TRF7970A RFID Reader Software Example

API Guide

DOCNUM-00.01.00

Contents

Chapter 1

Copyright

Please be aware that an important notice concerning availability, standard warranty, and use in critical applications of Texas Instruments semiconductor products and disclaimers thereto appears at the end of this document.

Texas Instruments
13532 N. Central Expressway
Dallas, TX 75243
www.ti.com

Chapter 2

Introduction

This Example detects ISO15693, Type 2, Type 3, Type 4A, Type 4B NFC/RFID tags. It then indicates the Tag type through LED's on the TRF7970A Booster pack. Information such as tag UID's and block data is sent out via a UART at 115200 Baud and can be read on a Computer.

The TRF7970A is an integrated analog front end and data framing system for a 13.56 MHz RFID reader system. Built-in programming options make it suitable for a wide range of applications both in proximity and vicinity RFID systems. The reader is configured by selecting the desired protocol in the control registers. Direct access to all control registers allows fine tuning of various reader parameters as needed.

The TRF7970A is interfaced to a MSP430G2553 through a SPI (serial) interface using a hardware USCI. The MCU is the master device and initiates all communication with the reader.

The anti-collision procedures (as described in the ISO standards 14443A/B and 15693) are implemented in the MCU firmware to help the reader detect and communicate with one PICC/VICC among several PICCs/VICCs.

Chapter 3

Disclaimer

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, modifications, enhancements, improvements, and other changes to its products and services at any time and to discontinue any product or service without notice. Customers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All products are sold subject to TI's terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its hardware products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by government requirements, testing of all parameters of each product is not necessarily performed.

TI assumes no liability for applications assistance or customer product design. Customers are responsible for their products and applications using TI components. To minimize the risks associated with customer products and applications, customers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any TI patent right, copyright, mask work right, or other TI intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information published by TI regarding third-party products or services does not constitute a license from TI to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of TI information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. Reproduction of this information with alteration is an unfair and deceptive business practice. TI is not responsible or liable for such altered documentation. Information of third parties may be subject to additional restrictions.

Resale of TI products or services with statements different from or beyond the parameters stated by TI for that product or service voids all express and any implied warranties for the associated TI product or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

TI products are not authorized for use in safety-critical applications (such as life support) where a failure of the TI product would reasonably be expected to cause severe personal injury or death, unless officers of the parties have executed an agreement specifically governing such use. Buyers represent that they have all necessary expertise in the safety and regulatory ramifications of their applications, and acknowledge and agree that they are solely responsible for all legal, regulatory and safety-related requirements concerning their products and any use of TI products in such safety-critical applications, notwithstanding any applications-related information or support that may be provided by TI. Further, Buyers must fully indemnify TI and its representatives against any damages arising out of the use of TI products in such safety-critical applications.

TI products are neither designed nor intended for use in military/aerospace applications or environments unless the TI products are specifically designated by TI as military-grade or "enhanced plastic." Only products designated by TI as military-grade meet military specifications. Buyers acknowledge and agree that any such use of TI products which TI has not designated as military-grade is solely at the Buyer's risk, and that they are solely responsible for compliance with all legal and regulatory requirements in connection with such use.

TI products are neither designed nor intended for use in automotive applications or environments unless the specific TI products are designated by TI as compliant with ISO/TS 16949 requirements. Buyers acknowledge and agree that, if they use any non-designated products in automotive applications, TI will not be responsible for any failure to meet such requirements.

You can obtain information on other Texas Instruments products and application solutions from www.ti.com.

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265

Chapter 4

Module Index

4.1 Modules

Here is a list of all modules:

Chapter 5

Data Structure Index

5.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 6

Module Documentation

6.1 NFC LLCP API Functions

Macros

- #define **LLCP_MAGIC_NUMBER_HIGH** 0x46
- #define **LLCP_MAGIC_NUMBER_MIDDLE** 0x66
- #define **LLCP_MAGIC_NUMBER_LOW** 0x6D
- #define LLCP_SSAP_CONNECT_SEND 0x20

  Source Service Access Point when sending.
- #define LLCP_SSAP_CONNECT_RECEIVED 0x04

  Source Service Access Point when receiving.
- #define DSAP_SERVICE_DISCOVERY_PROTOCOL 0x01

  Destination Service Access Point for discovery.
- #define **LLCP_MIU** 248
- #define **LLCP_MIUX_SIZE** (LLCP_MIU - 128)

Enumerations

- enum tLLCPParamaeter {
  LLCP_VERSION = 0x01, LLCP_MIUX = 0x02, LLCP_WKS = 0x03, LLCP_LTO = 0x04,
  LLCP_RW = 0x05, LLCP_SN = 0x06, LLCP_OPT = 0x07, LLCP_SDREQ = 0x08,
  LLCP_SDRES = 0x09, **LLCP_ERROR** }

  LLCP Parameter Enumerations.
- enum tLLCPPduPtype {
  LLCP_SYMM_PDU = 0x00, LLCP_PAX_PDU = 0x01, LLCP_AGF_PDU = 0x02, LLCP_UI_PDU = 0x03,
  LLCP_CONNECT_PDU = 0x04, LLCP_DISC_PDU = 0x05, LLCP_CC_PDU = 0x06, LLCP_DM_PDU = 0x07,
  LLCP_FRMR_PDU = 0x08, LLCP_SNL_PDU = 0x09, LLCP_I_PDU = 0x0C, LLCP_RR_PDU = 0x0D,
  LLCP_RNR_PDU = 0x0E, LLCP_RESERVED_PDU = 0x0F }

  PDU Type Enumerations.
- enum tLLCPConnectionStatus { LLCP_CONNECTION_IDLE = 0x00,
  LLCP_CONNECTION_ESTABLISHED, LLCP_CONNECTION_SENDING,
  LLCP_CONNECTION_RECEIVING }

  LLCP Connection Status Enumeration.
- enum tServiceName { **NPP_SERVICE** = 0, **SNEP_SERVICE**, **HANDOVER_SERVICE** }

  Service Name Enumerations - Only support SNEP_SERVICE.

- enum tDisconnectModeReason {
  DM_REASON_LLCP_RECEIVED_DISC_PDU = 0x00,
  DM_REASON_LLCP_RECEIVED_CONNECTION_ORIENTED_PDU = 0x01,
  DM_REASON_LLCP_RECEIVED_CONNECT_PDU_NO_SERVICE = 0x02,
  DM_REASON_LLCP_PROCESSED_CONNECT_PDU_REQ_REJECTED = 0x03,
  DM_REASON_LLCP_PERMNANTLY_NOT_ACCEPT_CONNECT_WITH_SAME_SSAP = 0x10,
  DM_REASON_LLCP_PERMNANTLY_NOT_ACCEPT_CONNECT_WITH_ANY_SSAP = 0x11,
  DM_REASON_LLCP_TEMMPORARILY_NOT_ACCEPT_PDU_WITH_SAME_SSSAPT = 0x20,
  DM_REASON_LLCP_TEMMPORARILY_NOT_ACCEPT_PDU_WITH_ANY_SSSAPT = 0x21 }

  Disconnected Mode Reasons Enumerations.

Functions

- void LLCP_init (void)
- uint16_t LLCP_getLinkTimeOut (void)
- uint8_t LLCP_addTLV (tLLCPParamaeter eLLCPparam, uint8_t *pui8TLVBufferPtr)
- void LLCP_processTLV (uint8_t *pui8TLVBufferPtr)
- uint8_t LLCP_stateMachine (uint8_t *pui8PduBufferPtr)
- uint8_t LLCP_processReceivedData (uint8_t *pui8RxBuffer, uint16_t ui16PduLength)
- uint8_t LLCP_sendSYMM (uint8_t *pui8PduBufferPtr)
- uint8_t LLCP_sendPAX (uint8_t *pui8PduBufferPtr)
- uint8_t LLCP_sendCONNECT (uint8_t *pui8PduBufferPtr)
- uint8_t LLCP_sendDISC (uint8_t *pui8PduBufferPtr)
- uint8_t LLCP_sendCC (uint8_t *pui8PduBufferPtr)
- uint8_t LLCP_sendDM (uint8_t *pui8PduBufferPtr, tDisconnectModeReason eDmReason)
- uint8_t LLCP_sendI (uint8_t *pui8PduBufferPtr)
- uint8_t LLCP_sendRR (uint8_t *pui8PduBufferPtr)

Variables

- tLLCPPduPtype **g_eNextPduQueue**
- tLLCPConnectionStatus **g_eLLCPConnectionStatus**
- uint8_t **g_ui8dsapValue**
- uint8_t **g_ui8ssapValue**
- tServiceName **g_eCurrentServiceEnabled**
- uint8_t **g_ui8NSNR**
- tDisconnectModeReason **g_eDMReason**
- uint16_t **g_ui16LLCPlto**
- uint8_t **g_ui8LLCPmiu**
- uint8_t **g_ui8LLCPmiuOffset**

6.1.1 Detailed Description

Logical Link Control Protocol is the NFC transport layer used to open / close virtual link used to transfer NDEFs between two devices in Peer to Peer mode via the Simple NDEF Exchange Protocol. For more information on LLCP please read the Logical Link Control Protocol Specification Version 1.1.

6.1.2 Enumeration Type Documentation

6.1.2.1 enum tDisconnectModeReason

Disconnected Mode Reasons Enumerations.

Enumerator

DM_REASON_LLCP_RECEIVED_DISC_PDU See LLCP Section 4.3.8.

6.1.2.2 enum tLLCPConnectionStatus

LLCP Connection Status Enumeration.

Enumerator

LLCP_CONNECTION_IDLE No TX/RX ongoing.

LLCP_CONNECTION_ESTABLISHED When a virtual link is created either when we send a CONNECT PDU and receive a CC PDU, or when we receive a CONNECT PDU and respond a CC PDU.

LLCP_CONNECTION_SENDING When sending data via SNEP.

LLCP_CONNECTION_RECEIVING When receiving data via SNEP.

6.1.2.3 enum tLLCPParamaeter

LLCP Parameter Enumerations.

Enumerator

LLCP_VERSION See LLCP 4.5.1.

LLCP_MIUX See LLCP 4.5.2.

LLCP_WKS See LLCP 4.5.3.

LLCP_LTO See LLCP 4.5.4.

LLCP_RW See LLCP 4.5.5.

LLCP_SN See LLCP 4.5.6.

LLCP_OPT See LLCP 4.5.7.

LLCP_SDREQ See LLCP 4.5.8.

LLCP_SDRES See LLCP 4.5.9.

6.1.2.4 enum tLLCPPduPtype

PDU Type Enumerations.

Enumerator

LLCP_SYMM_PDU See LLCP 4.3.1.

LLCP_PAX_PDU See LLCP 4.3.2.

LLCP_AGF_PDU See LLCP 4.3.3.

LLCP_UI_PDU See LLCP 4.3.4.

