to wait for data
error	pointer to NSError object, where error information is stored in case function fails. You can pass nil if you don't want that information

Returns

actual data being read or nil if error occured

2.28.2.14 - (BOOL) prnSelfTest: (BOOL) *longtest* error:(NSError ∗∗) *error*

Prints selftest.

Parameters

longtest	TRUE if you want complete test with fonts and codepage, FALSE for short one
error	returns error information, you can pass nil if you don't want it

Returns

TRUE upon success, FALSE otherwise

2.28.2.15 - (BOOL) prnSetBarcodeSettings: (int) *scale* height:(int) *height* hriPosition:(int) *hriPosition* align:(int) *align* error:(NSError ∗∗) *error*

Set various barcode parameters.

Parameters

scale	width of each barcode column in pixels (1/203 of the inch) between 2 and 4, default is 3
height	barcode height in pixels between 1 and 255. Default is 77
hriPosition	barcode hri code position, one of the BAR_TEXT_∗ constants
align	barcode aligning, one of the ALIGN_∗ constants
error	returns error information, you can pass nil if you don't want it

Returns

TRUE upon success, FALSE otherwise

2.28.2.16 - (BOOL) prnSetBlackMarkTreshold: (int) *treshold* error:(NSError $**$) *error*

Sets blackmark sensor treshold or UnsupportedOperationException if printer is not in blackmark mode.

This value tells the printer how dark a spot on the paper needs to be in order to be considered as blackmark.

Parameters

treshold	value between 0x20 and 0xc0, default is 0x68

Exceptions

NSPortTimeoutException	if there is no connection to the printer

2.28.2.17 - (BOOL) prnSetDensity: (int) *percent* error:(NSError $**$) *error*

Sets printer density level.

Parameters

percent	density level in percents (50%-200%)
error	returns error information, you can pass nil if you don't want it

Returns

TRUE upon success, FALSE otherwise

2.28.2.18 - (BOOL) prnSetLeftMargin: (int) *leftMargin* error:(NSError $**$) *error*

Sets left margin.

Parameters

leftMargin	left margin in pixels. Default is 0
error	returns error information, you can pass nil if you don't want it

Returns

TRUE upon success, FALSE otherwise

2.28.2.19 - (BOOL) prnSetLineSpace: (int) *lineSpace* error:(NSError $**$) *error*

Sets the line "height" in pixels If the characters are 16 pixelx high for example, setting the linespace to 20 will make the printer leave 4 blank lines before next line of text starts.

You cannot make text lines overlap.

Parameters

lineSpace	linespace in pixels, or 0 for automatic calculation. Default is 0
error	returns error information, you can pass nil if you don't want it

Returns

TRUE upon success, FALSE otherwise

2.28.2.20 - (BOOL) prnTurnOff: (NSError ∗∗) *error*

Forces printer to turn off.

Parameters

error	returns error information, you can pass nil if you don't want it

Returns

TRUE upon success, FALSE otherwise

2.28.2.21 - (BOOL) prnWaitPrintJob: (NSTimeInterval) *timeout* error:(NSError ∗∗) *error*

Waits specified timeout for the printout to complete.

It is best to call this function with the complete timeout you are willing to wait, rather than calling it in a loop

Parameters

timeout	the timeout to wait for the job to finish
error	returns error information, you can pass nil if you don't want it

Returns

TRUE if printer have successfully finished printing and ready to accept new data, FALSE if communication problem or the printer is still busy

2.28.2.22 - (BOOL) prnWriteDataToChannel: (int) *channel* data:(NSData ∗) *data* error:(NSError ∗∗) *error*

Sends data to the connected printer no matter the connection type.

This also handles the internal packet mode, so only the payload needs to be send.

Parameters

channel	one of the CHANNEL_∗ constants. Use CHANNEL_PRN for generic access to the printer. This parameter has only meaning in protocol mode.
data	data bytes to write
error	pointer to NSError object, where error information is stored in case function fails. You can pass nil if you don't want that information

Returns

TRUE if function succeeded, FALSE otherwise

2.29 Printing Page Mode Functions

Functions to work with the printer's page mode.

Functions

- (BOOL) - DTDevices::pageIsSupported

  Returns TRUE if page mode is supported on the connected device.
- (BOOL) - DTDevices::pageStart:

  Creates a new virtual page using the maximum supported page height.
- (BOOL) - DTDevices::pagePrint:

  Prints the content of the virtual page.
- (BOOL) - DTDevices::pageEnd:

  Exits page mode.
- (BOOL) - DTDevices::pageSetWorkingArea:top:width:height:error:

  Sets a working area and orientation inside the virtual page.
- (BOOL) - DTDevices::pageSetWorkingArea:top:width:heigth:orientation:error:

  Sets a working area and orientation inside the virtual page.
- (BOOL) - DTDevices::pageFillRectangle:error:

  Fills the current working area (or whole page if none is set) with the specified color.
- (BOOL) - DTDevices::pageFillRectangle:top:width:height:color:error:

  Fills a rectangle inside the current working area with specified color.
- (BOOL) - DTDevices::pageRectangleFrame:top:width:height:framewidth:color:error:

  Draws a rectangle frame inside the current working area with specified color.

2.29.1 Detailed Description

Functions to work with the printer's page mode. Page mode is a special operation mode, that allows you to define a virtual page and then draw inside text, graphics, barcodes and print it all at once. Page mode allows for extended positioning of the elements, rotation, inversion and basic graphics elements.

2.29.2 Function Documentation

2.29.2.1 - (BOOL) pageEnd: (NSError ∗∗) *error*

Exits page mode.

Parameters

error	returns error information, you can pass nil if you don't want it

Returns

TRUE upon success, FALSE otherwise

2.29.2.2 - (BOOL) pageFillRectangle: (UIColor ∗) *color* error:(NSError ∗∗) *error*

Fills the current working area (or whole page if none is set) with the specified color.

Parameters

color	- the color to use, either COLOR_INVERT or custom UIColor
error	returns error information, you can pass nil if you don't want it

Returns

TRUE upon success, FALSE otherwise

2.29.2.3 - (BOOL) pageFillRectangle: (int) *left* top:(int) *top* width:(int) *width* height:(int) *height* color:(UIColor ∗) *color* error:(NSError ∗∗) *error*

Fills a rectangle inside the current working area with specified color.

Parameters

left,top,width,height	rectangle coordinates
color	- the color to use, either COLOR_INVERT or custom UIColor
error	returns error information, you can pass nil if you don't want it

Returns

TRUE upon success, FALSE otherwise

2.29.2.4 - (BOOL) pagePrint: (NSError ∗∗) *error*

Prints the content of the virtual page.

Note

The white space from the top and bottom is not printed so the print ends at the last black dot. If you want to feed the paper use the error:(NSError ∗∗)error function

Parameters

error	returns error information, you can pass nil if you don't want it

Returns

TRUE upon success, FALSE otherwise

2.29.2.5 - (BOOL) pageRectangleFrame: (int) *left* top:(int) *top* width:(int) *width* height:(int) *height* framewidth:(int) *framewidth* color:(UIColor ∗) *color* error:(NSError ∗∗) *error*

Draws a rectangle frame inside the current working area with specified color.

Parameters

left,top,width,height	rectangle coordinates
framewidth	width of the frame (1-64)
color	- the color to use, either COLOR_INVERT or custom UIColor
error	returns error information, you can pass nil if you don't want it

Returns

TRUE upon success, FALSE otherwise

2.29.2.6 - (BOOL) pageSetWorkingArea: (int) *left* top:(int) *top* width:(int) *width* height:(int) *height* error:(NSError ∗∗) *error*

Sets a working area and orientation inside the virtual page.

No drawing can ever occur outside the said area

Parameters

left,top,width,height	working area rectangle in absolute pixels (i.e. does not depend on the page orientation)
error	returns error information, you can pass nil if you don't want it

Returns

TRUE upon success, FALSE otherwise

2.29.2.7 - (BOOL) pageSetWorkingArea: (int) *left* top:(int) *top* width:(int) *width* heigth:(int) *height* orientation:(int) *orientation* error:(NSError ∗∗) *error*

Sets a working area and orientation inside the virtual page.

No drawing can ever occur outside the said area

Parameters

left,top,width,height	working area rectangle in absolute pixels (i.e. does not depend on the page orientation)
orientation	one of the PAGE_∗ constants
error	returns error information, you can pass nil if you don't want it

Returns

TRUE upon success, FALSE otherwise

2.29.2.8 - (BOOL) pageStart: (NSError ∗∗) *error*

Creates a new virtual page using the maximum supported page height.

