libsdi Namespace Reference

Data Structures

class	CardDetection
	Interface for SDI Card Detection Interface, command class 23. More...
class	Dialog
class	ManualEntry
	Interface for SDI command MSR Card Data Entry (21-02) More...
struct	MatchingRecord
class	PED
class	SDI
class	SdiBase
class	SdiCmd
	Composition for TLV based SDI commands. More...
class	SdiCrypt

Enumerations

SDI_SW12 {

SDI_SW12_NONE = 0,
SDI_SW12_SUCCESS = 0x9000,
SDI_SW12_TAG_ERROR = 0x6200,
SDI_SW12_TAG_LENGTH_ERROR = 0x6300,

SDI_SW12_EXEC_ERROR = 0x6400,
SDI_SW12_CANCELED_BY_USER = 0x6405,
SDI_SW12_BUSY = 0x640A,
SDI_SW12_TIMEOUT_PIN_ENTRY = 0x640C,

SDI_SW12_TIMEOUT_CARD_REMOVAL = 0x64F7,
SDI_SW12_INTERCHAR_PIN_ENTRY = 0x64F8,
SDI_SW12_COMMAND_NOT_ALLOWED = 0x64F9,
SDI_SW12_MAIN_CONNECTION_USED = 0x64FA,

SDI_SW12_INVALID_FILE_CONTENT = 0x64FB,
SDI_SW12_FILE_ACCESS_ERROR = 0x64FC,
SDI_SW12_LOGIC_ERROR = 0x64FD,
SDI_SW12_SDI_PARAMETER_ERROR = 0x64FE,

SDI_SW12_LUHN_CHECK_FAILED = 0x64FF,
SDI_SW12_EXECUTION_ABORTED = 0x6500,
SDI_SW12_EXECUTION_TIMEOUT = 0x6600,
SDI_SW12_MESSAGE_LENGTH_ERROR = 0x6700,

SDI_SW12_NO_SDI_PLUGIN_AVAILABLE = 0x6800,
SDI_SW12_UNKNOWN_PLUGIN_ID = 0x6801,
SDI_SW12_UNKNOWN_PLUGING_ID = 0x6801,
SDI_SW12_INVALID_PLUGIN_RESPONSE = 0x6802,

SDI_SW12_EPP_CONNECTION_ERROR = 0x6900,
SDI_SW12_UNKNOWN_INS_BYTE = 0x6D00,
SDI_SW12_UNKNOWN_CLA_BYTE = 0x6E00,
SDI_SW12_CMAC_ERROR = 0x6FB0,
enum

SDI_SW12_CMAC_LENGTH_ERROR = 0x6FB1,
SDI_SW12_CMAC_MISSING_ERROR = 0x6FB2,
SDI_SW12_ENCRYPTION_ERROR = 0x6FB4,
SDI_SW12_ENCRYPTION_LENGTH_ERROR = 0x6FB5,

SDI_SW12_ENCRYPTION_MISSING_ERROR = 0x6FB6,
SDI_SW12_DECRYPTION_ERROR = 0x6FB8,
SDI_SW12_DECRYPTION_LENGTH_ERROR = 0x6FB9,

SDICLIENT_ERROR {

SDICLIENT_ERROR_NONE = 0,
SDICLIENT_ERROR_COMMUNICATION = -1,
SDICLIENT_ERROR_CONCURRENT_USE = -2,
SDICLIENT_ERROR_CONNECT = -3,

SDICLIENT_ERROR_OVERFLOW = -4,
enum SDICLIENT_ERROR_PARAM = -5,
SDICLIENT_ERROR_OTHER = -6,
SDICLIENT_ERROR_NO_RECEIVE = -7,