LLCP_CONNECT_PDU See LLCP 4.3.5.

LLCP_DISC_PDU See LLCP 4.3.6.

LLCP_CC_PDU See LLCP 4.3.7.

LLCP_DM_PDU See LLCP 4.3.8.

LLCP_FRMR_PDU See LLCP 4.3.9.

LLCP_SNL_PDU See LLCP 4.3.10.

LLCP_I_PDU See LLCP 4.3.11.

LLCP_RR_PDU See LLCP 4.3.12.

LLCP_RNR_PDU See LLCP 4.3.13.

LLCP_RESERVED_PDU See LLCP 4.3.14.

6.1.3 Function Documentation

6.1.3.1 uint8_t LLCP_addTLV (tLLCPParamaeter *eLLCPparam*, uint8_t * *pui8TLVBufferPtr*)

LLCP_addTLV - Adds a LLCP parameter to the LLCP PDU with the Type Length Value (TLV) format.

Parameters

eLLCPparam	is the LLCP type that will be added.
pui8TLVBufferPtr	is the pointer where the TLV will be written at.

The *eLLCPparam* parameter can be any of the following:

- **LLCP_VERSION** - Version Number
- **LLCP_MIUX** - Maximum Information Unit Extension
- **LLCP_WKS** - Well-Known Service List
- **LLCP_LTO** - Link Timeout
- **LLCP_RW** - Receive Window Size
- **LLCP_SN** - Service Name
- **LLCP_OPT** - Option
- **LLCP_SDREQ** - Service Discovery Request
- **LLCP_SDRES** - Service Discovery Response
- **LLCP_ERROR** - Reserved (used ro return length of 0)

This function is used to add a LLCP Parameter to the LLCP PDU to include more information about the LLCP layer. This function must be called inside LLCP_sendCONNECT(), LLCP_sendCC(), NFCDEP_sendATR_REQ() and NFCDEP_sendATR_RES().

Returns

ui8PacketLength Length of the LLCP Parameter added to the LLCP.

6.1.3.2 uint16_t LLCP_getLinkTimeOut (void)

LLCP_getLinkTimeOut - Gets link timout

This function returns the Link Timeout which may be modified if the LLCP_processTLV() function processes a LLCP_LTO TLV.

Returns

g_ui16LLCPlto the link timeout.

6.1.3.3 void LLCP_init (void)

LLCP_init - Initializes the Logical Link Control Protocol layer.

This function must be called prior to any other function offer by the LLCP driver. This function initializes the acknowledge packets, Maximum information units, current service enabled, the next PDU for the LLCP_stateMachine(), and initializes the SNEP layer with SNEP_init().

Returns

None

6.1.3.4 uint8_t LLCP_processReceivedData (uint8_t * *pui8RxBuffer,* uint16_t *ui16PduLength*)

LLCP_processReceivedData - Processes LLCP Data Received.

Parameters

pui8RxBuffer	is the start pointer of the LLCP data received.
ui16PduLength	is the length of the LLCP PDU received.

This function is used to handle the LLCP portion of the DEP_REQ / DEP_RES PDU. This function must be called inside NFCDEP_processReceivedRequest() and NFCDEP_processReceivedData().It currently does not support to handle the following PDUs : LLCP_PAX_PDU, LLCP_AGF_PDU, LLCP_UI_PDU, LLCP_FRMR_PDU, LLCP_SNL_PDU, LLCP_RNR_PDU, and LLCP_RESERVED_PDU.

Returns

eLLCPStatus is the boolean status if the command was processed (1) or not (0).

6.1.3.5 void LLCP_processTLV (uint8_t * *pui8TLVBufferPtr*)

LLCP_processTLV - Processes the LLCP Parameter TLV.

Parameters

pui8TLVBufferPtr	is the pointer to the Type value of the TLV.

This function processes the LLCP Parameters included in the ATR_RES. This function must be called inside the NFCDEP_processReceivedData(), to initialize the g_ui8LLCPmiu and g_ui16LLCPlto if they are included as part of ATR_RES.

Returns

None

6.1.3.6 uint8_t LLCP_sendCC (uint8_t * *pui8PduBufferPtr*)

LLCP_sendCC - Send CC message

Parameters

pui8PduBufferPtr	is the start pointer to store the CC PDU.

This function adds a CC PDU starting at pui8PduBufferPtr.For more details on this PDU read LLCP V1.1 Section 4.3.7.

Returns

ui8IndexTemp is the length of the CC PDU.

6.1.3.7 uint8_t LLCP_sendCONNECT (uint8_t * *pui8PduBufferPtr*)

LLCP_sendCONNECT - Send CONNECT message

Parameters

pui8PduBufferPtr	is the start pointer to store the CONNECT PDU.

This function adds a CONNECT PDU starting at pui8PduBufferPtr.For more

Returns

ui8IndexTemp is the length of the CONNECT PDU.

6.1.3.8 uint8_t LLCP_sendDISC (uint8_t * *pui8PduBufferPtr*)

LLCP_sendDISC - Send DISC message

Parameters

pui8PduBufferPtr	is the start pointer to store the DISC PDU.

This function adds a DISC PDU starting at pui8PduBufferPtr.For more details on this PDU read LLCP V1.1 Section 4.3.6.

Returns

ui8IndexTemp is the length of the DISC PDU.

6.1.3.9 uint8_t LLCP_sendDM (uint8_t * *pui8PduBufferPtr,* **tDisconnectModeReason** *eDmReason*)

LLCP_sendDM - Send DM message

Parameters

pui8PduBufferPtr	is the start pointer to store the DM PDU.
eDmReason	is the enumeration of the disconnection reason.

The *eDmReason* parameter can be any of the following:

- **DM_REASON_LLCP_RECEIVED_DISC_PDU**
- **DM_REASON_LLCP_RECEIVED_CONNECTION_ORIENTED_PDU**
- **DM_REASON_LLCP_RECEIVED_CONNECT_PDU_NO_SERVICE**
- **DM_REASON_LLCP_PROCESSED_CONNECT_PDU_REQ_REJECTED**
- **DM_REASON_LLCP_PERMNANTLY_NOT_ACCEPT_CONNECT_WITH_SAME_SSAP**
- **DM_REASON_LLCP_PERMNANTLY_NOT_ACCEPT_CONNECT_WITH_ANY_SSAP**
- **DM_REASON_LLCP_TEMMPORARILY_NOT_ACCEPT_PDU_WITH_SAME_SSSAPT**
- **DM_REASON_LLCP_TEMMPORARILY_NOT_ACCEPT_PDU_WITH_ANY_SSSAPT**

This function adds a DM PDU starting at pui8PduBufferPtr with a dm_reason. For more details on this PDU read LLCP V1.1 Section 4.3.8.

Returns

ui8IndexTemp is the length of the DM PDU.

6.1.3.10 uint8_t LLCP_sendI (uint8_t * *pui8PduBufferPtr*)

LLCP_sendI - Send I message

Parameters

pui8PduBufferPtr	is the start pointer to store the I PDU.

This function adds a I PDU starting at pui8PduBufferPtr.For more details on this PDU read LLCP V1.1 Section 4.3.10.

Returns

ui8IndexTemp is the length of the I PDU.

6.1.3.11 uint8_t LLCP_sendPAX (uint8_t * *pui8PduBufferPtr*)

LLCP_sendPAX - Send PAX message

Parameters

pui8PduBufferPtr	is the start pointer to store the PAX PDU.

This function adds a PAX PDU starting at pui8PduBufferPtr.For more

Returns

ui8IndexTemp is the length of the PAX PDU.

6.1.3.12 uint8_t LLCP_sendRR (uint8_t * *pui8PduBufferPtr*)

LLCP_sendRR - Send RR message

Parameters

pui8PduBufferPtr	is the start pointer to store the RR PDU.

This function adds a RR PDU starting at pui8PduBufferPtr.For more details on this PDU read LLCP V1.1 Section 4.3.11.

Returns

ui8IndexTemp is the length of the RR PDU.

6.1.3.13 uint8_t LLCP_sendSYMM (uint8_t * *pui8PduBufferPtr*)

LLCP_sendSYMM - Send SYMM message

Parameters

pui8PduBufferPtr	is the start pointer to store the SYMM PDU.

This function adds a SYMM PDU starting at pui8PduBufferPtr.For more

Returns

ui8IndexTemp is the length of the SYMM PDU.

6.1.3.14 uint8_t LLCP_stateMachine (uint8_t * *pui8PduBufferPtr*)

LLCP_stateMachine - Prepares the LLCP packet to be transmitted.

Parameters

pui8PduBufferPtr	is the start pointer to add the LLCP PDU.

This function is used to add the LLCP portion of the DEP_REQ / DEP_RES PDU. This function must be called inside NFCDEP_sendDEP_REQ() and NFCDEP_sendDEP_RES(). It currently does not support to send the following PDUs : LLCP_PAX_PDU, LLCP_AGF_PDU, LLCP_UI_PDU, LLCP_FRMR_PDU, LLCP_SNL_PDU, LLCP_RNR_PDU, and LLCP_RESERVED_PDU.

Returns

ui8PacketLength is the length of the LLCP PDU added to the pui8PduBufferPtr.

6.2 NFC A API Functions

Functions

- void NFC_A_setCIDSettings (bool bNFCACIDEnable, uint8_t ui8NfcACID)
- void NFC_A_init (void)
- void NFC_A_send_SENS_REQ (void)
- void NFC_A_send_ALL_REQ (void)
- void NFC_A_send_SDD_REQ (uint8_t ui8CascadeLevel)
- void NFC_A_send_SEL_REQ (uint8_t ui8CascadeLevel)
- void NFC_A_send_SENS_RES (void)
- void NFC_A_send_SDD_RES (uint8_t ui8CascadeLevel)
- void NFC_A_send_SEL_RES (uint8_t ui8CascadeLevel)
- uint8_t NFC_A_processSDDResponse (uint8_t ∗pui8RxBuffer, uint8_t ui8CascadeLevel)
- uint8_t NFC_A_processSENSRes (uint8_t ∗pui8RxBuffer, uint8_t ui8Length)
- uint8_t NFC_A_processSELResponse (uint8_t ∗pui8RxBuffer, uint8_t ui8Length)
- uint8_t NFC_A_check_SEL_REQ (uint8_t ui8CascadeLevel, uint8_t ∗pui8Payload, uint8_t ui8Length)
- uint8_t NFC_A_isInitiatorSelected (uint8_t ui8CascadeLevel)
- uint8_t NFC_A_isTargetSelected (uint8_t ui8CascadeLevel)
- void NFC_A_setDepSupport (bool bIsoSupport, bool bDepSupport)
- uint8_t NFC_A_isNfcDepSupported (void)
- uint8_t NFC_A_isIsoDepSupported (void)
- void NFC_A_sendRATS (void)
- void NFC_A_sendPPS_REQ (uint8_t ui8Bitrate)
- void NFC_A_stateMachine (void)
- tUNfcAState NFC_A_processReceivedData (uint8_t ∗pui8RxBuffer, uint8_t ui8Length)
- void NFC_A_setAutoSDD (bool bAutoSDD)
- void NFC_A_setNfcAId (uint8_t ∗ui8NewNfcAId, uint8_t ui8Length)
- uint16_t NFC_A_getSFGT (void)
- uint16_t NFC_A_getFWT (void)
- uint8_t NFC_A_getSAK (void)
- void NFC_A_getNfcAId (uint8_t ∗∗pui8NfcAId, uint8_t ∗ui8NfcAIdLength)

6.2.1 Detailed Description

NFC A is a NFC Layer used by NFC_Initiator and NFC_Target layers to activate the NFC-A technology. For more information on NFC-A please read the NFC Digital Specification Version 1.1.