Use getInfo:(int)infocmd to get the maximum page height supported. See pageStart for more detailed information The page mode allows constructing a virtual page inside the printer, draw text, graphics, and performs some basic graphics operations (draw rectangles, frames, invert parts of the page) at any place, rotated or not, then print the result. Page mode is useful if you want to create some non-standart printout, or print vertically. Tables functions also work in page mode allowing a huge tables to be created and printed vertically.

Parameters

error	returns error information, you can pass nil if you don't want it

Returns

TRUE upon success, FALSE otherwise

2.30 Printing Table Functions

Functions to create, fill and print tables.

Functions

- (BOOL) - DTDevices::tableIsSupported

  Checks if the currently connected printer supports tables.
- (BOOL) - DTDevices::tableCreate:error:

  Create a new table using custom flags.
- (BOOL) - DTDevices::tableCreate:

  Create a new table using default settings - both horizontal and vertical borders around it.
- (BOOL) - DTDevices::tableAddColumn:

  Adds a new column using default settings - 12x24 font, plain, vertical border between the cells, left aligning.
- (BOOL) - DTDevices::tableAddColumn:error:

  Adds a new column using default settings - plain text, vertical border between the cells, left aligning.
- (BOOL) - DTDevices::tableAddColumn:style:alignment:error:

  Adds a new column using custom font and vertical border between the cells.
- (BOOL) - DTDevices::tableAddColumn:style:alignment:flags:error:

  Adds a new column.
- (BOOL) - DTDevices::tableAddCell:error:

  Adds a new cell using the font size and style and aligning of the column that cell belongs to.
- (BOOL) - DTDevices::tableAddCell:font:error:

  Adds a new cell using the font style and aligning of the column that cell belongs to.
- (BOOL) - DTDevices::tableAddCell:font:style:error:

  Adds a new cell using custom font size and style and aligning of the column that cell belongs to.
- (BOOL) - DTDevices::tableAddCell:font:style:alignment:error:

  Adds a new cell using custom font size and style and aligning.
- (BOOL) - DTDevices::tableAddDelimiter:

  Adds aa horizontal black line to the entire row that separates it from the next.
- (BOOL) - DTDevices::tableSetRowHeight:error:

  Sets the row height that will be used by default for new cells added.
- (BOOL) - DTDevices::tablePrint:

  Prints current table or throws IllegalArgumentException if cell data cannot be fit into paper.

2.30.1 Detailed Description

Functions to create, fill and print tables.

2.30.2 Function Documentation

2.30.2.1 - (BOOL) tableAddCell: (NSString $*$) *data* error:(NSError $**$) *error*

Adds a new cell using the font size and style and aligning of the column that cell belongs to.

Parameters

data	string data
error	returns error information, you can pass nil if you don't want it

Returns

TRUE upon success, FALSE otherwise

2.30.2.2 - (BOOL) tableAddCell: (NSString ∗) *data* font:(int) *font* error:(NSError ∗∗) *error*

Adds a new cell using the font style and aligning of the column that cell belongs to.

Parameters

data	string data
font	font size, one of the FONT_size constants
error	returns error information, you can pass nil if you don't want it

Returns

TRUE upon success, FALSE otherwise

2.30.2.3 - (BOOL) tableAddCell: (NSString ∗) *data* font:(int) *font* style:(int) *style* alignment:(int) *alignment* error:(NSError ∗∗) *error*

Adds a new cell using custom font size and style and aligning.

Parameters

data	string data
font	font size, one of the FONT_size constants
style	one or more of the font style constants (FONT_BOLD, FONT_ITALIC, etc)
alignment	date aligning, one of the ALIGN_∗ constants
error	returns error information, you can pass nil if you don't want it

Returns

TRUE upon success, FALSE otherwise

2.30.2.4 - (BOOL) tableAddCell: (NSString ∗) *data* font:(int) *font* style:(int) *style* error:(NSError ∗∗) *error*

Adds a new cell using custom font size and style and aligning of the column that cell belongs to.

Parameters

data	string data
font	font size, one of the FONT_size constants
style	one or more of the font style constants (FONT_BOLD, FONT_ITALIC, etc)
error	returns error information, you can pass nil if you don't want it

Returns

TRUE upon success, FALSE otherwise

2.30.2.5 - (BOOL) tableAddColumn: (NSError ∗∗) *error*

Adds a new column using default settings - 12x24 font, plain, vertical border between the cells, left aligning.

Parameters

error	returns error information, you can pass nil if you don't want it

Returns

TRUE upon success, FALSE otherwise

2.30.2.6 - (BOOL) tableAddColumn: (int) *font* error:(NSError ∗∗) *error*

Adds a new column using default settings - plain text, vertical border between the cells, left aligning.

Parameters

font	one of the FONT_size constants
error	returns error information, you can pass nil if you don't want it

Returns

TRUE upon success, FALSE otherwise

2.30.2.7 - (BOOL) tableAddColumn: (int) *font* style:(int) *style* alignment:(int) *alignment* error:(NSError ∗∗) *error*

Adds a new column using custom font and vertical border between the cells.

Parameters

font	one of the FONT_size constants
style	one or more of the font style constants (FONT_BOLD, FONT_ITALIC, etc)
alignment	text alignment inside the cell, one of the ALIGN_∗ constants
error	returns error information, you can pass nil if you don't want it

Returns

TRUE upon success, FALSE otherwise

2.30.2.8 - (BOOL) tableAddColumn: (int) *font* style:(int) *style* alignment:(int) *alignment* flags:(int) *flags* error:(NSError ∗∗) *error*

Adds a new column.

Parameters

font	one of the FONT_size constants
style	one or more of the font style constants (FONT_BOLD, FONT_ITALIC, etc)
alignment	text alignment inside the cell, one of the ALIGN_∗ constants
flags	one or more of the TABLE_BORDERS_∗ constants
error	returns error information, you can pass nil if you don't want it

Returns

TRUE upon success, FALSE otherwise

2.30.2.9 - (BOOL) tableAddDelimiter: (NSError ∗∗) *error*

Adds aa horizontal black line to the entire row that separates it from the next.

Parameters

error	returns error information, you can pass nil if you don't want it

Returns

TRUE upon success, FALSE otherwise

2.30.2.10 - (BOOL) tableCreate: (NSError ∗∗) *error*

Create a new table using default settings - both horizontal and vertical borders around it.

Parameters

error	returns error information, you can pass nil if you don't want it

Returns

TRUE upon success, FALSE otherwise

2.30.2.11 - (BOOL) tableCreate: (int) *flags* error:(NSError ∗∗) *error*

Create a new table using custom flags.

Parameters

flags	one or more of the TABLE_BORDERS_∗ constants
error	returns error information, you can pass nil if you don't want it

Returns

TRUE upon success, FALSE otherwise

2.30.2.12 - (BOOL) tableIsSupported

Checks if the currently connected printer supports tables.

Returns

TRUE if tables are supported

2.30.2.13 - (BOOL) tablePrint: (NSError ∗∗) *error*

Prints current table or throws IllegalArgumentException if cell data cannot be fit into paper.

Parameters

error	returns error information, you can pass nil if you don't want it

Returns

TRUE upon success, FALSE otherwise

2.30.2.14 - (BOOL) tableSetRowHeight: (int) *height* error:(NSError $**$) *error*

Sets the row height that will be used by default for new cells added.

Parameters

height	row height, any value less than the characters height will be auto fixed. Default is LINESPAC-E_DEFAULT
error	returns error information, you can pass nil if you don't want it

Returns

TRUE upon success, FALSE otherwise

Chapter 3

Class Documentation

3.1 DTCAKeyInfo Class Reference

Information about Certification Authority keys.

Inherits NSObject.

Properties

- int keyIndex

  Key index.
- NSData $*$ RIDI

  RIDI.
- int moduleLength

  Length of the key module.

3.1.1 Detailed Description

Information about Certification Authority keys.

3.2 $<$DTDeviceDelegate$>$ Protocol Reference

Protocol describing various notifications that DTDevices SDK can send.

Instance Methods

- (void) - connectionState:

  Notifies about the current connection state.
- (void) - deviceButtonPressed:

  Notification sent when some of the device's buttons is pressed.
- (void) - deviceButtonReleased:

  Notification sent when some of the device's buttons is released.
- (void) - barcodeData:type:

  Notification sent when barcode is successfuly read.
- (void) - barcodeData:isotype:

  Notification sent when barcode is successfuly read.