SDICLIENT_ERROR_NOT_SUPPORTED = -10,
SDICLIENT_ERROR_NOT_ALLOWED = -11

}

SYSUploadType {

SYS_UPLOAD_SOFTWARE_UPDATE,
SYS_UPLOAD_CONFIG_WHITELIST,
SYS_UPLOAD_CONFIG_SENSITIVE_TAGS,
SYS_UPLOAD_CONFIG_CARD_RANGES,
enum

SYS_UPLOAD_INSTALL_CP_PACKAGE = 11,
SYS_UPLOAD_EMV_CONFIGURATION

}

Functions

enum SDICLIENT_ERROR	getNfcClientError ()
enum SDI_SW12	getNfcSW12 ()
CL_STATUS	NFC_Client_Init (CONNECTION_TYPE type)
CL_STATUS	NFC_Client_Init_CheckVer (CONNECTION_TYPE type, int maj, int min, int bld)
CL_STATUS	NFC_SerialOpen (void)
CL_STATUS	NFC_SerialClose (void)
ResponseCodes	NFC_Ping (rawData *output)
ResponseCodes	NFC_Get_Version (rawData *output)
ResponseCodes	NFC_Config_Init (void)

ResponseCodes	NFC_Set_Callback_Function (rawData *id, NfcCallbackFunction *callbackFunction)
ResponseCodes	NFC_Callback_Test (void)
ResponseCodes	NFC_PT_Open ()
ResponseCodes	NFC_PT_Close ()
ResponseCodes	NFC_PT_FieldOn ()
ResponseCodes	NFC_PT_FieldOff ()
ResponseCodes	NFC_PT_Polling (pollReq *inPollReq, pollRes *outPollRes)
ResponseCodes	NFC_PT_PollingFull (pollReq *inPollReq, pollResFull *outPollRes)
void	NFC_Free_Poll_Data (pollRes *outPollRes)
void	NFC_Free_Poll_Data_Full (pollResFull *outPollRes)
ResponseCodes	NFC_PT_Cancel_Polling (void)
ResponseCodes	NFC_PT_Activation (NFC_CARD_TYPE cardtype, rawData *rd_activationData)
ResponseCodes	NFC_PT_FtechBaud (NFC_F_BAUD baud)
ResponseCodes	NFC_PT_TxRx (NFC_CARD_TYPE cardtype, rawData *inBuff, rawData *outBuff)
ResponseCodes	NFC_Mifare_Authenticate (unsigned char blockNumber, MIFARE_KEY_TYPE keyType, rawData *Key)
ResponseCodes	NFC_Mifare_Read (I_MIFARE_CARD_TYPE m_cardType, unsigned int StartBlockNum, unsigned int blockAmount, rawData *out_buff)
ResponseCodes	NFC_Mifare_Write (I_MIFARE_CARD_TYPE m_cardType, unsigned int StartBlockNum, unsigned int blockAmount, rawData *in_buff)
ResponseCodes	NFC_Mifare_Increment (unsigned int blockNum, int amount)
ResponseCodes	NFC_Mifare_Decrement (unsigned int blockNum, int amount)
ResponseCodes	NFC_Mifare_Increment_Only (unsigned int blockNum, int amount)
ResponseCodes	NFC_Mifare_Decrement_Only (unsigned int blockNum, int amount)
ResponseCodes	NFC_Mifare_Transfer (unsigned int blockNum)
ResponseCodes	NFC_Mifare_Restore (unsigned int blockNum)
ResponseCodes	NFC_Felica_Exchange (felicaTxData *in_buff, felicaRxData *out_buff)
ResponseCodes	NFC_Felica_Polling (unsigned int pollTimeout, felicaPolling *inData, felicaPollingOutput *outData)
ResponseCodes	NFC_APDU_Exchange (apduTxData *txData, apduRxData *rxData)
VasStatus	NFC_Terminal_Config (rawData *input, rawData *output)
VasStatus	NFC_TERMINAL_ReadConfig (rawData *id, rawData *output)
VasStatus	NFC_VAS_ReadConfig (rawData *id, rawData *output)
VasStatus	NFC_VAS_Activate (rawData *id, rawData *input, rawData *output)
VasStatus	NFC_VAS_Cancel (void)
VasStatus	NFC_VAS_UpdateConfig (rawData *id, rawData *input, rawData *output)
VasStatus	NFC_VAS_CancelConfig (rawData *id)
VasStatus	NFC_VAS_PreLoad (rawData *id, rawData *input, rawData *output)
VasStatus	NFC_VAS_CancelPreLoad (rawData *id)
VasStatus	NFC_VAS_Decrypt (rawData *id, rawData *input, rawData *output)