6.2.2 Function Documentation

6.2.2.1 uint8_t NFC_A_check_SEL_REQ (uint8_t *ui8CascadeLevel*, uint8_t ∗ *pui8Payload*, uint8_t *ui8Length*)

NFC_A_check_SEL_REQ - Checks the received SEL REQ from the PCD. (PICC Mode)

Parameters

ui8CascadeLevel	is the current cascade level.
pui8Payload	is the pointer to the received request from the PCD.
ui8Length	is the length of the received response.

This function checks the SEL request is valid based on the UID and BCC.

Returns

ui8Status returns STATUS_SUCCESS if the SEL request is valid, otherwise it returns STATUS_FAIL.

6.2.2.2 uint16_t NFC_A_getFWT (void)

NFC_A_getFWT - This function returns the FWT (PCD Mode)

This function returns the g_ui16NfcAFwt.

Returns

g_ui16NfcAFwt the max time the PCD should wait before timing out.

6.2.2.3 void NFC_A_getNfcAId (uint8_t ** *pui8NfcAId,* uint8_t * *ui8NfcAIdLength*)

NFC_A_getNfcAId - This function returns the NFC-A UID (PICC or PCD Mode)

Parameters

pui8NfcAId	is a double pointer to be modified with the NFCAId buffer.
ui8NfcAIdLength	is a pointer to store the NFC ID Length.

This function returns the NFC-A UID.

Returns

None.

6.2.2.4 uint8_t NFC_A_getSAK (void)

NFC_A_getSAK - This function returns the SAK response (PCD Mode)

This function returns the g_ui8NfcASak.

Returns

g_ui8NfcASak the PICC response to the SEL REQ from the PCD.

6.2.2.5 uint16_t NFC_A_getSFGT (void)

NFC_A_getSFGT - This function returns the SFGT (PCD Mode)

This function returns the g_ui16NfcASfgt.

Returns

g_ui16NfcASfgt the guard time, the PCD must delay before sending a command to the PICC.

6.2.2.6 void NFC_A_init (void)

NFC_A_init - Initialize the NFC A driver.

This function must be called prior to any other function offered by the NFC A driver. This function initializes the NFC DEP and ISO support for PICC and PCD, the length of the tag, driver current state, CID, and SAK response.

Returns

None.

6.2.2.7 uint8_t NFC_A_isInitiatorSelected (uint8_t *ui8CascadeLevel*)

NFC_A_isInitiatorSelected - Checks if the initiator is selected. (PCD Mode)

Parameters

ui8CascadeLevel	is the current cascade level.

This function checks if the PCD activated the PICC.

Returns

ui8Status returns STATUS_SUCCESS if the initiator is selected, otherwise it returns STATUS_FAIL.

6.2.2.8 uint8_t NFC_A_isIsoDepSupported (void)

NFC_A_isIsoDepSupported - Checks if ISO DEP is supported.

This function checks if ISO DEP is supported.

Returns

ui8RetStatus returns STATUS_SUCCESS if it is supported, otherwise it returns STATUS_FAIL.

6.2.2.9 uint8_t NFC_A_isNfcDepSupported (void)

NFC_A_isNfcDepSupported - Checks if NFC DEP is supported.

This function checks if NFC DEP is supported.

Returns

ui8RetStatus returns STATUS_SUCCESS if it is supported, otherwise it returns STATUS_FAIL.

6.2.2.10 uint8_t NFC_A_isTargetSelected (uint8_t *ui8CascadeLevel*)

NFC_A_isTargetSelected - Checks if the target is selected. (PICC Mode)

Parameters

ui8CascadeLevel	is the current cascade level.

This function checks if the PICC is activated.

Returns

ui8Status returns STATUS_SUCCESS if the target is selected, otherwise it returns STATUS_FAIL.

6.2.2.11 tUNfcAState NFC_A_processReceivedData (uint8_t * *pui8RxBuffer,* uint8_t *ui8Length*)

NFC_A_processReceivedData - This function processes the response from the PICC. (PCD Mode)

Parameters

pui8RxBuffer	is a pointer to the recevied data.
ui8Length	is the length of received response.

This function processes the response to the RATS and PPS commands.

Returns

g_eNfcACurrentState the current NFC-A state.

6.2.2.12 uint8_t NFC_A_processSDDResponse (uint8_t * *pui8RxBuffer,* uint8_t *ui8CascadeLevel*)

NFC_A_processSDDResponse - Processes the received SDD response from the PICC.

Parameters

pui8RxBuffer	is the pointer to the received response from the PICC.
ui8CascadeLevel	is the cascade level.

This function checks the SDD response from the tag, and stores the UID in the g_pui8NfcAId buffer.

Returns

ui8Status returns STATUS_SUCCESS if the SDD response is valid, otherwise it returns STATUS_FAIL.

6.2.2.13 uint8_t NFC_A_processSELResponse (uint8_t * *pui8RxBuffer,* uint8_t *ui8Length*)

NFC_A_processSELResponse - Processes the received SEL response from the PICC.

Parameters

pui8RxBuffer	is the pointer to the received response from the PICC.
ui8Length	is the length of the received response.

This function checks the SEL response and stores the SAK response in g_ui8NfcASak.

Returns

ui8Status returns STATUS_SUCCESS if the SEL response is valid, otherwise it returns STATUS_FAIL.

6.2.2.14 uint8_t NFC_A_processSENSRes (uint8_t * *pui8RxBuffer,* uint8_t *ui8Length*)

NFC_A_processSENSRes - Processes the received SENS response from the PICC.

Parameters

pui8RxBuffer	is the pointer to the received response from the PICC.
ui8Length	is the length of the received response.

This function checks the SENS response and sets the size of the PICC UID.

Returns

ui8Status returns STATUS_SUCCESS if the SENS response is valid, otherwise it returns STATUS_FAIL.

6.2.2.15 void NFC_A_send_ALL_REQ (void)

NFC_A_send_ALL_REQ - Sends the ALL REQ to the PICC.

This function sends an ALL Request - first PCD command used to wake / activate NFC-A tags.

Returns

None.

6.2.2.16 void NFC_A_send_SDD_REQ (uint8_t *ui8CascadeLevel*)

NFC_A_send_SDD_REQ - Sends the SDD REQ to the PICC.

Parameters

ui8CascadeLevel	is the cascade level which determines the first byte of the SDD Request.

This function sends a SDD Request based on the indicated cascade level.

Returns

None.

6.2.2.17 void NFC_A_send_SDD_RES (uint8_t *ui8CascadeLevel*)

NFC_A_send_SDD_RES - Sends the SDD RES to the PCD.

Parameters

ui8CascadeLevel	is the cascade level which determines what response to send to the PCD depending on the UID length.

This function sends a SDD Response to the PCD.

Returns

None.

6.2.2.18 void NFC_A_send_SEL_REQ (uint8_t *ui8CascadeLevel*)

NFC_A_send_SEL_REQ - Sends the SEL REQ to the PICC.

Parameters

ui8CascadeLevel	is the cascade level which determines the first byte of the SEL Request.

This function sends a SEL Request based on the indicated cascade level.

Returns

None.

6.2.2.19 void NFC_A_send_SEL_RES (uint8_t *ui8CascadeLevel*)

NFC_A_send_SEL_RES - Sends the SEL RES to the PCD.

Parameters

ui8CascadeLevel	is the cascade level which determines what response to send to the PCD depending on the UID length.

This function sends a SEL Response to the PCD. Once the complete UID has been sent to the PCD. The ISO and NFC DEP flags will be indicated on the 5th and 6th bits of the SAK response.

Returns

None.

6.2.2.20 void NFC_A_send_SENS_REQ (void)

NFC_A_send_SENS_REQ - Sends the SENS REQ to the PICC.

This function sends a SENS Request - first PCD command used to activate NFC-A tags.

Returns

None.

6.2.2.21 void NFC_A_send_SENS_RES (void)

NFC_A_send_SENS_RES - Sends the SENS RES to the PCD.

This function sends a SENS Response to the PCD including the size of the UID.

Returns

None.

6.2.2.22 void NFC_A_sendPPS_REQ (uint8_t *ui8Bitrate*)

NFC_A_sendPPS_REQ - This function sends PPS Req command. (PCD Mode)

Parameters

ui8Bitrate	the PPS bitrate byte.

This function sends PPS REQ to the PICC, to increase the bitrate communication.

Returns

None.

6.2.2.23 void NFC_A_sendRATS (void)

NFC_A_sendRATS - This function sends RATs command. (PCD Mode)

This function sends RATS command to the PICC.

Returns

None.

6.2.2.24 void NFC_A_setAutoSDD (bool *bAutoSDD*)

NFC_A_setAutoSDD - This function sets the AutoSDD enable flag. (PICC Mode)

Parameters

bAutoSDD	is the AutoSDD flag.

This function sets the AutoSDD enable flag.

Returns

None.

6.2.2.25 void NFC_A_setCIDSettings (bool *bNFCACIDEnable,* uint8_t *ui8NfcACID*)

NFC_A_setCIDSettings - Sets the CID Enable flag and CID value.

Parameters

bNFCACIDEnable	the CID Enable flag.
ui8NfcACID	the CID number.

This function sets the NFC CID enable flag. When the PICC supports CID the code the PCD code will use the ui8NfcACID.

Returns

None.

6.2.2.26 void NFC_A_setDepSupport (bool *bIsoSupport,* bool *bDepSupport*)

NFC_A_setDepSupport - Sets the ISO and DEP support for the NFC-A driver.

Parameters

bIsoSupport	the ISO support flag.
bDepSupport	the NFC DEP support flag.

This function sets the NFC DEP and ISO DEP support. If Auto-SDD is enabled by the user, it enables the Auto-SDD in the transceiver. (only recommended to enable Auto-SDD when testing NFC Wave 1 Certifcation).

Returns

None.