- (void) - barcodeNSData:type:

    Notification sent when barcode is successfuly read.
- (void) - barcodeNSData:isotype:

    Notification sent when barcode is successfuly read.
- (void) - magneticCardData:track2:track3:

    Notification sent when magnetic card is successfuly read.
- (void) - magneticCardEncryptedData:tracks:data:

    Notification sent when magnetic card is successfuly read.
- (void) - magneticCardEncryptedData:tracks:data:track1masked:track2masked:track3:

    Notification sent when magnetic card is successfuly read.
- (void) - magneticCardEncryptedData:tracks:data:track1masked:track2masked:track3:source:

    Notification sent when magnetic card is successfuly read.
- (void) - magneticCardRawData:

    Notification sent when magnetic card is successfuly read.
- (void) - magneticCardEncryptedRawData:data:

    Notification sent when magnetic card is successfuly read.
- (void) - firmwareUpdateProgress:percent:

    Notification sent when firmware update process advances.
- (void) - bluetoothDiscoverComplete:

    Notification sent when bluetooth discovery finds new bluetooth device.
- (void) - bluetoothDeviceDiscovered:name:

    Notification sent when bluetooth discovery finds new bluetooth device.
- (void) - bluetoothDeviceConnected:

    Notification sent when bluetooth device is connected.
- (void) - bluetoothDeviceDisconnected:

    Notification sent when bluetooth connection is lost.
- (BOOL) - bluetoothDeviceRequestedConnection:name:

    Notification sent when a bluetooth device requests.
- (NSString ∗) - bluetoothDevicePINCodeRequired:name:

    Notification sent when a bluetooth device requests.
- (void) - magneticJISCardData:

    Notification sent when JIS I & II magnetic card is successfuly read.
- (void) - rfCardDetected:info:

    Notification sent when a new supported RFID card enters the field.
- (void) - rfCardRemoved:

    Notification sent when the card leaves the field.
- (void) - deviceFeatureSupported:value:

    Notification sent when some of the features gets enabled or disabled.
- (void) - smartCardInserted:

    Notification sent when smartcard was inserted.
- (void) - smartCardRemoved:

    Notification sent when smartcard was removed.
- (void) - PINEntryCompleteWithError:

    Notification sent when PIN entry procedure have completed or was cancelled.
- (void) - paperStatus:

    Notification sent when printer's paper sensor changes.
- (void) - sdkDebug:source:

    Notification sent to display debug messages from the sdk or device.
- (void) - emv2OnTransactionStarted

    Notification sent when EMV kernel detects a card and start processing it.
- (void) - emv2OnUserInterfaceCode:status:holdTime:

Notification sent when the EMV kernel wants to update the user interface.

- (void) - emv2OnApplicationSelection:

  Notification sent when the card has multiple applications and one needs to be selected.
- (void) - emv2OnOnlineProcessing:

  Notification sent when the kernel and the card require online processing.
- (void) - emv2OnTransactionFinished:

  Notification sent when the transaction is complete.

3.2.1 Detailed Description

Protocol describing various notifications that DTDevices SDK can send.

3.3 DTDevices Class Reference

Provides universal access to all supported devices' functions.

Inherits NSObject.

Public Types

- enum **APP_SELECTION_METHODS** { **SELECTION_PSE** =0, **SELECTION_AIDLIST** }
- enum **APP_MATCH_CRITERIAS** { **MATCH_FULL** =1, **MATCH_PARTIAL_VISA**, **MATCH_PARTIAL_EUR-OPAY** }
- enum **AUTH_RESULTS** { **AUTH_RESULT_SUCCESS** =1, **AUTH_RESULT_FAILURE**, **AUTH_FAIL_PIN_-ENTRY_NOT_DONE**, **AUTH_FAIL_USER_CANCELLATION** }
- enum **BYPASS_MODES** { **BYPASS_CURRENT_METHOD_MODE** =0, **BYPASS_ALL_METHODS_MODE** }
- enum **CERTIFICATE_AC_TYPES** { **CERTIFICATE_AAC** =0, **CERTIFICATE_TC**, **CERTIFICATE_ARQC** }
- enum **CARD_RISK_TYPES** { **CDOL_1** =1, **CDOL_2** }
- enum **TAG_TYPES** { **TAG_TYPE_BINARY** =0, **TAG_TYPE_BCD**, **TAG_TYPE_STRING** }

Instance Methods

- (void) - addDelegate:

  Allows unlimited delegates to be added to a single class instance.
- (void) - removeDelegate:

  Removes delegate, previously added with addDelegate.
- (void) - connect

  Tries to connect to supported devices in the background, connection status notifications will be passed through the delegate.
- (void) - disconnect

  Stops the sdk from trying to connect to supported devices and breaks existing connections.
- (BOOL) - **isPresent:**
- (BOOL) - setActiveDeviceType:error:

  The sdk can work with many devices at the same time, but some functions can be executed on a single device at a time (for example barcodeStartScan), this function sets the prefered device to execute the function by type.
- (BOOL) - setAutoOffWhenIdle:whenDisconnected:error:

  Sets the time in seconds, after which Linea will shut down to conserve battery.
- (BOOL) - getBatteryCapacity:voltage:error:

  Returns active device's battery capacity.
- (BOOL) - playSound:beepData:length:error:

Plays a sound using the built-in speaker on the active device.

- (BOOL) - getCharging:error:

  Returns if the connected device is charging the iOS device from it's own battery.
- (BOOL) - setCharging:error:

  Enables or disables Lines's capability to charge the handheld from it's own battery.
- (BOOL) - getPassThroughSync:error:

  Returns the current state of the pass-through synchronization.
- (BOOL) - setPassThroughSync:error:

  Enables or disables pass-through synchronization when you plug usb cable.
- (BOOL) - setUSBChargeCurrent:error:

  Sets the charge current that lightning connector based Lineas will allow the iPod/iPhone/iPad to be charged with when connected via USB port.
- (NSDictionary ∗) - getFirmwareFileInformation:error:

  Returns information about the specified firmware data.
- (BOOL) - updateFirmwareData:error:

  Updates connected device's firmware with specified firmware data.
- (int) - getSupportedFeature:error:

  Returns if a feature is supported on connected device(s) and what type it is.
- (BOOL) - msEnable:

  Enables reading of magnetic cards.
- (BOOL) - msDisable:

  Disables magnetic card reading.
- (NSDictionary ∗) - msProcessFinancialCard:track2:

  Helper function to parse financial card and extract the data - name, number, expiration date.
- (BOOL) - msSetCardDataMode:error:

  Sets Linea's magnetic card data mode.
- (NSString ∗) - barcodeType2Text:

  Helper function to return string name of barcode type.
- (BOOL) - barcodeStartScan:

  Starts barcode engine.
- (BOOL) - barcodeStopScan:

  Stops ongoing scan started with startScan.
- (BOOL) - barcodeGetScanButtonMode:error:

  Returns the current scan button mode.
- (BOOL) - barcodeSetScanButtonMode:error:

  Sets scan button mode.
- (BOOL) - barcodeSetScanBeep:volume:beepData:length:error:

  Sets the sound, which is used upon successful barcode scan.
- (BOOL) - barcodeGetScanMode:error:

  Returns the current scan mode.
- (BOOL) - barcodeSetScanMode:error:

  Sets barcode engine scan mode.
- (BOOL) - barcodeGetTypeMode:error:

  Returns the current barcode type mode.
- (BOOL) - barcodeSetTypeMode:error:

  Sets barcode type mode.
- (BOOL) - barcodeEngineResetToDefaults:

  Performs factory reset of the barcode module.
- (BOOL) - barcodeEngineCheckReady:error:

  Performs a check if the barcode engine is ready to operate.
- (BOOL) - barcodeOpticonSetInitString:error:

Allows for a custom initialization string to be sent to the Opticon barcode engine.