VasStatus	NFC_VAS_Action (rawData *id, int action, rawData *inData, rawData *outBuff)

Enumeration Type Documentation

? SDI_SW12

enum SDI_SW12

SDI Server Status Words

see ADK-SDI Programmers Guide - Status Word Coding

Enumerator	
SDI_SW12_NONE	no status word received, e.g. no connection
SDI_SW12_SUCCESS	all okay
SDI_SW12_TAG_ERROR	tag error
SDI_SW12_TAG_LENGTH_ERROR	tag length error
SDI_SW12_EXEC_ERROR	execution error
SDI_SW12_CANCELED_BY_USER	canceled by user
SDI_SW12_BUSY	SDI server is busy
SDI_SW12_TIMEOUT_PIN_ENTRY	timeout during PIN entry
SDI_SW12_TIMEOUT_CARD_REMOVAL	mag. stripe reading on hybrid readers
SDI_SW12_INTERCHAR_PIN_ENTRY	inter-character timeout during PIN entry
SDI_SW12_COMMAND_NOT_ALLOWED	command not allowed
SDI_SW12_MAIN_CONNECTION_USED	main connection (protocol type B) already in use

SDI_SW12_INVALID_FILE_CONTENT	invalid file content
SDI_SW12_FILE_ACCESS_ERROR	file access error
SDI_SW12_LOGIC_ERROR	logic error, e.g. wrong command order
SDI_SW12_SDI_PARAMETER_ERROR	parameter error
SDI_SW12_LUHN_CHECK_FAILED	LUHN check of PAN failed.
SDI_SW12_EXECUTION_ABORTED	execution aborted
SDI_SW12_EXECUTION_TIMEOUT	execution timeout
SDI_SW12_MESSAGE_LENGTH_ERROR	message length erro
SDI_SW12_NO_SDI_PLUGIN_AVAILABLE	no SDI plugin available
SDI_SW12_UNKNOWN_PLUGIN_ID	unknown plugin ID in the Instruction byte (INS)
SDI_SW12_UNKNOWN_PLUGING_ID	typo, obsolete use SDI_SW12_UNKNOWN_PLUGIN_ID
SDI_SW12_INVALID_PLUGIN_RESPONSE	invalid or no result data returned by the plugin
SDI_SW12_EPP_CONNECTION_ERROR	EPP connection error.
SDI_SW12_UNKNOWN_INS_BYTE	unknown Instruction (INS)
SDI_SW12_UNKNOWN_CLA_BYTE	unknown Class (CLA)
SDI_SW12_CMAC_ERROR	CMAC error.

SDI_SW12_CMAC_LENGTH_ERROR	CMAC length error.
SDI_SW12_CMAC_MISSING_ERROR	CMAC missing.
SDI_SW12_ENCRYPTION_ERROR	encryption error
SDI_SW12_ENCRYPTION_LENGTH_ERROR	encryption length error
SDI_SW12_ENCRYPTION_MISSING_ERROR	encryption missing error
SDI_SW12_DECRYPTION_ERROR	decryption error
SDI_SW12_DECRYPTION_LENGTH_ERROR	decryption length error
SDI_SW12_DECRYPTION_MISSING_ERROR	decryption mssing error
SDI_SW12_EXCESSIVE_PIN_REQUESTS	excessive PIN requests
SDI_SW12_LOW_BATTERY	low battery
SDI_SW12_NO_DUKPT_KEYS_LOADED	no DUKPT keys loaded
SDI_SW12_UNIT_TAMPERED	unit tampered
SDI_SW12_RECOVERY_MODE	SDI only allows Check For Update (20-1D) command. (Android only)
SDI_SW12_PIN_BYPASSED	PIN bypassed.
SDI_SW12_NO_MACTCH_FOR_CARD_VALIDATION	no match for card validation
SDI_SW12_SMART_CARD_REMOVED	smart card removed = error caused by card holder