6.2.2.27 void NFC_A_setNfcAId (uint8_t * *ui8NewNfcAId,* uint8_t *ui8Length*)

NFC_A_setNfcAId - This function sets the UID of the NFC-A when PICC mode is activated. (PICC Mode)

Parameters

ui8NewNfcAId	is the pointer to the NFCAId.
ui8Length	is the length of the UID. (Must be 4,7, or 10)

This function sets the ID and the size of the PICC UID.

Returns

None.

6.2.2.28 void NFC_A_stateMachine (void)

NFC_A_stateMachine - This function handles the activation of the PICC. (PCD Mode)

This function sends the RATS, PPS (optional), and activates the T4T tags.

Returns

None.

6.3 NFC B API Functions

Functions

- void NFC_B_init (void)
- void NFC_B_send_SENSB_REQ (void)
- void NFC_B_send_ALLB_REQ (void)
- void NFC_B_send_SLPB_REQ (void)
- void NFC_B_send_ATTRIB_REQ (void)
- void NFC_B_stateMachine (void)
- tNfcBPollState NFC_B_processReceivedRsp (uint8_t *pui8RxBuffer, uint8_t ui8Length)
- void NFC_B_send_SENSB_RES (void)
- void NFC_B_send_SLPB_RES (void)
- void NFC_B_send_ATTRIB_RES (void)
- uint8_t NFC_B_getSENSBN (uint8_t *pui8Payload, uint8_t ui8Length)
- uint8_t NFC_B_checkSENSBAfi (uint8_t *pui8Payload, uint8_t ui8Length)
- uint8_t NFC_B_checkAttribId (uint8_t *pui8Payload, uint8_t ui8Length)
- uint8_t NFC_B_checkSLPBId (uint8_t *pui8Payload, uint8_t ui8Length)
- uint8_t NFC_B_checkAttribParam (uint8_t *pui8Payload, uint8_t ui8Length)
- uint8_t NFC_B_setNfcBId (uint8_t *ui8NewNfcBId, uint8_t ui8Length)
- void NFC_B_getNfcBId (uint8_t **pui8NfcBId, uint8_t *ui8NfcBIdLength)
- uint16_t NFC_B_getSFGT (void)
- uint16_t NFC_B_getFWT (void)
- bool NFC_B_checkDIDSupported (void)
- bool NFC_B_isSelected (void)
- bool NFC_B_isISOCompliant (void)

6.3.1 Detailed Description

NFC B is a NFC Layer used by NFC_Initiator and NFC_Target layers to activate the NFC-B technology. For more information on NFC-B please read the NFC Digital Specification Version 1.1.

6.3.2 Function Documentation

6.3.2.1 uint8_t NFC_B_checkAttribId (uint8_t * *pui8Payload*, uint8_t *ui8Length*)

NFC_B_checkAttribId - Check if the ID number from the ATTRIB Request is correct. (PICC mode)

Parameters

pui8Payload	is the pointer to the start of the ATTRIB REQ.
ui8Length	is the length of the ATTRIB REQ.

This function checks if the ID the ATTRIB command corresponds to the emulated PICC.

Returns

ui8Status is STATUS_SUCCESS if the ID is valid, otherwise it returns STATUS_FAIL.

6.3.2.2 uint8_t NFC_B_checkAttribParam (uint8_t * *pui8Payload*, uint8_t *ui8Length*)

NFC_B_checkAttribParam - Check if the param values from the ATTRIB Request are correct. (PICC mode)

Parameters

pui8Payload	is the pointer to the start of the ATTRIB REQ.
ui8Length	is the length of the ATTRIB REQ.

This function checks if the params from the ATTRIB command are valid. This function handles test cases BI_08_10 and BI_08_13

Returns

ui8Status is STATUS_SUCCESS if the params are valid, otherwise it returns STATUS_FAIL.

6.3.2.3 bool NFC_B_checkDIDSupported (void)

NFC_B_checkDIDSupported - This function returns if the PICC supports DID. (PCD Mode)

This function returns if the PICC supports DID.

Returns

a boolean flag true if the the PICC supports DID, otherwise false.

6.3.2.4 uint8_t NFC_B_checkSENSBAfi (uint8_t * *pui8Payload,* uint8_t *ui8Length*)

NFC_B_checkSENSBAfi - Check if the AFI from the SENSB Request is correct. (PICC mode)

Parameters

pui8Payload	is the pointer to the start of the SENSB REQ.
ui8Length	is the length of the SENSB REQ.

This function checks if the AFI paramater from the SENSB Request is 0x00.

Returns

ui8Status is STATUS_SUCCESS if the AFI value is 0x00, otherwise it returns STATUS_FAIL.

6.3.2.5 uint8_t NFC_B_checkSLPBId (uint8_t * *pui8Payload,* uint8_t *ui8Length*)

NFC_B_checkSLPBId - Check if the ID number from the SLP Request is correct. (PICC mode)

Parameters

pui8Payload	is the pointer to the start of the SLPB REQ.
ui8Length	is the length of the SLPB REQ.

This function checks if the ID the SLPB command corresponds to the emulated PICC.

Returns

ui8Status is STATUS_SUCCESS if the ID is valid, otherwise it returns STATUS_FAIL.

6.3.2.6 uint16_t NFC_B_getFWT (void)

NFC_B_getFWT - This function returns the g_ui16NfcBFwt (PCD Mode)

This function returns max timeout (mS) the transceiver should wait for a PICC response.

Returns

g_ui16NfcBFwt is the max timeout time the PCD must wait for a response.

6.3.2.7 void NFC_B_getNfcBId (uint8_t ** *pui8NfcBId*, uint8_t * *ui8NfcBIdLength*)

NFC_B_getNfcBId - This function returns the NFC-B UID (PICC or PCD Mode)

Parameters

pui8NfcBId	is a double pointer to be modified with the NFCBId buffer.
ui8NfcBIdLength	is a pointer to store the NFC ID Length.

This function returns the NFC-B UID.

Returns

None.

6.3.2.8 uint8_t NFC_B_getSENSBN (uint8_t * *pui8Payload*, uint8_t *ui8Length*)

NFC_B_getSENSBN - Returns the SENSB N from the SENSB request. (PICC mode)

Parameters

pui8Payload	is the pointer to the start of the SENSB REQ.
ui8Length	is the length of the SENSB REQ.

This function returns the N paramater from the SENSB Request.

Returns

ui8NValue is the N paramater from the SENSB Request. .

6.3.2.9 uint16_t NFC_B_getSFGT (void)

NFC_B_getSFGT - This function returns the g_ui16NfcBSfgt (PCD Mode)

This function returns the time (mS) the transceiver must wait before sending a command to the PICC.

Returns

g_ui16NfcBSfgt is the guard time the PCD must wait.

6.3.2.10 void NFC_B_init (void)

NFC_B_init - Initialize the NFC B driver.(PCD and PICC modes)

This function must be called prior to any other function offered by the NFC B driver. This function initializes the NFC DEP and ISO support for PICC and PCD, the length of the tag, driver current state, CID, and SAK response.

Returns

None.

6.3.2.11 bool NFC_B_isISOCompliant (void)

NFC_B_isISOCompliant - This function returns if the PICC is ISO compliant. (PCD Mode)

This function returns if the PICC is ISO compliant.

Returns

a boolean flag true if the the PICC is ISO compliant, otherwise false.

6.3.2.12 bool NFC_B_isSelected (void)

NFC_B_isSelected - This function returns if the PICC is activated. (PCD Mode)

This function returns if the PICC has been activated.

Returns

a boolean flag true if the the PICC is activated, otherwise false.

6.3.2.13 tNfcBPollState NFC_B_processReceivedRsp (uint8_t * *pui8RxBuffer,* uint8_t *ui8Length*)

NFC_B_processReceivedRsp - This function processes the response from the PICC. (PCD Mode)

Parameters

pui8RxBuffer	is a pointer to the recevied data.
ui8Length	is the length of received response.

This function processes the response to the SENSB REQ and ATTRIB REQ commands.

Returns

g_sNfcBPollState the current NFC-B state.

6.3.2.14 void NFC_B_send_ALLB_REQ (void)

NFC_B_send_ALLB_REQ - Sends the ALLB REQ to the PICC. (PCD mode)

This function sends an ALLB Request - first PCD command used to wake / activate NFC-B tags.

Returns

None.

6.3.2.15 void NFC_B_send_ATTRIB_REQ (void)

NFC_B_send_ATTRIB_REQ - Sends the ATTRIB REQ to the PICC. (PCD mode)

This function sends a ATTRIB Request to the PCD.

Returns

None.

6.3.2.16 void NFC_B_send_ATTRIB_RES (void)

NFC_B_send_ATTRIB_RES - Sends the ATTRIB RES to the PCD. (PICC Mode)

This function sends a ATTRIB Response to the PCD to acknowledge the ATTRIB REQ.

Returns

None.

6.3.2.17 void NFC_B_send_SENSB_REQ (void)

NFC_B_send_SENSB_REQ - Sends the SENSB REQ to the PICC. (PCD mode)

This function sends an SENSB Request - first PCD command used to activate NFC-B tags.

Returns

None.

6.3.2.18 void NFC_B_send_SENSB_RES (void)

NFC_B_send_SENSB_RES - Sends the SENSB RES to the PCD. (PICC Mode)

This function sends a SENSB Response to the PCD including the UID and NFC B capabilitiy information.

Returns

None.

6.3.2.19 void NFC_B_send_SLPB_REQ (void)

NFC_B_send_SLPB_REQ - Sends the SLPB REQ to the PICC. (PCD mode)

This function sends a SLPB Request to the PCD.

Returns

None.

6.3.2.20 void NFC_B_send_SLPB_RES (void)

NFC_B_send_SLPB_RES - Sends the SLPB RES to the PCD. (PICC Mode)

This function sends a SLPB Response to the PCD to acknowledge the SLPB REQ.

Returns

None.

6.3.2.21 uint8_t NFC_B_setNfcBId (uint8_t * *ui8NewNfcBId,* uint8_t *ui8Length*)

NFC_B_setNfcBId - This function sets the UID of the NFC-B when PICC mode is activated. (PICC Mode)

Parameters

ui8NewNfcAId	is the pointer to the NFCBId.
ui8Length	is the length of the UID. (Must be 4)

This function sets the ID and the size of the PICC UID.

Returns

ui8Status when the length of the UID is 4 bytes, else it returns STATUS_FAIL.

6.3.2.22 void NFC_B_stateMachine (void)

NFC_B_stateMachine - This function handles the activation of the PICC. (PCD Mode)

This function sends the SENSB Req, SLPB REQ and ATTRIB REQ.

Returns

None.

6.4 NFC Controller APIs Functions

Functions

- void NFC_init (void)
- void NFC_setListenTime (uint16_t ui16ListenTime)
- tNfcState NFC_run (void)
- tNfcState NFC_pollStateMachine (void)
- tNfcState NFC_listenStateMachine (void)
- void NFC_disable (void)
- void NFC_setSupportCertification (bool bSupportCertification)
- tTRF79x0_IRQFlag NFC_waitForCommand (uint16_t ui16TimeOut)

6.4.1 Detailed Description

NFC Controller APIs.