- (BOOL) - barcodeOpticonSetParams:saveToFlash:error:

  Sends configuration parameters directly to the opticon barcode engine.
- (NSString ∗) - barcodeOpticonGetIdent:

  Reads barcode engine's identification.
- (BOOL) - barcodeOpticonUpdateFirmware:bootLoader:error:

  Performs firmware update on the optiocon 2D barcode engines.
- (BOOL) - barcodeCodeSetParam:value:error:

  Sends configuration parameters directly to the code barcode engine.
- (BOOL) - barcodeCodeGetParam:value:error:

  Reads configuration parameters directly from the code barcode engine.
- (BOOL) - barcodeCodeUpdateFirmware:data:error:

  Performs firmware update on the Code 2D barcode engines.
- (NSDictionary ∗) - **barcodeCodeGetInformation:**
- (BOOL) - barcodeIntermecSetInitData:error:

  Allows for a custom initialization string to be sent to the Intermec barcode engine.
- (NSData ∗) - barcodeNewlandQuery:error:

  Sends a custom command to the barcode engine and receives a reply.
- (BOOL) - barcodeNewlandSetInitString:error:

  Allows for a custom initialization string to be sent to the Newland barcode engine.
- (BOOL) - btDiscoverSupportedDevicesInBackground:maxTime:filter:error:

  Performs background discovery of nearby supported bluetooth devices.
- (BOOL) - btDiscoverDevicesInBackground:maxTime:codTypes:error:

  Performs background discovery of the nearby bluetooth devices.
- (BOOL) - btDiscoverPrintersInBackground:maxTime:error:

  Performs background discovery of supported printers.
- (BOOL) - btDiscoverPrintersInBackground:

  Performs background discovery of supported printers.
- (BOOL) - btDiscoverPinpadsInBackground:maxTime:error:

  Performs background discovery of supported printers.
- (BOOL) - btDiscoverPinpadsInBackground:

  Performs background discovery of supported printers.
- (BOOL) - btConnect:pin:error:

  Tries to connect to remote device.
- (BOOL) - btDisconnect:error:

  Disconnects from remote device.
- (BOOL) - btConnectSupportedDevice:pin:error:

  Tries to connect to supported bluetooth device.
- (BOOL) - btWrite:length:error:

  Sends data to the connected remote device.
- (BOOL) - btWrite:error:

  Sends data to the connected remote device.
- (int) - btRead:length:timeout:error:

  Tries to read data from the connected remote device for specified timeout.
- (NSString ∗) - btReadLine:error:

  Tries to read string data, ending with CR/LF up to specifed timeout.
- (BOOL) - btEnableWriteCaching:error:

  Enables or disables write caching on the bluetooth stream.
- (NSArray ∗) - btDiscoverDevices:maxTime:codTypes:error:

  Performs synchronous discovery of the nearby bluetooth devices.
- (NSString ∗) - btGetDeviceName:error:

Queries device name given the address.

- (BOOL) - btSetDataNotificationMaxTime:maxLength:sequenceData:error:

  Sets the conditions to fire the NSStreamEventHasBytesAvailable event on bluetooth streams.
- (BOOL) - btListenForDevices:discoverable:localName:cod:error:

  Initiates/kills listen for incoming bluetooth connections.
- (BOOL) - extOpenSerialPort:baudRate:parity:dataBits:stopBits:flowControl:error:

  Opens the external serial port with specified settings.
- (BOOL) - extCloseSerialPort:error:

  Closes the external serial port opened with extOpenSerialPort.
- (BOOL) - extWriteSerialPort:data:error:

  Sends data to the connected remote device via serial port.
- (NSData ∗) - extReadSerialPort:length:timeout:error:

  Reads data from the connected remote device via serial port.
- (BOOL) - tcpConnectSupportedDevice:error:

  Tries to connect to supported device over the network.
- (BOOL) - tcpDisconnect:error:

  Disconnects from remote device.
- (NSData ∗) - cryptoRawGenerateRandomData:

  Generates 16 byte block of random numbers, required for some of the other crypto functions.
- (BOOL) - cryptoRawSetKey:encryptedData:keyVersion:keyFlags:error:
- (BOOL) - cryptoSetKey:key:oldKey:keyVersion:keyFlags:error:

  Used to store AES256 keys into Linea internal memory.
- (BOOL) - cryptoGetKeyVersion:keyVersion:error:

  Returns key version.
- (NSData ∗) - cryptoRawAuthenticateDevice:error:
- (BOOL) - cryptoAuthenticateDevice:error:
- (BOOL) - cryptoRawAuthenticateHost:error:
- (BOOL) - cryptoAuthenticateHost:error:
- (BOOL) - emsrSetActiveHead:error:

  In case there are multiple encrypted heads on the device, sets the active one.
- (NSDictionary ∗) - emsrGetFirmwareInformation:error:

  Returns information about the specified head firmware data.
- (BOOL) - emsrIsTampered:error:

  Checks if the head was tampered or not.
- (BOOL) - emsrGetKeyVersion:keyVersion:error:

  Retrieves the key version (if any) of a loaded key.
- (BOOL) - emsrLoadInitialKey:error:

  Loads Terminal Master Key (TMK) or reenable after tampering.
- (BOOL) - emsrLoadKey:error:

  Loads new key, in plain or encrypted with already loaded AES256 Key Encryption Key (KEK).
- (NSData ∗) - emsrGetDUKPTSerial:

  Returns DUKPT serial number, if DUKPT key is set.
- (NSString ∗) - emsrGetDeviceModel:

  Returns head's model.
- (BOOL) - emsrGetFirmwareVersion:error:

  Returns head's firmware version as number MAJOR∗100+MINOR, i.e.
- (BOOL) - emsrGetSecurityVersion:error:

  Returns head's security version as number MAJOR∗100+MINOR, i.e.
- (NSData ∗) - emsrGetSerialNumber:

  Return head's unique serial number as byte array.
- (BOOL) - emsrUpdateFirmware:error:

Performs firmware update on the encrypted head.

- (NSArray ∗) - emsrGetSupportedEncryptions:

  Returns supported encryption algorhtms by the encrypted head.
- (BOOL) - emsrSetEncryption:params:error:

  Selects the prefered encryption algorithm.
- (BOOL) - emsrSetEncryption:keyID:params:error:

  Selects the prefered encryption algorithm.
- (BOOL) - emsrConfigMaskedDataShowExpiration:unmaskedDigitsAtStart:unmaskedDigitsAtEnd:error:

  Fine-tunes which part of the card data will be masked, and which will be sent in clear text for display/print purposes.
- (BOOL) - emsrConfigMaskedDataShowExpiration:unmaskedDigitsAtStart:unmaskedDigitsAtEnd:unmasked-DigitsAfter:error:

  Fine-tunes which part of the card data will be masked, and which will be sent in clear text for display/print purposes.
- (BOOL) - **emsrLoadRSAKeyPEM:version:error:**
- (EMSRDeviceInfo ∗) - emsrGetDeviceInfo:

  Returns general information about the encrypted head - firmware version, ident, serial number.
- (EMSRKeysInfo ∗) - emsrGetKeysInfo:

  Returns information about the loaded keys in the encrypted head and tampered status.
- (DTVoltageInfo ∗) - voltageGetInfo:

  Returns various information about Voltage state.
- (BOOL) - voltageLoadConfiguration:error:

  Loads new configuration.
- (BOOL) - voltageGenerateNewKey:

  Forces generation of a new key.
- (BOOL) - rfInit:error:

  Initializes and powers on the RF card reader module.
- (BOOL) - rfClose:

  Powers down RF card reader module.
- (BOOL) - rfRemoveCard:error:

  Call this function once you are done with the card, a delegate call rfCardRemoved will be called when the card leaves the RF field and new card is ready to be detected.
- (BOOL) - mfAuthByKey:type:address:key:error:

  Authenticate mifare card block with direct key data.
- (BOOL) - mfStoreKeyIndex:type:key:error:

  Store key in the internal module memory for later use.
- (BOOL) - mfAuthByStoredKey:type:address:keyIndex:error:

  Authenticate mifare card block with previously stored key.
- (NSData ∗) - mfRead:address:length:error:

  Reads one more more blocks of data from Mifare Classic/Ultralight cards.
- (int) - mfWrite:address:data:error:

  Writes one more more blocks of data to Mifare Classic/Ultralight cards.
- (BOOL) - mfUlcSetKey:key:error:

  Sets the 3DES key of Mifare Ultralight C cards.
- (BOOL) - mfUlcAuthByKey:key:error:

  Performs 3DES authentication of Mifare Ultralight C card using the given key.
- (NSData ∗) - iso15693Read:startBlock:length:error:

  Reads one more more blocks of data from ISO 15693 card.
- (int) - iso15693Write:startBlock:data:error:

  Writes one more more blocks of data to ISO 15693 card.
- (NSData ∗) - iso15693GetBlocksSecurityStatus:startBlock:nBlocks:error:

  Reads the security status of one more more blocks from ISO 15693 card.
- (BOOL) - iso15693LockBlock:block:error:

Locks a single ISO 15693 card block.
- (BOOL) - iso15693WriteAFI:afi:error:

  Changes ISO 15693 card AFI.
- (BOOL) - iso15693LockAFI:error:

  Locks ISO 15693 AFI preventing further changes.
- (BOOL) - iso15693WriteDSFID:dsfid:error:

  Changes ISO 15693 card DSFID.
- (BOOL) - iso15693LockDSFID:error:

  Locks ISO 15693 card DSFID preventing further changes.
- (NSData ∗) - iso14GetATS:error:

  Initializes ISO1443B card and returns Answer To Select.
- (NSData ∗) - iso14APDU:cla:ins:p1:p2:data:apduResult:error:

  Executes APDU command on ISO1443B compatible card.
- (BOOL) - felicaSetPollingParamsRequestCode:systemCode:error:

  Sets polling parameters of FeliCa card.
- (NSData ∗) - **felicaSendCommand:command:data:error:**
- (NSData ∗) - felicaRead:serviceCode:startBlock:length:error:

  Reads one more more blocks of data from FeliCa card.
- (int) - felicaWrite:serviceCode:startBlock:data:error:

  Writes one more more blocks of data to FeliCa card.
- (BOOL) - felicaSmartTagGetBatteryStatus:status:error:

  Returns FeliCa SmartTag battery status.
- (BOOL) - felicaSmartTagClearScreen:error:

  Clears the screen of FeliCa SmartTag.
- (BOOL) - felicaSmartTagDrawImage:image:topLeftX:topLeftY:drawMode:layout:error:

  Draws image on FeliCa SmartTag's screen.
- (BOOL) - felicaSmartTagSaveLayout:layout:error:

  Saves the current display as layout number.
- (BOOL) - felicaSmartTagDisplayLayout:layout:error:

  Displays previously stored layout.
- (int) - felicaSmartTagWrite:address:data:error:

  Writes data in FeliCa SmartTag.
- (NSData ∗) - felicaSmartTagRead:address:length:error:

  Writes data in FeliCa SmartTag.
- (BOOL) - felicaSmartTagWaitCompletion:error:

  Waits for FeliCa SmartTag to complete current operation.
- (NSData ∗) - stSRIRead:address:length:error:

  Reads one more more blocks of data from ST SRI card.
- (int) - stSRIWrite:address:data:error:

  Writes one more more blocks of data to ST SRI card.
- (NSData ∗) - **hidGetVersionInfo:**
- (NSData ∗) - **hidGetSerialNumber:**
- (NSData ∗) - **hidGetContentElement:pin:rootSoOID:error:**
- (BOOL) - scInit:error:

  Initializes SmartCard module.
- (NSData ∗) - scCardPowerOn:error:

  Powers on the SmartCard, resets it and returns ATR (Answer To Reset).
- (BOOL) - scCardPowerOff:error:

  Powers off SmartCard, call this function when you are done with the card.
- (BOOL) - scIsCardPresent:error:

  Manually checks if there is a card in the reader.

- (NSData ∗) - scCAPDU:apdu:error:

  Performs APDU command in the card.
- (BOOL) - scClose:error:

  Shuts down SmartCard module.
- (BOOL) - ppadStartPINEntry:startY:timeout:echoChar:message:error:

  Initiates asynchronous PIN entry procedure.
- (BOOL) - ppadCancelPINEntry:

  Tries to cancel asynchronous PIN entry procedure.
- (BOOL) - ppadMagneticCardEntry:timeout:error:

  Initiates synchronous magnetic card entry procedure.
- (NSData ∗) - ppadGetPINBlockUsingFixedKey:keyVariant:pinFormat:error:

  Gets encrypted pin data using pre-loaded 3DES key The returned data consists of:
- (NSData ∗) - pinGetPINBlockUsingDUKPT:keyVariant:pinFormat:error:

  Gets encrypted pin data using DUKPT.
- (DTKeyInfo ∗) - ppadGetKeyInfo:error:

  Gets information about some of the keys loaded in the pinpad.
- (NSData ∗) - **ppadGetDUKPTKeyKSN:error:**
- (NSData ∗) - ppadCryptoExchangeKeyID:kekID:usage:version:data:cbc:error:

  Loads/changes 3DES key into the pinpad.
- (NSData ∗) - ppadCryptoTR31ExchangeKeyID:kekID:tr31:error:

  Loads/changes 3DES key into the pinpad.
- (NSData ∗) - ppadCrypto3DESECBEncryptKeyID:inData:error:

  Encrypts a data on the pinpad using 3DES ECB.
- (NSData ∗) - ppadCrypto3DESECBDecryptKeyID:inData:error:

  Decrypts a data on the pinpad using 3DES ECB.
- (NSData ∗) - ppadCrypto3DESCBCEncryptKeyID:initVector:inData:error:

  Encrypts a data on the pinpad using 3DES CBC.
- (NSData ∗) - ppadCrypto3DESCBCDecryptKeyID:initVector:inData:error:

  Decrypts a data on the pinpad using 3DES CBC.
- (BOOL) - ppadSetButtonCaption:caption:error:

  Sets the text that is drawn above functional buttons in MPED400.
- (DTPinpadInfo ∗) - ppadGetSystemInfo:

  Returns pinpad specific information.
- (BOOL) - ppadKeyboardControl:error:

  Captures or releases keyboard.
- (BOOL) - ppadReadKey:error:

  Reads key from the pinpad.
- (BOOL) - caImportKeyNumber:RIDI:module:exponent:error:

  Import CA key.
- (BOOL) - caWriteKeysToFlash:

  Writes CA keys to flash.
- (NSArray ∗) - caGetKeysData:

  Returns keys data.
- (NSData ∗) - caImportIssuerKeyNumber:exponent:remainder:certificate:error:

  Import issuer key.
- (NSData ∗) - caImportICCKeyType:exponent:remainder:certificate:error:

  Import ICC key.
- (NSData ∗) - caRSAVerify:inData:error:

  RSA verify.
- (BOOL) - emv2Initialise:

  This command initializes the emv kernel, call it before calling any other EMV function.

- (BOOL) - emv2Deinitialise:

  This command deinitializes the emv kernel and frees the allocated resources, call it after you are done with the EMV transaction.
- (BOOL) - emv2SetCardEmulationMode:encryption:keyID:error:

  Activates magnetic card emulation mode for the EMV.
- (BOOL) - emv2LoadConfigurationData:error:

  Loads EMV kernel configuration data.
- (DTEMV2Info ∗) - emv2GetInfo:

  Returns information about loaded configuration.
- (BOOL) - emv2SetTransactionType:amount:currencyCode:error:

  Sets EMV transaction parameters, this function must be called before starting EMV transaction.
- (BOOL) - emv2StartTransactionWithFlags:initData:error:

  Starts EMV transaction.
- (BOOL) - emv2SelectApplication:error:

  Selects application to be used by EMV kernel.
- (BOOL) - emv2SetOnlineResult:error:

  Responds to the EMV kernel after an online request was sent to the card and the communication with the financial institution is complete.
- (BOOL) - emv2CancelTransaction:

  Cancels an active EMV operation and clears all data.
- (NSData ∗) - emv2GetCardTracksEncryptedWithFormat:keyID:error:

  After transaction is finished, you can get the card data in magnetic-stripe format by using one of the available Encrypted Head formats.
- (NSData ∗) - emv2GetTagsEncrypted:format:keyID:error:

  After transaction is finished, you can get the tags, encrypted and in a specified format.
- (NSData ∗) - emv2GetTagsPlain:error:

  After transaction is finished, you can get the tags in plain.
- (BOOL) - emvInitialise:

  This command initializes the emv kernel, call it before calling any other EMV function.
- (BOOL) - emvDeinitialise:

  This command deinitializes the emv kernel and frees the allocated resources, call it after you are done with the EMV transaction.
- (BOOL) - emvATRValidation:warmReset:error:

  The command is in charge of validating the ATR sequence got from the card to ensure that is fully EMV compliant and that obeys the rules stated in the specification.
- (BOOL) - emvLoadAppList:selectionMethod:includeBlockedAIDs:error:

  The command initiates the application selection process, loading the application list supported by the terminal.
- (NSArray ∗) - emvGetCommonAppList:error:

  The command gets back the list of common applications supported by the terminal and the card, actually this commands will end or resume the selection procedure.
- (BOOL) - emvInitialAppProcessing:error:

  Once an application has been selected, the next phase is to start the transaction with it by issuing the GET PROCESSING ommand and analyzing the information got.
- (BOOL) - emvReadAppData:error:

  The command reads and validates the data informed in the AFL and that will be used along the transaction.
- (BOOL) - emvAuthentication:error:

  Through this command the card data is authenticated depending on the capabilities of the card and the kernel.
- (BOOL) - emvProcessRestrictions:

  The command performs the restrictions processing related to application version, application usage control and effective and expiry dates.
- (BOOL) - emvTerminalRisk:error:

  The application risk control is done by this command, including Floorlimit checking, Random selection (only if offline is enabled) and Velocity checking.