SDI_SW12_SMART_CARD_ERROR_TRM	smart card error caused by terminal
SDI_SW12_SMART_CARD_ERROR	smart card error caused by ICC
SDI_SW12_TWO_CARDS	CTLS collision, multiple cards in NFC field.
SDI_SW12_SMART_CARD_ERR_INIT	smart card error caused for initialization
SDI_SW12_SMART_CARD_ERR_PARAM	smart card error caused passing invalid parameters
SDI_SW12_EMV_TLV_ERROR	
SDI_SW12_ERROR	**Deprecated:** any other error
SDI_SW12_TIMEOUT	**Deprecated:** card detection/removal
SDI_SW12_NOT_ALLOWED	**Deprecated:** not allowed for the moment, e.g. card removal
SDI_SW12_PARAMETER_ERROR	**Deprecated:** parameter error (only in relation with EMV commands)

? SDICLIENT_ERROR

enum SDICLIENT_ERROR

Additional Error Code complementing the different component interfaces by client side errors such as inter-process communication problems, concurrent use, parameter error detected on client side, etc. This enumeration is based on libsdiprotocol'S SDI Protocol Error Codes and can be read with SdiBase::getClientError() and getNfcClientError().

Enumerator

SDICLIENT_ERROR_NONE	no error on client side, error originates from server

SDICLIENT_ERROR_COMMUNICATION	read/write or protocol error
SDICLIENT_ERROR_CONCURRENT_USE	SDI connection used by other thread
SDICLIENT_ERROR_CONNECT	no connection to SDI server
SDICLIENT_ERROR_OVERFLOW	output buffer too small
SDICLIENT_ERROR_PARAM	function parameter wrong or NULL pointer not allowed
SDICLIENT_ERROR_OTHER	any other problem like thread creation, memory allocation, etc.
SDICLIENT_ERROR_NO_RECEIVE	returned by SDI_Send(): command successfully sent, but response for this command is suppressed, therefore, no SDI_Receive() must be called afterwards
SDICLIENT_ERROR_NOT_SUPPORTED	command not supported by this library
SDICLIENT_ERROR_NOT_ALLOWED	command not allowed to be sent

? SYSUploadType

enum SYSUploadType

Types for sw/file upload command

Enumerator

SYS_UPLOAD_SOFTWARE_UPDATE	Software update
SYS_UPLOAD_CONFIG_WHITELIST	Whitelist configuration: whitelist.json
SYS_UPLOAD_CONFIG_SENSITIVE_TAGS	Sensitive tags configuration: sensitivetags.json
SYS_UPLOAD_CONFIG_CARD_RANGES	Card ranges configuration: cardranges.json

SYS_UPLOAD_INSTALL_CP_PACKAGE	Install commerce platform package
SYS_UPLOAD_EMV_CONFIGURATION	EMV configuration package (uncompressed TAR file)

Function Documentation

? getNfcClientError()

enum SDICLIENT_ERROR libsdi::getNfcClientError ( )

Read client side error after NFC/VAS command invocation

? getNfcSW12()

enum SDI_SW12 libsdi::getNfcSW12 ( )

Read SDI Server status after NFC/VAS command invocation

? NFC_APDU_Exchange()

ResponseCodes libsdi::NFC_APDU_Exchange (apduTxData * *txData*,
apduRxData * *rxData*
)

Data transive of APDU protocol.