6.4.2 Function Documentation

6.4.2.1 void NFC_disable (void)

NFC_disable - disables the NFC stack.

This function disables the polling/listening statemachines.

Returns

None.

6.4.2.2 void NFC_init (void)

NFC_init - initializes the NFC controller variables.

This function intializes the NFC controller variables required for valid operation when using the NFC_run() function. This function must be executed before any of the other NFC controller functions.

Returns

None.

6.4.2.3 tNfcState NFC_listenStateMachine (void)

NFC_listenStateMachine - executes the listen statemachines.

This function is called from the NFC_run() function when the NFC stack enables the listening functionality. Initially this function will wait to receive a command from a reader for a 500 mS (default) or g_ui16LoopDelay which can be modified with the NFC_setListenTime() function. Once a reader sends a polling command for a technology we are waiting for, this function will reply with a valid response.

Returns

sReturnNfcState returns the current state of the NFC stack.

6.4.2.4 tNfcState NFC_pollStateMachine (void)

NFC_pollStateMachine - executes the polling statemachines.

This function is called from the NFC_run() function when the NFC stack enables the polling functionality. This function will try to activate different NFC technologies based on the polling configurations. Once a technology sends a successfull response, the NFC stack will activate that technology without continuing to scan for other technologies.

Returns

sReturnNfcState returns the current state of the NFC stack.

6.4.2.5 tNfcState NFC_run (void)

NFC_run - executes the polling and listening statemachines.

This function checks what modes were enabled for Card Emulation, Peer to Peer, and Reader / Writer to configure the supporting NFC mode variables. The supported NFC variables are used to decide which NFC technologies the NFC_listenStateMachine() function will listen for, and which NFC technologies the NFC_pollStateMachine() will try to activate. Once a NFC technology is activated, this function will continue to activate the necessary layers. Once the statemachine reaches the NFC_DATA_EXCHANGE_PROTOCOL state, the application can then execute commands based on NFC mode and technology that gets activated. The application must use the NFC_P2P_getModeStatus(), NFC_RW_getModeStatus(), and NFC_CE_getModeStatus() function to identify which technology is activated.

Returns

sReturnNfcState returns the current state of the NFC stack.

6.4.2.6 void NFC_setListenTime (uint16_t *ui16ListenTime*)

NFC_setListenTime - initializes the time the NFC stack will wait for an incoming PCD command.

Parameters

ui16ListenTime	the time the NFC_listenStateMachine() function will wait to receive an incoming reader command.

This function sets the time the NFC stack will wait for an incoming PCD command.The default time for the delay time is 500 mS.

Returns

None.

6.4.2.7 void NFC_setSupportCertification (bool *bSupportCertification*)

NFC_setSupportCertification - sets the NFC certification flag.

Parameters

bSupportCertification	is a boolean flag indicating NFC certification mode enabled.

This function sets the value for g_bSupportCertification (NFC Certifcation mode enabled). This function must be executed when performing tests with the NFC Wave 1 Certification Test Suite.For normal NFC operation this function must be called with bSupportCertification equal to false.

Returns

None.

6.4.2.8 tTRF79x0_IRQFlag NFC_waitForCommand (uint16_t *ui16TimeOut*)

NFC_waitForCommand - this function waits to receive a command from the PICC / PCD.

Parameters

ui16TimeOut	is the time the function will wait to receive a command.

This function waits to receive a command from the PICC / PCD. If there is a protocol error, packet overflow, SDD complete event, or a valid packet is received the function will return to the NFC_run() function.

Returns

eTrfIrqStatus is the transceiver flag indicating the communication status.

6.5 NFC P2P APIs Functions

Functions

- bool NFC_P2P_getModeStatus (t_sNfcP2PMode ∗psP2PMode, t_sNfcP2PCommBitrate ∗psP2PBitrate)
- tNfcP2PRxStatus NFC_P2P_getReceiveState (void)
- uint8_t NFC_P2P_sendNdefPacket (uint8_t ∗pui8Ndef, bool bPacketStart, uint8_t ui8FragmentLength, uint32_t ui32PacketSize)

6.5.1 Detailed Description

NFC Peer to Peer APIs.

6.5.2 Function Documentation

6.5.2.1 bool NFC_P2P_getModeStatus (t_sNfcP2PMode ∗ *psP2PMode,* t_sNfcP2PCommBitrate ∗ *psP2PBitrate*)

NFC_P2P_getModeStatus - checks if the P2P mode has been activated.

Parameters

psP2PMode	is the pointer where the current P2P mode is stored if P2P is actived.
psP2PBitrate	is the pointer where the current P2P bitrate is stored if P2P is actived.

This function checks if P2P mode is activated. If it is, it stores the current P2P mode and bitrates in the respectives variables.

Returns

bP2PEnableStatus is a boolean indicating if P2P is activated.

6.5.2.2 tNfcP2PRxStatus NFC_P2P_getReceiveState (void)

NFC_P2P_getReceiveState - returns current SNEP receive statet.

This function returns the current SNEP receive state when a SNEP connection has been established over P2P.

Returns

sReceiveStatus the current SNEP receive status.

6.5.2.3 uint8_t NFC_P2P_sendNdefPacket (uint8_t ∗ *pui8Ndef,* bool *bPacketStart,* uint8_t *ui8FragmentLength,* uint32_t *ui32PacketSize*)

NFC_P2P_sendNdefPacket - this function sends an NDEF packet using the Simple NDEF Exchange Protocol layer.

Parameters

pui8Ndef	is the pointer to the current fragment to be transmitted.
bPacketStart	is the boolean flag indicating the first fragment of a packet.
ui8FragmentLength	is the fragment length
ui32PacketSize	is the complete packet size

This function checks if there is available space in the SNEP buffer, and queues the data to be sent.

Returns

ui8TransmittedBytes the number of bytes queued in the SNEP layer.

6.6 NFC RW APIs Functions

Functions

- bool NFC_RW_getModeStatus (t_sNfcRWMode ∗psRWMode, t_sNfcRWCommBitrate ∗psRWBitrate)
- t_sNfcRWCommBitrate NFC_RW_getRWSuportedBitrates (void)
- void NFC_RW_triggerRWProtocolError (void)
- void NFC_RW_enableMifareMode (void)

6.6.1 Detailed Description

NFC Reader/Writer APIs.

6.6.2 Function Documentation

6.6.2.1 void NFC_RW_enableMifareMode (void)

NFC_RW_enableMifareMode - Enables the Mifare Data Exchange Protocol State.

This function switches the current initiator state to NFC_MIFARE_DATA_EXCHANGE_PROTOCOL. This is required for Mifare reading and writing functionality.

Returns

None.

6.6.2.2 bool NFC_RW_getModeStatus (t_sNfcRWMode ∗ *psRWMode,* t_sNfcRWCommBitrate ∗ *psRWBitrate*)

NFC_RW_getModeStatus - checks if the RW mode has been activated.

Parameters

psRWMode	is the pointer where the current RW mode is stored if RW is actived.
psRWBitrate	is the pointer where the current RW bitrate is stored if RW is actived.

This function checks if RW mode is activated. If it is, it stores the current RW mode and bitrates in the respectives variables.

Returns

bRwEnableStatus is a boolean indicating if RW is activated.

6.6.2.3 t_sNfcRWCommBitrate NFC_RW_getRWSuportedBitrates (void)

NFC_RW_getRWSuportedBitrates - returns the supported initiator NFC RW bitrates.

This function returns the supported RW bitrates.

Returns

sRWCommSupport the supported RW bitrates.

6.6.2.4 void NFC_RW_triggerRWProtocolError (void)

NFC_RW_triggerRWProtocolError - disables the RF field and resets the polling statemachine.

This function triggers a protocol error disabling the RF field. It should be used when the NFC stack is in polling mode.

Returns

None.

6.7 NFC CE APIs Functions

Functions

- bool NFC_CE_getModeStatus (t_sNfcCEMode ∗psCEMode)

6.7.1 Detailed Description

NFC Card Emulation APIs.

6.7.2 Function Documentation

6.7.2.1 bool NFC_CE_getModeStatus (t_sNfcCEMode ∗ *psCEMode*)

NFC_CE_getModeStatus - checks if the CE mode has been activated.

Parameters

psCEMode	is the pointer where the current CE mode is stored if CE is actived.

This function checks if CE mode is activated. If it is, it stores the current CE mode in psCEMode.

Returns

bCEEnableStatus is a boolean indicating if CE is activated.

6.8 NFC DEP API Functions

Functions

- void NFCDEP_configure_P2P (t_sNfcDEP_P2PSetup sP2PSetupOptions, bool bTestEnable, uint8_t ui8DIDi)
- void NFCDEP_init (void)
- void NFCDEP_sendATR_REQ (void)
- void NFCDEP_sendDEP_REQ (void)
- void NFCDEP_sendPSL_REQ (void)
- void NFCDEP_sendRSL_REQ (void)
- void NFCDEP_sendDSL_REQ (void)
- uint8_t NFCDEP_sendATR_RES (void)
- void NFCDEP_sendDEP_RES (void)
- void NFCDEP_sendPSL_RES (void)
- void NFCDEP_sendRSL_RES (void)
- void NFCDEP_sendDSL_RES (void)
- uint16_t NFCDEP_getDEPCommand (uint8_t *pui8RxPayload, uint8_t ui8CmdLength)
- void NFCDEP_stateMachine (void)
- uint8_t NFCDEP_processReceivedRequest (uint8_t *pui8RxPayload, uint8_t ui8CmdLength)
- tDepState NFCDEP_processReceivedData (uint8_t *pui8RxBuffer, uint8_t ui8Length)
- t_sTRF79x0_Bitrate NFCDEP_getBitrate (void)
- uint16_t NFCDEP_getLinkTimeOut (void)
- void NFCDEP_setNFCID3t (uint8_t *pui8NFCID3, uint8_t ui8Length)
- void NFCDEP_setNFCID2i (uint8_t *pui8NFCID2, uint8_t ui8Length)
- uint8_t NFCDEP_sendSOT (uint8_t ui8Offset)
- bool NFCDEP_checkEOT (uint8_t *pui8RxBuffer, uint8_t ui8Length)
- uint8_t NFCDEP_triggerTimeout (void)
- uint8_t NFCDEP_triggerProtocolError (void)

6.8.1 Detailed Description

NFC Data Exchange Protocol Statemachine based on the NFC Forum Digital Protocol ver 1.1.

6.8.2 Function Documentation

6.8.2.1 bool NFCDEP_checkEOT (uint8_t * *pui8RxBuffer,* uint8_t *ui8Length*)

NFCDEP_checkEOT - This function initializaes checks if the received packet is the End of Test Frame.