- (BOOL) - emvGetAuthenticationMethod:

  The command starts or resumes the cardholder authentication procedure, the current verification method is communicated to the application.
- (BOOL) - emvSetAuthenticationResult:error:

  Using this command the kernel gets the result of the previously informed verification method.
- (BOOL) - emvVerifyPinOffline:

  The command allows the application to apply the offline PIN verification (plaintext or encrypted) method.
- (BOOL) - emvGenerateCertificate:risk:error:

  Using this command the application will be able to generate an application cryptogram, the first or the second one, as required by the transaction.
- (BOOL) - emvMakeTransactionDecision:

  The command checks the action codes (provided by the application and read from the card), the TVR and will determine how the transaction is resolved.
- (BOOL) - emvMakeDefaultDecision:

  The command checks the default action code (provided by the application and read from the card), the TVR and will determine how the transaction is resolved by default.
- (BOOL) - emvAuthenticateIssuer:

  The command is used to validate the cryptogram got from the issuer.
- (BOOL) - emvScriptProcessing:error:

  The script processing retrieved in the online authorization is handled by this command.
- (BOOL) - emvUpdateTVRByte:bit:value:error:

  The command allows modifying the TVR directly, setting or unsetting the desired bits.
- (BOOL) - emvUpdateTSIByte:bit:value:error:

  The command allows modifying the TSI directly, setting or unsetting the desired bits.
- (BOOL) - emvCheckTVRByte:bit:error:

  The command is intended to verify an individual bit within the TVR.
- (BOOL) - emvCheckTSIByte:bit:error:

  The command is intended to verify an individual bit within the TSI.
- (BOOL) - emvRemovePublicKey:RID:error:

  The command is intended to delete a given CA public key.
- (BOOL) - emvSetDataAsBinary:data:error:

  The command sets a data item with data in binary format (raw data).
- (BOOL) - emvSetDataAsString:data:error:

  The command sets a data item with data in string format.
- (NSData ∗) - emvGetDataAsBinary:error:

  The command gets a data item in binary format (raw data).
- (NSString ∗) - emvGetDataAsString:error:

  The command gets a data item in string format.
- (BOOL) - emvGetDataDetails:tagType:maxLen:currentLen:error:

  The command allows the application direct access to the data of a given item.
- (BOOL) - emvSetBypassMode:error:

  With this command is possible to setup the behavior of the KERNEL regarding the PIN based method bypass, so that only the current method will be bypassed or any other found later in the CVM list will be considered so as well.
- (BOOL) - emvSetTags:error:

  Loads multiple tags at the same time, this is much faster than calling them 1 by 1.
- (NSData ∗) - emvGetTags:error:

  Reads multiple tags at the same time, this is much faster than calling them 1 by 1.
- (NSData ∗) - emvGetTagsEncrypted3DES:keyID:uniqueID:error:

  Reads multiple tags at the same time and sends them encrypted, this is much faster than calling them 1 by 1.
- (NSData ∗) - emvGetTagsEncryptedDUKPT:keyID:uniqueID:error:

  Reads multiple tags at the same time and sends them encrypted, this is much faster than calling them 1 by 1.
- (BOOL) - uiGetScreenInfoWidth:height:colorMode:error:

Returns screen properties.

- (BOOL) - uiDrawText:topLeftX:topLeftY:font:error:

  Disaplay some text, starting at a specified position.
- (BOOL) - uiFillRectangle:topLeftY:width:height:color:error:

  Fills rectangle on the screen with specified color.
- (BOOL) - uiSetContrast:error:

  Set display contrast.
- (BOOL) - uiPutPixel:y:color:error:

  Draws pixel on the screen with specified color.
- (BOOL) - uiDisplayImage:topLeftY:image:error:

  Displays image on the screen.
- (BOOL) - uiStartAnimation:topLeftX:topLeftY:animated:error:

  Draws predefined animation on the screen.
- (BOOL) - uiStopAnimation:error:

  Stops animation playback started with ppUiStartAnimation.
- (BOOL) - uiControlLEDsWithBitMask:error:

  Enables or disables controllable LEDs on the device based on bit mask.
- (BOOL) - uiEnableVibrationForTime:error:

  Activates vibration motor (if available) for a specific time.
- (BOOL) - uiEnableSpeaker:error:

  Enables or disables external speaker.
- (BOOL) - uiIsSpeakerEnabled:error:

  Returns the state of external speaker.
- (BOOL) - prnFlushCache:

  Forces data still in the sdk buffers to be sent directly to the printer.
- (BOOL) - prnWriteDataToChannel:data:error:

  Sends data to the connected printer no matter the connection type.
- (NSData ∗) - prnReadDataFromChannel:length:timeout:error:

  Tries to read data from the connected remote device for specified timeout.
- (BOOL) - prnWaitPrintJob:error:

  Waits specified timeout for the printout to complete.
- (BOOL) - prnGetPrinterStatus:error:

  Retrieves current printer status.
- (BOOL) - prnSelfTest:error:

  Prints selftest.
- (BOOL) - prnTurnOff:

  Forces printer to turn off.
- (BOOL) - prnFeedPaper:error:

  Feeds the paper X lines (1/203 of the inch) or as needed (different length based on the printer model) so it allows paper to be teared.
- (BOOL) - prnPrintBarcode:barcode:error:

  Prints barcode.
- (BOOL) - prnPrintLogo:error:

  Prints the stored logo.
- (BOOL) - prnSetBarcodeSettings:height:hriPosition:align:error:

  Set various barcode parameters.
- (BOOL) - prnSetDensity:error:

  Sets printer density level.
- (BOOL) - prnSetLineSpace:error:

  Sets the line "height" in pixels If the characters are 16 pixelx high for example, setting the linespace to 20 will make the printer leave 4 blank lines before next line of text starts.

- (BOOL) - prnSetLeftMargin:error:

  Sets left margin.
- (BOOL) - prnPrintText:usingEncoding:error:

  Prints text with specified font/styles.
- (BOOL) - prnPrintText:error:

  Prints text with specified font/styles.
- (BOOL) - prnPrintDelimiter:error:

  Prints the delimiter character at the whole width of the paper, adjusting itself to the paper width.
- (BOOL) - prnGetBlackMarkTreshold:error:

  Returns blackmark sensor treshold or UnsupportedOperationException if printer is not in blackmark mode.
- (BOOL) - prnSetBlackMarkTreshold:error:

  Sets blackmark sensor treshold or UnsupportedOperationException if printer is not in blackmark mode.
- (BOOL) - prnCalibrateBlackMark:error:

  Provides blackmark sensor calibration by scaning 200mm of paper for possible black marks and adjust the sensor treshold.
- (BOOL) - prnLoadLogo:align:error:

  Loads logo into printer's memory.
- (BOOL) - prnPrintImage:align:error:

  Prints Bitmap object using specified alignment.
- (BOOL) - pageIsSupported

  Returns TRUE if page mode is supported on the connected device.
- (BOOL) - pageStart:

  Creates a new virtual page using the maximum supported page height.
- (BOOL) - pagePrint:

  Prints the content of the virtual page.
- (BOOL) - pageEnd:

  Exits page mode.
- (BOOL) - pageSetWorkingArea:top:width:height:error:

  Sets a working area and orientation inside the virtual page.
- (BOOL) - pageSetWorkingArea:top:width:heigth:orientation:error:

  Sets a working area and orientation inside the virtual page.
- (BOOL) - pageFillRectangle:error:

  Fills the current working area (or whole page if none is set) with the specified color.
- (BOOL) - pageFillRectangle:top:width:height:color:error:

  Fills a rectangle inside the current working area with specified color.
- (BOOL) - pageRectangleFrame:top:width:height:framewidth:color:error:

  Draws a rectangle frame inside the current working area with specified color.
- (BOOL) - tableIsSupported

  Checks if the currently connected printer supports tables.
- (BOOL) - tableCreate:error:

  Create a new table using custom flags.
- (BOOL) - tableCreate:

  Create a new table using default settings - both horizontal and vertical borders around it.
- (BOOL) - tableAddColumn:

  Adds a new column using default settings - 12x24 font, plain, vertical border between the cells, left aligning.
- (BOOL) - tableAddColumn:error:

  Adds a new column using default settings - plain text, vertical border between the cells, left aligning.
- (BOOL) - tableAddColumn:style:alignment:error:

  Adds a new column using custom font and vertical border between the cells.
- (BOOL) - tableAddColumn:style:alignment:flags:error:

  Adds a new column.