This command is not allowed on SDI Server external interface

Parameters
[in] txData apduTxData data to send
[out] rxData apduRxData data received

Returns
EMB_APP_COMMAND_NOT_SUPPORTED

? NFC_Callback_Test()

ResponseCodes libsdi::NFC_Callback_Test (void)

Undocumented function, just included because part of NFC_Interface.h.

Returns
EMB_APP_COMMAND_NOT_SUPPORTED

? NFC_Client_Init()

CL_STATUS libsdi::NFC_Client_Init (CONNECTION_TYPE *type*)

This is required to called once before any NFC/VAS is possible. Should be called with CL_TYPE_FUNCTION as client-server is not supported anymore When the SDI server response is not 90xx or there was a communication problem an appropriate CL_STATUS or CL_STATUS_GENERAL_ERROR is returned. The functions getNfcClientError and getNfcSW12 will provide the error indication.

Parameters
[in] type CL_TYPE_FUNCTION

Returns
CL_STATUS CL_STATUS_NFC_INITILIZED_ALREADY for subsequent calls

? NFC_Client_Init_CheckVer()

CL_STATUS libsdi::NFC_Client_Init_CheckVer (CONNECTION_TYPE *type*,
int *maj*,
int *min*,
int *bld*
)

Not supported

Returns
CL_STATUS_NOT_SUPPORTED

? NFC_Config_Init()

ResponseCodes libsdi::NFC_Config_Init (void)

According NFC documentation: Initializes NFC Configuration. But this function does nothing and always returns error.

Returns
EMB_APP_FAILED

? NFC_Felica_Exchange()

ResponseCodes libsdi::NFC_Felica_Exchange (felicaTxData * *in_buff*,
felicaRxData * *out_buff*
)

Data transive over Felica protocol

Parameters
[in] in_buff binary input
[out] out_buff binary output

Returns
ResponseCodes

? NFC_Felica_Polling()

ResponseCodes libsdi::NFC_Felica_Polling (unsigned int *pollTimeout*,
felicaPolling * *inData*,
felicaPollingOutput * *outData*
)

FeliCa Polling request

Parameters
[in] pollTimeout timeout in Milli Seconds
[in] inData felicaPolling input data
[out] outData felicaPollingOutput result data

Returns
ResponseCodes

? NFC_Free_Poll_Data()

void libsdi::NFC_Free_Poll_Data (pollRes * *outPollRes*)

Releases memory allocated in the pollRes *outPollRes when NFC_PT_Polling() was called.

? NFC_Free_Poll_Data_Full()

void libsdi::NFC_Free_Poll_Data_Full (pollResFull * *outPollRes*)

Releases memory allocated in the pollResFull *outPollRes when NFC_PT_PollingFull() was called.

? NFC_Get_Version()

ResponseCodes libsdi::NFC_Get_Version (rawData * *output*)

Returns ADK-NFC build and kernels versions as JSON string Depending on return code getNfcSW12() or getNfcClientError() might provide the error reason.

Parameters

[out] output data buffer for the version information

Returns

NFC result, EMB_APP_FAILED for other SW12, EMB_APP_COMM_ERROR for client side error

? NFC_Mifare_Authenticate()

ResponseCodes libsdi::NFC_Mifare_Authenticate (unsigned char *blockNumber*,
MIFARE_KEY_TYPE *keyType*,
rawData * *Key*
)

Authenticates the section with given block address with either key A or B.