Parameters

pui8RxBuffer	is the pointer to the received buffer.
ui8Length	is the length of the received packet.

This function checks if the received packet is the End of Test Frame indicating that the NFC DEP test has been completed.

Returns

bReceiveEOT indicates if the EOT frame has been received.

6.8.2.2 void NFCDEP_configure_P2P (t_sNfcDEP_P2PSetup *sP2PSetupOptions,* bool *bTestEnable,* uint8_t *ui8DIDi*)

NFCDEP_configure_P2P - Configures the NFC DEP layer for certifcation purposes.

Parameters

sP2PSetupOptions	holds the configuration options required for certifcation. Including Loopback and enable/disable the LLCP layer.
bTestEnable	boolean to enable/disable certfication testing
ui8DIDi	the DID for initiator.

This function must be called in the NFC Configuration function.

Returns

None.

6.8.2.3 t_sTRF79x0_Bitrate NFCDEP_getBitrate (void)

NFCDEP_getBitrate - This function returns the current connection bitrate.

This function returns the current initiator or target bitrate.

Returns

sPayloadFrequency the current communication bitrate.

6.8.2.4 uint16_t NFCDEP_getDEPCommand (uint8_t * *pui8RxPayload,* uint8_t *ui8CmdLength*)

NFCDEP_getDEPCommand - This function provides the NFCDEP command

Parameters

pui8RxPayload	is the pointer to the received NFC DEP packet.
ui8CmdLength	is the length of the received NFC DEP packet.

This function returns the NFC DEP command code if the length of the NFC DEP Packet is greater than 2 bytes.

Returns

ui16DEPCmd is the NFC DEP command code.

6.8.2.5 uint16_t NFCDEP_getLinkTimeOut (void)

NFCDEP_getLinkTimeOut - This function returns the current connection timeout.

This function returns the current connection timeout. If there is a RTOX request then it returns the RTOXTimeout.

Returns

g_ui16NfcDepTimeOut the current NFC DEP timeout.

6.8.2.6 void NFCDEP_init (void)

NFCDEP_init - Initializes the variables for the NFC DEP layer.

This function must be called prior to any other NFC DEP function in in order to properly initialize all variables which have been declared. Failure to do so may cause the software to enter unknown states. It will initialize the LLCP layer.

Returns

None.

6.8.2.7 tDepState NFCDEP_processReceivedData (uint8_t * *pui8RxBuffer,* uint8_t *ui8Length*)

NFCDEP_processReceivedData - This function executes when the transceiver is in initiator mode.

Parameters

pui8RxPayload	is a pointer to the received payload.
ui8CmdLength	is the length of the received payload.

This function parses the incoming response from the target. If the response is valid, the function sets the g_eCurrentDepState state.

Returns

g_eCurrentDepState the current DEP state.

6.8.2.8 uint8_t NFCDEP_processReceivedRequest (uint8_t * *pui8RxPayload,* uint8_t *ui8CmdLength*)

NFCDEP_processReceivedRequest - This function executes when the transceiver is in target mode.

Parameters

pui8RxPayload	is a pointer to the received payload.
ui8CmdLength	is the length of the received payload.

This function parses the incoming requests from the initiator. If the request is valid, the function replies with the corresponding cmd response.

Returns

ui8NfcDepStatus whether the request command is valid or not.

6.8.2.9 void NFCDEP_sendATR_REQ (void)

NFCDEP_sendATR_REQ - Sends ATR Request.

This function sends ATR Request.

Returns

None.

6.8.2.10 uint8_t NFCDEP_sendATR_RES (void)

NFCDEP_sendATR_RES - Sends ATR Response.

This function sends ATR Response.

Returns

STATUS_SUCCESS(0x01) if packet is sent succesfully else STATUS_FAIL(0x00).

6.8.2.11 void NFCDEP_sendDEP_REQ (void)

NFCDEP_sendDEP_REQ - Sends DEP Request.

This function sends DEP Request.

Returns

None.

6.8.2.12 void NFCDEP_sendDEP_RES (void)

NFCDEP_sendDEP_RES - Sends DEP Response.

This function sends DEP Response.

Returns

STATUS_SUCCESS(0x01) if packet is sent succesfully else STATUS_FAIL(0x00).

6.8.2.13 void NFCDEP_sendDSL_REQ (void)

NFCDEP_sendDSL_REQ - Sends DSL Request.

This function sends DSL Request.

Returns

None.

6.8.2.14 void NFCDEP_sendDSL_RES (void)

NFCDEP_sendDSL_RES - Sends DSL Response.

This function sends DSL Response.

Returns

STATUS_SUCCESS(0x01) if packet is sent succesfully else STATUS_FAIL(0x00).

6.8.2.15 void NFCDEP_sendPSL_REQ (void)

NFCDEP_sendPSL_REQ - Sends PSL Request.

This function sends PSL Request.

Returns

None.

6.8.2.16 void NFCDEP_sendPSL_RES (void)

NFCDEP_sendPSL_RES - Sends PSL Response.

This function sends PSL Response.

Returns

STATUS_SUCCESS(0x01) if packet is sent succesfully else STATUS_FAIL(0x00).

6.8.2.17 void NFCDEP_sendRSL_REQ (void)

NFCDEP_sendRSL_REQ - Sends RSL Request.

This function sends RSL Request.

Returns

None.

6.8.2.18 void NFCDEP_sendRSL_RES (void)

NFCDEP_sendRSL_RES - Sends RSL Response.

This function sends RSL Response.

Returns

STATUS_SUCCESS(0x01) if packet is sent succesfully else STATUS_FAIL(0x00).

6.8.2.19 uint8_t NFCDEP_sendSOT (uint8_t *ui8Offset*)

NFCDEP_sendSOT - This function sends the Start Of Test Frame for NFC Wave 1 Certification.

Parameters

ui8Offset	is the offset on the current TX packet to write the SOT.

This function handles sending the Start of Test frames for Wave 1 NFC Forum Certification

Returns

ui8Offset the length of the total SOT packet inluding the header overhead and the SOT Test frame.

6.8.2.20 void NFCDEP_setNFCID2i (uint8_t * *pui8NFCID2,* uint8_t *ui8Length*)

NFCDEP_setNFCID2i - This function initializes the NFCID2i buffer.

Parameters

pui8NFCID2	is the pointer to the NFCID2 payload.
ui8Length	is the length of the NFCID2 buffer.

This function sets the NFCID3t buffer. The length must be 8 bytes.

Returns

None.

6.8.2.21 void NFCDEP_setNFCID3t (uint8_t * *pui8NFCID3,* uint8_t *ui8Length*)

NFCDEP_setNFCID3t - This function initializes the NFCID3t buffer.

Parameters

pui8NFCID3	is the pointer to the NFCID3 payload.
ui8Length	is the length of the NFCID3 buffer.

This function sets the NFCID3t buffer. The length must be 10 bytes.

Returns

None.

6.8.2.22 void NFCDEP_stateMachine (void)

NFCDEP_stateMachine - This function executes when the transceiver is in initiator mode.

This function executes the main statemachine when the transceiver is in initiator mode.

Returns

None.

6.8.2.23 uint8_t NFCDEP_triggerProtocolError (void)

NFCDEP_triggerTimeout - This function is used to trigger a protocol error event to the NFC DEP layer.

This function is used in order to keep track of how many protocol errors have occured. A maximum number of protocol errors will be entered during initialization.When the maximum number of protocol errors is reached, the software will reset the part. Otherwise, it will send an NACK PDU.

Returns

ui8Status indicates STATUS_SUCCESS if the protocol error variable has not reached the max number of protocol errors, else it returns STATUS_FAIL.

6.8.2.24 uint8_t NFCDEP_triggerTimeout (void)

NFCDEP_triggerTimeout - This function is used to trigger a timeout event to the NFC DEP layer.

This function is used in order to keep track of how many sequential timeouts have occured. A maximum number of timeouts will be entered during initialization.When the maximum number of timeouts is reached, the software will reset the part. Otherwise, it will send an ATN PDU.

Returns

ui8Status indicates STATUS_SUCCESS if the timeout has not reached the max timeout value, else it returns STATUS_FAIL.

6.9 NFC F API Functions

Functions

- void NFC_F_Init (void)
- void NFC_F_send_SENSF_REQ (uint16_t ui16SC, uint8_t ui8RC, uint8_t ui8Slot)
- void NFC_F_send_SENSF_RES (uint8_t ui8RDValue)
- uint8_t NFC_F_processReceivedRequest (uint8_t *pui8RxBuffer, uint8_t ui8Length)
- void NFC_F_getNFCID2 (uint8_t **pui8NfcFId, uint8_t *ui8NfcFIdLength)
- void NFC_F_setNfcDepSupport (bool bDepSupport)
- uint8_t NFC_F_isNfcDepSupported (void)
- uint8_t NFC_F_setNfcId2 (uint8_t *ui8NewNfcId2, uint8_t ui8Length)

6.9.1 Detailed Description

NFC F is a NFC Layer used by NFC_Initiator and NFC_Target layers to activate the NFC-F technology. For more information on NFC-F please read the NFC Digital Specification Version 1.1.

6.9.2 Function Documentation

6.9.2.1 void NFC_F_getNFCID2 (uint8_t ** *pui8NfcFId*, uint8_t * *ui8NfcFIdLength*)

NFC_F_getNFCID2 - This function returns the NFCID2 ID (PICC or PCD Mode)

Parameters

pui8NfcFId	is a double pointer to be modified with the NFCID2 buffer.
ui8NfcFIdLength	is a pointer to store the NFCID2 Length.

This function returns the NFCID2.

Returns

None.

6.9.2.2 void NFC_F_Init (void)

NFC_F_Init - Initialize the NFC F driver.

This function must be called prior to any other function offered by the NFC F driver. This function initializes the NFC DEP support for PICC and PCD,

Returns

None.

6.9.2.3 uint8_t NFC_F_isNfcDepSupported (void)

NFC_F_isNfcDepSupported - Checks if NFC DEP is supported.

This function checks if NFC DEP is supported.

Returns

ui8RetStatus returns STATUS_SUCCESS if it is supported, otherwise it returns STATUS_FAIL.

6.9.2.4 uint8_t NFC_F_processReceivedRequest (uint8_t * *pui8RxBuffer,* uint8_t *ui8Length*)

NFC_F_processReceivedRequest - Processes the received SENSF_REQ or SENSFRES.

Parameters

pui8RxBuffer	is the pointer to the received data.
ui8Length	is the length of the received packet.

This function checks if the command received is a SENSF_REQ or SENSF_RES. If it is a SENSF_RES, it stores the NFFCID2. If the command is a SENSF_REQ, it replies according the request code it received.

Returns

ui8Status returns STATUS_SUCCESS if the packet is valid, otherwise it returns STATUS_FAIL.