- (BOOL) - tableAddCell:error:

  Adds a new cell using the font size and style and aligning of the column that cell belongs to.
- (BOOL) - tableAddCell:font:error:

  Adds a new cell using the font style and aligning of the column that cell belongs to.
- (BOOL) - tableAddCell:font:style:error:

  Adds a new cell using custom font size and style and aligning of the column that cell belongs to.
- (BOOL) - tableAddCell:font:style:alignment:error:

  Adds a new cell using custom font size and style and aligning.
- (BOOL) - tableAddDelimiter:

  Adds aa horizontal black line to the entire row that separates it from the next.
- (BOOL) - tableSetRowHeight:error:

  Sets the row height that will be used by default for new cells added.
- (BOOL) - tablePrint:

  Prints current table or throws IllegalArgumentException if cell data cannot be fit into paper.

Class Methods

- (id) + sharedDevice

  Creates and initializes new class instance or returns already initalized one.

Properties

- NSInputStream ∗ btInputStream

  Bluetooth input stream, you can use it after connecting with btConnect.
- NSOutputStream ∗ btOutputStream

  Bluetooth output stream, you can use it after connecting with btConnect.
- NSArray ∗ btConnectedDevices

  Contains bluetooth addresses of the currently connected bluetooth devices or empty array if no connected devices are found.
- NSArray ∗ tcpConnectedDevices

  Contains tcp addresses of the currently connected network devices or empty array if no connected devices are found.
- int uiDisplayWidth

  Contains display width in pixels.
- int uiDisplayHeight

  Contains display height in pixels.
- BOOL uiDisplayAtBottom

  Contains display height in pixels.
- id delegate

  Adds delegate to the class.
- NSMutableArray ∗ delegates

  Provides a list of currently registered delegates.
- int connstate

  Returns current connection state.
- NSString ∗ deviceName

  Returns connected device name.
- NSString ∗ deviceModel

  Returns connected device model.
- NSString ∗ firmwareRevision

  Returns connected device firmware version.
- NSString ∗ hardwareRevision

Returns connected device hardware version.

- NSString ∗ serialNumber

  Returns connected device serial number.

- int sdkVersion

  SDK version number in format MAJOR∗100+MINOR, i.e.

- NSDate ∗ sdkBuildDate

  SDK build date.

- short emvLastStatus

  EMV last status, one of the EMV_∗ constants.

3.3.1 Detailed Description

Provides universal access to all supported devices' functions.

In order to use one of the supported accessories in your program, several steps have to be performed:

- Include DTDevices.h and libdtdev.a in your project.
- Go to Frameworks and add ExternalAccessory framework
- Edit your program plist file, add new element and select "Supported external accessory protocols" from the list, then add the protocol names of the accessories you want to connect to:

  For Linea series: com.datecs.linea.pro.msr and com.datecs.linea.pro.bar

  For Pinpad: com.datecs.iserial.communication and com.datecs.ppad

  For iSerial: com.datecs.iserial.communication

  For ESC/POS printers: com.datecs.printer.escpos

Since this SDK is based on features, the specific device is not that important, for example, if your program relies on barcode scanning, then Linea, Pinpad or the ESC/POS printers can provide that functionality, so you can include all their protocols.

3.3.2 Method Documentation

3.3.2.1 - (BOOL) emv2CancelTransaction: (NSError ∗∗) *error*

Cancels an active EMV operation and clears all data.

Using this function is not required if the emv transaction completed.

Parameters

error	pointer to NSError object, where error information is stored in case function fails. You can pass nil if you don't want that information

Returns

TRUE if function succeeded, FALSE otherwise

3.3.2.2 - (BOOL) emv2Deinitialise: (NSError ∗∗) *error*

This command deinitializes the emv kernel and frees the allocated resources, call it after you are done with the EMV transaction.

Parameters

error	returns error information, you can pass nil if you don't want it

Returns

TRUE upon success, FALSE otherwise

3.3.2.3 - (NSData ∗) emv2GetCardTracksEncryptedWithFormat: (int) *format* keyID:(int) *keyID* error:(NSError ∗∗) *error*

After transaction is finished, you can get the card data in magnetic-stripe format by using one of the available Encrypted Head formats.

Parameters

format	encryption algorhtm used, one of the ALG_∗ constants
keyID	key ID, one of KEY_∗ constants. Passing 0 will use the default key for the specified algorith
error	pointer to NSError object, where error information is stored in case function fails. You can pass nil if you don't want that information

Returns

TRUE if function succeeded, FALSE otherwise

3.3.2.4 - (DTEMV2Info ∗) emv2GetInfo: (NSError ∗∗) *error*

Returns information about loaded configuration.

Parameters

error	pointer to NSError object, where error information is stored in case function fails. You can pass nil if you don't want that information

Returns

configuration information or nil if function failed

3.3.2.5 - (NSData ∗) emv2GetTagsEncrypted: (NSData ∗) *tagList* format:(int) *format* keyID:(int) *keyID* error:(NSError ∗∗) *error*

After transaction is finished, you can get the tags, encrypted and in a specified format.

Parameters

tagList	a list of tags to get. The format is like TLV list without length and value, i.e. every tag takes as many bytes as needed
format	one of the TAGS_FORMAT_∗ constants
keyID	key ID, one of KEY_∗ constants
error	pointer to NSError object, where error information is stored in case function fails. You can pass nil if you don't want that information

Returns

TRUE if function succeeded, FALSE otherwise

3.3.2.6 - (NSData ∗) emv2GetTagsPlain: (NSData ∗) *tagList* error:(NSError ∗∗) *error*

After transaction is finished, you can get the tags in plain.

Only non-sensitive tags can be retrieved in plain, no pan/discretionary data will be returned

Parameters

tagList	a list of tags to get. The format is like TLV list without length and value, i.e. every tag takes as many bytes as needed
error	pointer to NSError object, where error information is stored in case function fails. You can pass nil if you don't want that information

Returns

TRUE if function succeeded, FALSE otherwise

3.3.2.7 - (BOOL) emv2Initialise: (NSError ∗∗) *error*

This command initializes the emv kernel, call it before calling any other EMV function.

Parameters

error	returns error information, you can pass nil if you don't want it

Returns

TRUE upon success, FALSE otherwise

3.3.2.8 - (BOOL) emv2LoadConfigurationData: (NSData ∗) *data* error:(NSError ∗∗) *error*

Loads EMV kernel configuration data.

Configuration consists of custom tags setting various terminal capabilities and specific application parameters

Parameters

data	TLV list of configuration tags. If the configuration is loaded for the first time, or there is no configuration encryption key tag set, then it is possible to load it in plain
error	pointer to NSError object, where error information is stored in case function fails. You can pass nil if you don't want that information

Returns

TRUE if function succeeded, FALSE otherwise

3.3.2.9 - (BOOL) emv2SelectApplication: (int) *application* error:(NSError ∗∗) *error*

Selects application to be used by EMV kernel.

Call this function only after being notified by available applications.

Parameters

application	selected application index
error	pointer to NSError object, where error information is stored in case function fails. You can pass nil if you don't want that information

Returns

TRUE if function succeeded, FALSE otherwise

3.3.2.10 - (BOOL) emv2SetCardEmulationMode: (BOOL) *enabled* encryption:(int) *encryption* keyID:(int) *keyID* error:(NSError ∗∗) *error*

Activates magnetic card emulation mode for the EMV.

In this mode when a card is read, it will be encrypted by it and sent via magneticCardEncryptedData delegate. You still need to start the emv transaction, but providing emv2OnOnlineProcessing function or emv2OnTransaction-Finished is not needed. The emv2Deinitialise will be automatically called once the track data is dispatched.

Parameters

encryption	encryption algorhtm used, one of the ALG_∗ constants
keyID	key identifier, one of the KEY_∗ constants
error	pointer to NSError object, where error information is stored in case function fails. You can pass nil if you don't want that information

Returns

TRUE if function succeeded, FALSE otherwise

3.3.2.11 - (BOOL) emv2SetOnlineResult: (NSData ∗) *result* error:(NSError ∗∗) *error*

Responds to the EMV kernel after an online request was sent to the card and the communication with the financial institution is complete.

Parameters

result	TLV structure with response tags
error	pointer to NSError object, where error information is stored in case function fails. You can pass nil if you don't want that information

Returns

TRUE if function succeeded, FALSE otherwise

3.3.2.12 - (BOOL) emv2SetTransactionType: (int) *type* amount:(int) *amount* currencyCode:(int) *currencyCode* error:(NSError ∗∗) *error*

Sets EMV transaction parameters, this function must be called before starting EMV transaction.