Parameters

[in] blockNumber MIFARE block address within the section to authenticate '00' .. 'FF'
[in] keyType MIFARE_KEY_TYPE_A or MIFARE_KEY_TYPE_B
[in] Key MIFARE key (e.g. MIFARE classic crypto-1 key with 48 bit)

Returns

ResponseCodes

? NFC_Mifare_Decrement()

ResponseCodes libsdi::NFC_Mifare_Decrement (unsigned int *blockNum*,
int *amount*
)

Decrement MIFARE value block by amount and transfer to original location

Parameters

[in] blockNum source and destination block address

[in] amount 4 byte signed integer

Returns

ResponseCodes

? NFC_Mifare_Decrement_Only()

ResponseCodes libsdi::NFC_Mifare_Decrement_Only (unsigned int *blockNum*,
int *amount*
)

Decrement MIFARE value block by amount and store at transfer buffer

Parameters

[in] blockNum source block address

[in] amount 4 byte signed integer

Returns

ResponseCodes

? NFC_Mifare_Increment()

ResponseCodes libsdi::NFC_Mifare_Increment (unsigned int *blockNum*,
int *amount*
)

Increment MIFARE value block by amount and transfer to original location

Parameters

[in] blockNum source and destination block address

[in] amount 4 byte signed integer

Returns

ResponseCodes

? NFC_Mifare_Increment_Only()

ResponseCodes libsdi::NFC_Mifare_Increment_Only (unsigned int *blockNum*,
int *amount*
)

Increment MIFARE value block by amount and store at transfer buffer

Parameters
[in] blockNum source block address
[in] amount 4 byte signed integer

Returns
ResponseCodes

? NFC_Mifare_Read()

ResponseCodes libsdi::NFC_Mifare_Read (I_MIFARE_CARD_TYPE *m_cardType*,
unsigned int *StartBlockNum*,
unsigned int *blockAmount*,
rawData * *out_buff*
)

Read block data of up to 15 blocks

Parameters
[in] m_cardType I_MIFARE_CARD_TYPE
[in] StartBlockNum address of first block
[in] blockAmount number of blocks to read
[out] out_buff output buffer, required size is 16*blockAmount

Returns
ResponseCodes

? NFC_Mifare_Restore()

ResponseCodes libsdi::NFC_Mifare_Restore (unsigned int *blockNum*)

Write value from source block to transfer buffer

Parameters
[in] blockNum source block address

Returns

ResponseCodes

? NFC_Mifare_Transfer()

ResponseCodes libsdi::NFC_Mifare_Transfer (unsigned int *blockNum*)

Write value from transfer buffer to destination block

Parameters

[in] blockNum destination block address

Returns

ResponseCodes

? NFC_Mifare_Write()

ResponseCodes libsdi::NFC_Mifare_Write (I_MIFARE_CARD_TYPE *m_cardType*,
unsigned int *StartBlockNum*,
unsigned int *blockAmount*,
rawData * *in_buff*
)

Write block data of up to 15 blocks

Parameters

[in] m_cardType I_MIFARE_CARD_TYPE
[in] StartBlockNum address of first block
[in] blockAmount number of blocks to read
[in] in_buff data to write of size 16*blockAmount

Returns

ResponseCodes

? NFC_Ping()

ResponseCodes libsdi::NFC_Ping (rawData * *output*)

Return NFC Framework State of the NFC framework.

? NFC_PT_Activation()

ResponseCodes libsdi::NFC_PT_Activation (NFC_CARD_TYPE *cardtype*,
rawData * *rd_activationData*
)

Activates (selects) the card found during polling. When the SDI server response is not 90xx or there was a communication problem an appropriate CL_STATUS or CL_STATUS_GENERAL_ERROR is returned. The functions getNfcClientError and getNfcSW12 will provide the error indication.

Parameters

[in] cardtype	value from pollRes::cardInfo]n].cardType or pollResFull::cards_info_arr[n].m_modulation
[in] rd_activationData	value from pollRes::cards_info_arr[n].card_info or pollResFull::card_info_arr[n].mrd_UID

? NFC_PT_Cancel_Polling()

ResponseCodes libsdi::NFC_PT_Cancel_Polling (void)

Stop polling before timeout. Note: This command has to be send asynchronously while waiting for polling response. Since is not yet supported on SDI Server side there is also no implementation on client side.