6.9.2.5 void NFC_F_send_SENSF_REQ (uint16_t *ui16SC,* uint8_t *ui8RC,* uint8_t *ui8Slot*)

NFC_F_send_SENSF_REQ - Sends the SENSF REQ to the PICC.

Parameters

ui16SC	is the system code.
ui8RC	is the request code.
ui8Slot	is the number of slots.

This function sends a SENSF REQ to the PICC.

Returns

None.

6.9.2.6 void NFC_F_send_SENSF_RES (uint8_t *ui8RDValue*)

NFC_F_send_SENSF_RES - Sends the SENSF RES to the PCD.

Parameters

ui8RDValue	is based on the request SENS_REQ.

This function sends a SENS Response to the PCD including the NFCID2.

Returns

None.

6.9.2.7 void NFC_F_setNfcDepSupport (bool *bDepSupport*)

NFC_F_setNfcDepSupport - Sets the NFCDEP support for the NFC-F driver.

Parameters

bDepSupport	the NFC DEP support flag.

This function sets the NFC DEP support.

Returns

None.

6.9.2.8 uint8_t NFC_F_setNfcId2 (uint8_t * *ui8NewNfcId2,* uint8_t *ui8Length*)

NFC_F_setNfcId2 - This function sets the UID of the NFC-F when PICC mode is activated. (PICC Mode)

Parameters

ui8NewNfcId2	is the pointer to the NFCID2.
ui8Length	is the length of the UID. (Must be 8)

This function sets the ID and the size of the PICC UID.

Returns

ui8Status returns STATUS_SUCCESS if the command is valid, otherwise it returns STATUS_FAIL.

6.10 NFC Initiator API Functions

Functions

- void NFC_Initiator_Init (void)
- uint16_t NFC_Initiator_GetPICCTimeOut (void)
- void NFC_Initiator_SetState (tNfcInitiatorState eState)
- tNfcInitiatorState NFC_Initiator_StateMachine (t_sTRF79x0_InitiatorMode sInitiatorMode)
- tNfcInitiatorState NFC_Initiator_ProccessResponse (uint8_t *pui8Payload, uint8_t ui8Length, t_sTRF79x0_InitiatorMode sInitiatorMode)
- tNfcInitiatorState NFC_Initiator_GetState (void)

Variables

- tNfcInitiatorState **g_sCurrentPollState**
- uint16_t **g_ui16PICCTimeOut**

6.10.1 Detailed Description

NFC Initiator Statemachine based on the NFC Forum Activity Spec Ver 1.0.

6.10.2 Function Documentation

6.10.2.1 uint16_t NFC_Initiator_GetPICCTimeOut (void)

NFC_Initiator_GetPICCTimeOut - returns current PICC timeout.

This function returns the current PICC timeout.

Returns

g_ui16PCDTimeOut the current PICC timeout.

6.10.2.2 tNfcInitiatorState NFC_Initiator_GetState (void)

NFC_Initiator_GetState - returns current initator state.

This function returns the current initiator state.

Returns

g_sCurrentPollState the current initiator state.

6.10.2.3 void NFC_Initiator_Init (void)

NFC_Initiator_Init - Initializes the variables for the initiator statemachine.

This function must be called prior to any other NFC_Initiator function in in order to properly initialize all variables which have been declared. Failure to do so may cause the software to enter unknown states. It will initialize the NFC_A, NFC_B, NFC_F, T2T, T3T, T5T, Mifare, NFC_DEP, and ISO_DEP layers.

Returns

None.

6.10.2.4 tNfcInitiatorState NFC_Initiator_ProccessResponse (uint8_t * *pui8Payload,* uint8_t *ui8Length,* t_sTRF79x0_InitiatorMode *sInitiatorMode*)

NFC_Initiator_ProccessResponse - Handles incoming commands from PICC, and sets the PICC timeout based on the NFC technologies that get activated.

Parameters

pui8Payload	is the pointer to data received from the PICC.
ui8Length	is the number of bytes received from the PICC.
sInitiatorMode	is the initiator mode enabled in the transceiver.

This function processes the response from PICC and sets the state for the next command to be sent from the PCD in the NFC_Initiator_StateMachine() function.

Returns

g_sCurrentPollState the current initiator state.

6.10.2.5 void NFC_Initiator_SetState (tNfcInitiatorState *eState*)

NFC_Initiator_SetState - sets the current initiator state.

Parameters

eState	the initiator state

This function sets g_eCurrentTargetState with a new initiator state (eState), used inside the NFC_Initiator_StateMachine() function.

Returns

None.

6.10.2.6 tNfcInitiatorState NFC_Initiator_StateMachine (t_sTRF79x0_InitiatorMode *sInitiatorMode*)

NFC_Initiator_StateMachine - Handles incoming commands from PCD, and sets the PCD timeout based on the NFC technologies that get activated.

Parameters

sInitiatorMode	is the initiator mode enabled in the transceiver.

This function handles the initiator statemachine for the application where the transceiver is in polling mode. It handles the anti-collision / activation of NFC-A, NFC-B and NFC-F. Once the technologies are activated it calls the NFCDEP_stateMachine() function for P2P mode, NFC_RW_T2T_stateMachine() function for T2T tags, NFC_RW_T3T_stateMachine() function for T3T tags, ISODEP_stateMachine() function for T4T tags, MIFARE_stateMachine() function for Mifare Classic tags, and NFC_RW_T5T_stateMachine() function T5T tags.

Returns

g_sCurrentPollState the current initiator mode.

6.11 NFC Type 2 Reader API Functions

Functions

- void NFC_RW_T2T_init (void)
- void NFC_RW_T2T_stateMachine (void)
- tNfcRwT2TConnectionStatus NFC_RW_T2T_processReceivedData (uint8_t *pui8RxBuffer, uint8_t ui8Length)
- tNfcRwT2TConnectionStatus NFC_RW_T2T_getWriteStatus (void)
- tNfcRwT2TConnectionStatus NFC_RW_T2T_getReadStatus (void)
- tNfcRwT2TConnectionStatus NFC_RW_T2T_getSectorSelStatus (void)
- uint8_t NFC_RW_T2T_sendReadCmd (uint8_t ui8BlockNumber)
- uint8_t NFC_RW_T2T_sendWriteCmd (uint8_t ui8BlockNumber, const uint8_t *pui8Buffer, uint8_t ui8Length)
- uint8_t NFC_RW_T2T_sendSectorSel (uint8_t ui8SectorNumber)
- uint8_t NFC_RW_T2T_sendSleep (void)
- void NFC_RW_T2T_getPacketStatus (uint8_t **pui8RXData, uint8_t *pui8Length)

6.11.1 Detailed Description

A Type 2 Reader/Writer issues commands based on the ISO14443-3 specifications.

For more information on Type 2 Readers and Tags please read the NFC Forum Type 2 Specifications.

6.11.2 Function Documentation

6.11.2.1 void NFC_RW_T2T_getPacketStatus (uint8_t ** *pui8RXData*, uint8_t * *pui8Length*)

NFC_RW_T2T_getPacketStatus - Gets the last Type 2 data packet received.

Parameters

pui8RXData	this is a double pointer to receive the current pointer to the location of the Type 2 data bytes received buffer
pui8Length	this is a pointer to get the length of the received Type 2 data.

This function returns the packet length and the location of the packet contents of the latest Type 2 data packet that was received.

Returns

None.

6.11.2.2 tNfcRwT2TConnectionStatus NFC_RW_T2T_getReadStatus (void)

NFC_RW_T2T_getReadStatus - Checks for a Read response.

This function checks if a Read response was received, and returns the Type 2 Reader status of either a SUCCESS or a FAIL. If a Read response has been received, the current status will be changed to IDLE regardless of whether the Type 2 Reader status was a SUCCESS or FAIL.

Returns

eStatusTemp is the Type 2 Reader status.

6.11.2.3 tNfcRwT2TConnectionStatus NFC_RW_T2T_getSectorSelStatus (void)

NFC_RW_T2T_getSectorSelStatus - Checks for a Sector Sel response.

This function checks if a Sector Sel response was received, and returns the Type 2 Reader status of either a SUCCESS or a FAIL. If a Sector Sel response has been received, the current status will be changed to IDLE regardless of whether the Type 2 Reader status was a SUCCESS or FAIL.

Returns

eStatusTemp is the Type 2 Reader status.

6.11.2.4 tNfcRwT2TConnectionStatus NFC_RW_T2T_getWriteStatus (void)

NFC_RW_T2T_getWriteStatus - Checks for a Write response.

This function checks if a Write response was received, and returns the Type 2 Reader status of either a SUCCESS or a FAIL. If a Write response has been received, the current status will be changed to IDLE regardless of whether the Type 2 Reader status was a SUCCESS or FAIL.

Returns

eStatusTemp is the Type 2 Reader status.

6.11.2.5 void NFC_RW_T2T_init (void)

NFC_RW_T2T_init - Initializes the variables for the Type 2 Reader/Writer.

This function must be called prior to any other NFC_RW_T2T function in in order to properly initialize all variables which have been declared. Failure to do so may cause the software to enter unknown states.

Returns

None.

6.11.2.6 tNfcRwT2TConnectionStatus NFC_RW_T2T_processReceivedData (uint8_t * *pui8RxBuffer,* uint8_t *ui8Length*)

NFC_RW_T2T_processReceivedData - Processes the Type 2 Data Received.

Parameters

pui8RxBuffer	is the pointer to where T2T data can be received.
ui8Length	is the length of the T2T data packet received.

This function is used to handle the processing of all Type 2 data packets that have been received from Type 3 tags. This function includes a timeout monitor to ensure the software will not be locked up if the tag is unexpectedly removed from the RF field.

This function will wait for replies specific to the command that was issued. If it receives an expected reply then it will indicated a successful response and the application layer would then have to determine what command to issue next. This gives developers full control over the sequencing of Type 2 Reader commands. If an unexpected reply is received, it will be treated as a protocol error and the Initiator state machine will reset.

Returns

g_eNfcRwT2TCommStatus returns the current status of the Type 2 Reader based on what response was received.

6.11.2.7 uint8_t NFC_RW_T2T_sendReadCmd (uint8_t *ui8BlockNumber*)

NFC_RW_T2T_sendReadCmd - Sends the Read Command.

Parameters

ui8BlockNumber	is the block number to be read.

This function is used set the correct state for sending a Update command, and also to process any application based variables that have been passed into the Type 2 Reader layer.

Returns

ui8Status indicates if the state was successfully changed or not.

6.11.2.8 uint8_t NFC_RW_T2T_sendSectorSel (uint8_t *ui8SectorNumber*)

NFC_RW_T2T_sendSectorSel - Sends the Sector Sel Command.

Parameters

ui8SectorNumber	is the sector number to select.

This function is used set the correct state for sending a Sector Select command, and also to process any application based variables that have been passed into the Type 2 Reader layer.

Returns

ui8Status indicates if the state was successfully changed or not.