Parameters

transaction	type, tag 9C, the value depends on the payment institution, use 00 if you are unsure
amount	transaction amount as integer, i.e. for USD, $12.50 will be sent as 1250
currencyCode	currency code, according to ISO 4217
error	pointer to NSError object, where error information is stored in case function fails. You can pass nil if you don't want that information

Returns

TRUE if function succeeded, FALSE otherwise

3.3.2.13 - (BOOL) emv2StartTransactionWithFlags: (int) *flags* initData:(NSData ∗) *initData* error:(NSError ∗∗) *error*

Starts EMV transaction.

The function waits for payment card to be available, then processes it and notifies of completion. You can cancel the transaction at any time.

Parameters

flags	controls the EMV transaction, use 0 for now
initData	optional TLV structure with additional parameters to be sent to the kernel
error	pointer to NSError object, where error information is stored in case function fails. You can pass nil if you don't want that information

Returns

TRUE if function succeeded, FALSE otherwise

3.3.3 Property Documentation

3.3.3.1 - (int) sdkVersion `[read]`, `[atomic]`, `[assign]`

SDK version number in format MAJOR∗100+MINOR, i.e.

version 1.15 will be returned as 115

3.4 DTEMV2Info Class Reference

Information about EMV L2 configuration.

Inherits NSObject.

Properties

- int configurationVersion

  Version number of the configuration.
- int emvKernelVersion

  Version number of the EMV L2 engine.

3.4.1 Detailed Description

Information about EMV L2 configuration.

3.5 DTEMVApplication Class Reference

Provides information about EMV application.

Inherits NSObject.

Properties

- NSData ∗ aid

  Application AID.

- NSString ∗ label

  Application label.
- int matchCriteria

  How the application is matched to the ones in the card:

3.5.1 Detailed Description

Provides information about EMV application.

3.5.2 Property Documentation

3.5.2.1 - (int) matchCriteria `[read], [write], [atomic], [assign]`

How the application is matched to the ones in the card:

MATCH_FULL	Complete match
MATCH_PARTIAL_VISA	Partial Visa match
MATCH_PARTIAL_EUROPAY	Partial Europay match

3.6 DTKeyInfo Class Reference

Pinpad key information.

Inherits NSObject.

Properties

- NSData ∗ checkValue

  Key check value.
- int type

  Key type.
- NSString ∗ usage

  Key usage, according to TR31: Usage/Mode:
- char mode

  Key mode, according to TR31.
- int version

  Key version.

3.6.1 Detailed Description

Pinpad key information.

3.6.2 Property Documentation

3.6.2.1 - (NSString∗) usage `[read], [write], [atomic], [copy]`

Key usage, according to TR31: Usage/Mode:

'B0' 'N' Base Derivation Key

'P0' 'E' pin key 'M1' 'C' key for ISO 9797-1 MAC Algorithm 1 'M3' 'C' key for ISO 9797-1 MAC Algorithm 3 'M0' 'C' key for ISO 16609 MAC algorithm 1 'D0' 'E' key for data encrypting 'D0' 'D' key for data decrypting

Custom method usage vaules:

’01’ transport key for pin key ’02’ transport key for ISO 9797-1 MAC Algorithm 1 key ’03’ transport key for ISO 9797-1 MAC Algorithm 3 key ’04’ transport key for ISO 16609 MAC algorithm 1 key ’05’ transport key for data encrypting key ’06’ transport key for data decrypting key

3.7 DTPinpadInfo Class Reference

Information about connected Pinpad.

Inherits NSObject.

Properties

- NSData ∗ cpuSerial

  Unique CPU serial number.
- uint32_t cpuVersion

  CPU version.
- uint32_t cpuLoaderVersion

  CPU loader version.
- uint32_t cpuHALVersion

  HAL version.
- NSData ∗ pinpadSerial

  PinPad serial number.
- NSString ∗ loaderName

  Loader name.
- uint32_t loaderVersion

  Loader version.
- NSString ∗ fwName

  Firmware name.
- uint32_t fwVersion

  Firmware version.

3.7.1 Detailed Description

Information about connected Pinpad.

3.8 DTRFCardInfo Class Reference

Information about RF card.

Inherits NSObject.

Properties

- int type

  RF card type, one of the CARD_∗ constants.
- NSString ∗ typeStr

  RF card type as string, useful for display purposes.
- NSData ∗ UID

RF card unique identifier, if any.

- int ATQA

  Mifare card ATQA.
- int SAK

  Mifare card SAK.
- int AFI

  ISO15693 card AFI.
- int DSFID

  ISO15693 card DSFID.
- int blockSize

  ISO15693 card block size.
- int nBlocks

  ISO15693 card number of blocks.

3.8.1 Detailed Description

Information about RF card.

3.9 DTVoltageInfo Class Reference

Information about Voltage.

Inherits NSObject.

Properties

- BOOL keyGenerated

  Key is available, card can be read and encrypted.
- BOOL keyGenerationInProgress

  Key generation in progress, wile the key is generated the old key will be used for encryption.
- NSDate $*$ keyGenerationDate

  The date/time of the last key generated.
- int settingsVersion

  Version of the voltage settings.

3.9.1 Detailed Description

Information about Voltage.

3.10 EMSRDeviceInfo Class Reference

The class that represents Encrypted Magnetic Head information.

Inherits NSObject.

Properties

- NSString ∗ ident

  Identification string, for example "EMSR R".
- NSData ∗ serialNumber

  Unique serial number (16 bytes)
- NSString ∗ serialNumberString

  Unique serial number (16 bytes) in hexadeciamal string for display purposes.
- int firmwareVersion

  Firmware version number in format MAJOR∗1000 + MINOR, i.e.
- NSString ∗ firmwareVersionString

  Firmware version number in string format, for display purposes.
- int securityVersion

  Security firmware version number in format MAJOR∗1000 + MINOR, i.e.
- NSString ∗ securityVersionString

  Firmware version number in string format, for display purposes.

3.10.1 Detailed Description

The class that represents Encrypted Magnetic Head information.

3.10.2 Property Documentation

3.10.2.1 - (int) firmwareVersion `[read], [write], [atomic], [assign]`

Firmware version number in format MAJOR∗1000 + MINOR, i.e.

version 1.123 will be presented as 1123

3.10.2.2 - (int) securityVersion `[read], [write], [atomic], [assign]`

Security firmware version number in format MAJOR∗1000 + MINOR, i.e.

version 1.123 will be presented as 1123

3.11 EMSRKey Class Reference

The class that represents Encrypted Magnetic Head key.

Inherits NSObject.

Properties

- int keyID

  The ID of the key, one of the KEY_∗ constants.
- int keyVersion

  The version of the key.
- NSString ∗ keyName

  The name of the key (for display purposes)

3.11.1 Detailed Description

The class that represents Encrypted Magnetic Head key.

3.12 EMSRKeysInfo Class Reference

The class that represents Encrypted Magnetic Head keys information.

Inherits NSObject.

Instance Methods

- (int) - getKeyVersion:

  Returns key version.

Class Methods

- (NSString ∗) + keyNameByID:

  Returns the name of a key (for display purposes)

Properties

- NSArray ∗ keys

  An array of EMSRKey objects representing the keys in the head.
- bool tampered

  Indicates if the head is tampered or not.

3.12.1 Detailed Description

The class that represents Encrypted Magnetic Head keys information.

3.12.2 Method Documentation

3.12.2.1 - (int) getKeyVersion: (int) *keyID*

Returns key version.

Parameters

keyID	key ID, one of the KEY_∗ constants

Returns

key version or 0 if the key is missing

3.12.2.2 + (NSString ∗) keyNameByID: (int) *keyID*

Returns the name of a key (for display purposes)

Parameters

keyID	key ID, one of the KEY_∗ constants

Returns

name string or nil if the ID was wrong

3.12.3 Property Documentation

3.12.3.1 - (bool) tampered [read], [write], [atomic], [assign]

Indicates if the head is tampered or not.

Tampered head needs to be reactivated at secure facility after checking

Chapter 4

Example Documentation

4.1 To

Executes a raw command on FeliCa card. The IDm(UUID) parameter is automatically added and needs to be skipped. read block 0 with service code 0x0900, then you need to send command 06 with data 01 09 00 01 80 00

Parameters

cardIndex	the index of the card as sent by rfCardDetected delegate call
command	command code, refer to FeliCa documentation
data	optional data to the command
error	pointer to NSError object, where error information is stored in case function fails. You can pass nil if you don't want that information

Returns

NSData object containing the data received or nil if an error occured. Received data contains command code (1 byte) and command data, IDm(UUID) is skipped

Index