Returns
EMB_APP_COMMAND_NOT_SUPPORTED

? NFC_PT_Close()

ResponseCodes libsdi::NFC_PT_Close ( )

Release NFC L1 driver. When the SDI server response is not 90xx or there was a communication problem an appropriate CL_STATUS or CL_STATUS_GENERAL_ERROR is returned. The functions getNfcClientError and getNfcSW12 will provide the error indication.

? NFC_PT_FieldOff()

ResponseCodes libsdi::NFC_PT_FieldOff ( )

Turns RF field off. When the SDI server response is not 90xx or there was a communication problem an appropriate CL_STATUS or CL_STATUS_GENERAL_ERROR is returned. The functions getNfcClientError and getNfcSW12 will provide the error indication.

? NFC_PT_FieldOn()

ResponseCodes libsdi::NFC_PT_FieldOn ( )

Turns RF field on. When the SDI server response is not 90xx or there was a communication problem an appropriate CL_STATUS or CL_STATUS_GENERAL_ERROR is returned. The functions getNfcClientError and getNfcSW12 will provide the error indication.

? NFC_PT_FtechBaud()

ResponseCodes libsdi::NFC_PT_FtechBaud (NFC_F_BAUD *baud*)

Changes NFC-F baud rate

Parameters

[in] baud value of NFC_F_BAUD

Returns

value of ResponseCodes

? NFC_PT_Open()

ResponseCodes libsdi::NFC_PT_Open ( )

Initialise NFC L1 driver. When the SDI server response is not 90xx or there was a communication problem an appropriate CL_STATUS or CL_STATUS_GENERAL_ERROR is returned. The functions getNfcClientError and getNfcSW12 will provide the error indication.

? NFC_PT_Polling()

ResponseCodes libsdi::NFC_PT_Polling (pollReq * *inPollReq*,
pollRes * *outPollRes*
)

Activates polling. See NFC documentation. When the SDI server response is not 90xx or there was a communication problem an appropriate CL_STATUS or CL_STATUS_GENERAL_ERROR is returned. The functions getNfcClientError and getNfcSW12 will provide the error indication.

? NFC_PT_PollingFull()

ResponseCodes libsdi::NFC_PT_PollingFull (pollReq * *inPollReq*,
pollResFull * *outPollRes*

)

Activates polling. See NFC documentation. When the SDI server response is not 90xx or there was a communication problem an appropriate CL_STATUS or CL_STATUS_GENERAL_ERROR is returned. The functions getNfcClientError and getNfcSW12 will provide the error indication.

? NFC_PT_TxRx()

ResponseCodes libsdi::NFC_PT_TxRx (NFC_CARD_TYPE *cardtype*,
rawData * *inBuff*,
rawData * *outBuff*
)

Sends and receives raw data using ISO 14443-3 protocol (31-08) This function is no more available at SDI server external interface

Returns
EMB_APP_COMMAND_NOT_SUPPORTED

? NFC_SerialClose()

CL_STATUS libsdi::NFC_SerialClose (void)

Not supported

Returns
CL_STATUS_NOT_SUPPORTED

? NFC_SerialOpen()

CL_STATUS libsdi::NFC_SerialOpen (void)

Not supported

Returns
CL_STATUS_NOT_SUPPORTED

? NFC_Set_Callback_Function()

ResponseCodes libsdi::NFC_Set_Callback_Function (rawData * *id*,
NfcCallbackFunction * *callbackFunction*

)

Set UI callback function handling text, status indicators (LEDs) and buzzer. So far, there is no client side implementation.

Returns
EMB_APP_COMMAND_NOT_SUPPORTED

? NFC_Terminal_Config()

VasStatus libsdi::NFC_Terminal_Config (rawData * *input*,
rawData * *output*
)

Terminal wide VAS configuration

Parameters
[in] input JSON string see Terminal Configuration Parameters
[out] output nothing - RFU

Returns
VasStatus

? NFC_TERMINAL_ReadConfig()

VasStatus libsdi::NFC_TERMINAL_ReadConfig (rawData * *id*,
rawData * *output*
)

Reads the most updated terminal configuration.Static parameter will be returned in case appID is unknown or NFC_VAS_PreLoad() issued without changing Terminal configuration.

Parameters
[in] id application unique identifier
[out] output JSON string most updated Read Terminal Configuration Parameters

Returns
VasStatus

? NFC_VAS_Action()

VasStatus libsdi::NFC_VAS_Action (rawData * *id*,

```
                                    int           action,
                                    rawData *     inData,
                                    rawData *     outBuff
                                   )
```

Key transfer between Counter Top and External PIN pad.

Not to be used at external SDI Server interface

Returns

VasStatus

? NFC_VAS_Activate()

```
VasStatus libsdi::NFC_VAS_Activate ( rawData *  id,
                                     rawData *  input,
                                     rawData *  output
                                    )
```

Activates NFC interface, runs through wallet kernel flow and retrieves VAS data.

Parameters

[in] id application unique identifier

[in] input JSON string set of dynamic parameters to be merged with configuration from data base.

[out] output JSON string VAS data received from the mobile.

Returns

VasStatus

? NFC_VAS_Cancel()

```
VasStatus libsdi::NFC_VAS_Cancel ( void   )
```

Stop VAS activate polling before timeout. Note: This command has to be send asynchronously while waiting for polling response. Since is not yet supported on SDI Server side there is also no implementation on client side.

Returns

EMB_APP_COMMAND_NOT_SUPPORTED

? NFC_VAS_CancelConfig()

VasStatus libsdi::NFC_VAS_CancelConfig (rawData * *id*)

Clears all the VAS configuration by application ID

Parameters
[in] id application unique identifier

Returns
VasStatus

? NFC_VAS_CancelPreLoad()

VasStatus libsdi::NFC_VAS_CancelPreLoad (rawData * *id*)

Clear preloaded configuration by application ID and pulls latest static configuration from data base.

Parameters
[in] id application unique identifier

Returns
VasStatus

? NFC_VAS_Decrypt()

VasStatus libsdi::NFC_VAS_Decrypt (rawData * *id*,
rawData * *input*,
rawData * *output*
)

Decrypts an encrypted VAS response.

Parameters
[in] id application unique identifier
[in] input The json in the same format of the Vas Data Response with the included encrypted message
[out] output The json in the same format of the Vas Data Response with the included decrypted message

Returns
VasStatus

? NFC_VAS_PreLoad()

```
VasStatus libsdi::NFC_VAS_PreLoad ( rawData *  id,
                                    rawData *  input,
                                    rawData *  output
                                  )
```

Configures the terminal with wallet specific parameters. NFC_VAS_Activate() has to be called to get VAS data. Only single PreLoaded configuration is available.

Parameters

[in] id application unique identifier

[in] input input Set of PreLoad parameters to be merged with configuration from data base.

[out] output none - RFU

Returns

VasStatus

? NFC_VAS_ReadConfig()

```
VasStatus libsdi::NFC_VAS_ReadConfig ( rawData *  id,
                                       rawData *  output
                                     )
```

Reads the most updated wallets configuration.

Parameters

[in] id application unique identifier

[out] output JSON string most updated configuration for terminal and all wallets

Returns

VasStatus

? NFC_VAS_UpdateConfig()

```
VasStatus libsdi::NFC_VAS_UpdateConfig ( rawData *  id,
                                         rawData *  input,
                                         rawData *  output
                                       )
```

Configures the terminal with wallet specific parameters.

Parameters

[in] id application unique identifier

[in] input JSON string set of parameters to configure ome or multiple wallets
[out] output none - RFU

Returns
VasStatus