6.11.2.9 uint8_t NFC_RW_T2T_sendSleep (void)

NFC_RW_T2T_sendSleep - Sends the Sleep Command.

This function is used set the correct state for sending a Sleep command, and also to process any application based variables that have been passed into the Type 2 Reader layer.

Returns

ui8Status indicates if the state was successfully changed or not.

6.11.2.10 uint8_t NFC_RW_T2T_sendWriteCmd (uint8_t *ui8BlockNumber,* const uint8_t * *pui8Buffer,* uint8_t *ui8Length*)

NFC_RW_T2T_sendWriteCmd - Sends the Write Command.

Parameters

ui8BlockNumber	is the block number to be written.
pui8Buffer	is a pointer to the data to be written
ui8Length	is the length of the packet to be written.

This function is used set the correct state for sending a Write command, and also to process any application based variables that have been passed into the Type 2 Reader layer.

Returns

ui8Status indicates if the state was successfully changed or not.

6.11.2.11 void NFC_RW_T2T_stateMachine (void)

NFC_RW_T2T_stateMachine - Conditions and transmits the data packets for Type 2 Reader commands.

This function constructs the data packets for Type 2 commands based on the current status. The packet is then transmitted at the end of the state machine. Currently the supported Type 2 commands are: Read, Sector Select, and Write.

Returns

None.

6.12 NFC Type 3 Reader API Functions

Functions

- void NFC_RW_T3T_init (void)
- void NFC_RW_T3T_stateMachine (void)
- tNfcRwT3TConnectionStatus NFC_RW_T3T_processReceivedData (uint8_t *pui8RxBuffer, uint8_t ui8Length)
- tNfcRwT3TConnectionStatus NFC_RW_T3T_getUpdateStatus (void)
- tNfcRwT3TConnectionStatus NFC_RW_T3T_getCheckStatus (void)
- tNfcRwT3TConnectionStatus NFC_RW_T3T_getSensFStatus (void)
- uint8_t NFC_RW_T3T_sendCheckCmd (tT3TPacketData sT3TData)
- uint8_t NFC_RW_T3T_sendSensFReq (uint16_t ui16SC, uint8_t ui8RC, uint8_t ui8Slot)
- uint8_t NFC_RW_T3T_sendUpdateCmd (tT3TPacketData sT3TData)
- void NFC_RW_T3T_getPacketStatus (uint8_t **pui8RXData, uint8_t *pui8Length)

6.12.1 Detailed Description

A Type 3 Reader/Writer issues commands based on the FeliCa specifications.

For more information on Type 3 Readers and Tags please read the FeliCa Specifications.

6.12.2 Function Documentation

6.12.2.1 tNfcRwT3TConnectionStatus NFC_RW_T3T_getCheckStatus (void)

NFC_RW_T3T_getCheckStatus - Checks for a check response.

This function checks if a Check response was received, and returns the Type 3 Reader status of either a SUCCESS or a FAIL. If a Check response has been received, the current status will be changed to IDLE regardless of whether the Type 3 Reader status was a SUCCESS or FAIL.

Returns

eStatusTemp is the Type 3 Reader status.

6.12.2.2 void NFC_RW_T3T_getPacketStatus (uint8_t ** *pui8RXData*, uint8_t * *pui8Length*)

NFC_RW_T3T_getPacketStatus - Gets the last Type 3 data packet received.

Parameters

pui8RXData	this is a double pointer to receive the current pointer to the location of the Type 3 data bytes received buffer
pui8Length	this is a pointer to get the length of the received Type 3 data.

This function returns the packet length and the location of the packet contents of the latest Type 3 data packet that was received.

Returns

None.

6.12.2.3 tNfcRwT3TConnectionStatus NFC_RW_T3T_getSensFStatus (void)

NFC_RW_T3T_getSensFStatus - Checks for a SenF response.

This function checks if a SENSF response was received, and returns the Type 3 Reader status of either a SUCCESS or a FAIL. If a SENSF response has been received, the current status will be changed to IDLE regardless of whether the Type 3 Reader status was a SUCCESS or FAIL.

Returns

eStatusTemp is the Type 3 Reader status.

6.12.2.4 tNfcRwT3TConnectionStatus NFC_RW_T3T_getUpdateStatus (void)

NFC_RW_T3T_getUpdateStatus - Checks for a Update response.

This function checks if a Update response was received, and returns the Type 3 Reader status of either a SUCCESS or a FAIL. If a Read Single response has been received, the current status will be changed to IDLE regardless of whether the Type 3 Reader status was a SUCCESS or FAIL.

Returns

eStatusTemp is the Type 3 Reader status.

6.12.2.5 void NFC_RW_T3T_init (void)

NFC_RW_T3T_init - Initializes the variables for the Type 3 Reader/Writer.

This function must be called prior to any other NFC_RW_T3T function in in order to properly initialize all variables which have been declared. Failure to do so may cause the software to enter unknown states.

Returns

None

6.12.2.6 tNfcRwT3TConnectionStatus NFC_RW_T3T_processReceivedData (uint8_t * *pui8RxBuffer,* uint8_t *ui8Length*)

NFC_RW_T3T_processReceivedData - Processes the Type 3 Data Received.

Parameters

pui8RxBuffer	is the pointer to where T3T data can be received.
ui8Length	is the length of the T3T data packet received.

This function is used to handle the processing of all Type 3 data packets that have been received from Type 3 tags. This function includes a timeout monitor to ensure the software will not be locked up if the tag is unexpectedly removed from the RF field.

This function will wait for replies specific to the command that was issued. If it receives an expected reply then it will indicated a successful response and the application layer would then have to determine what command to issue next. This gives developers full control over the sequencing of Type 3 Reader commands. If an unexpected reply is received, it will be treated as a protocol error and the Initiator state machine will reset.

Returns

g_eNfcRwT3TCommStatus returns the current status of the Type 3 Reader based on what response was received.

6.12.2.7 uint8_t NFC_RW_T3T_sendCheckCmd (tT3TPacketData *sT3TData*)

NFC_RW_T3T_sendCheckCmd - Sends the Check Command.

Parameters

sT3TData	is the Check command to be sent with the Type 3 command

This function is used set the correct state for sending a Check command, and also to process any application based variables that have been passed into the Type 3 Reader layer.

Returns

ui8Status indicates if the state was successfully changed or not.

6.12.2.8 uint8_t NFC_RW_T3T_sendSensFReq (uint16_t *ui16SC,* uint8_t *ui8RC,* uint8_t *ui8Slot*)

NFC_RW_T3T_sendSensFReq - Sends the SENSF Req Command.

Parameters

ui16SC	is the System Code of the SENSF Req.
ui8RC	is the Request Code of the SENSF Req.
ui8Slot	is the Time Slot Number of the SENSF Req.

This function is used set the correct state for sending a SENSF Req. command, and also to process any application based variables that have been passed into the Type 3 Reader layer.

Returns

ui8Status indicates if the state was successfully changed or not.

6.12.2.9 uint8_t NFC_RW_T3T_sendUpdateCmd (tT3TPacketData *sT3TData*)

NFC_RW_T3T_sendUpdateCmd - Sends the Update Command.

Parameters

sT3TData	is the Update command to be sent with the Type 3 command

This function is used set the correct state for sending a Update command, and also to process any application based variables that have been passed into the Type 3 Reader layer.

Returns

ui8Status indicates if the state was successfully changed or not.

6.12.2.10 void NFC_RW_T3T_stateMachine (void)

NFC_RW_T3T_stateMachine - Conditions and transmits the data packets for Type 3 Reader commands.

This function constructs the data packets for Type 3 commands based on the current status. The packet is then transmitted at the end of the state machine. Currently the supported Type 3 commands are: Check, Update, and SENSF_REQ.

Returns

None.

6.13 NFC Type V Reader API Functions

Functions

- void NFC_RW_T5T_init (void)
- void NFC_RW_T5T_stateMachine (void)
- tNfcRwT5TConnectionStatus NFC_RW_T5T_processReceivedData (uint8_t *pui8RxBuffer, uint8_t ui8Length)
- tNfcRwT5TConnectionStatus NFC_RW_T5T_getInventoryStatus (void)
- tNfcRwT5TConnectionStatus NFC_RW_T5T_getReadSingleStatus (void)
- tNfcRwT5TConnectionStatus NFC_RW_T5T_getWriteSingleStatus (void)
- tNfcRwT5TConnectionStatus NFC_RW_T5T_getLockBlockStatus (void)
- tNfcRwT5TConnectionStatus NFC_RW_T5T_getSelectStatus (void)
- tNfcRwT5TConnectionStatus NFC_RW_T5T_getResetToReadyStatus (void)
- tNfcRwT5TConnectionStatus NFC_RW_T5T_getReadMultipleStatus (void)
- tNfcRwT5TConnectionStatus NFC_RW_T5T_getWriteAFIStatus (void)
- tNfcRwT5TConnectionStatus NFC_RW_T5T_getLockAFIStatus (void)
- tNfcRwT5TConnectionStatus NFC_RW_T5T_getGetSysInfoStatus (void)
- tNfcRwT5TConnectionStatus NFC_RW_T5T_getRawWriteStatus (void)
- uint8_t NFC_RW_T5T_sendInventoryCmd (uint8_t ui8ReqFlag, uint8_t ui8AFI, bool bSendAFI)
- uint8_t NFC_RW_T5T_sendStayQuietCmd (uint8_t ui8ReqFlag)
- uint8_t NFC_RW_T5T_sendReadSingleCmd (uint8_t ui8ReqFlag, uint16_t ui16BlockNumber)
- uint8_t NFC_RW_T5T_sendWriteSingleCmd (uint8_t ui8ReqFlag, uint16_t ui16BlockNumber, const uint8_t *pui8Buffer, uint8_t ui8Length)
- uint8_t NFC_RW_T5T_sendLockBlockCmd (uint8_t ui8ReqFlag, uint16_t ui16BlockNumber)
- uint8_t NFC_RW_T5T_sendSelect (uint8_t ui8ReqFlag)
- uint8_t NFC_RW_T5T_sendResetToReady (uint8_t ui8ReqFlag)
- uint8_t NFC_RW_T5T_sendReadMultipleCmd (uint8_t ui8ReqFlag, uint16_t ui16BlockNumber, uint8_t ui8BlockCount)
- uint8_t NFC_RW_T5T_sendWriteAFICmd (uint8_t ui8ReqFlag, uint8_t ui8NewAFI)
- uint8_t NFC_RW_T5T_sendLockAFICmd (uint8_t ui8ReqFlag)
- uint8_t NFC_RW_T5T_sendGetSysInfoCmd (uint8_t ui8ReqFlag, uint8_t ui8CommandCode)
- uint8_t NFC_RW_T5T_sendRawWriteCmd (uint8_t ui8ReqFlag, uint8_t ui8CommandCode, const uint8_t *pui8Buffer, uint8_t ui8Length, bool bOptionDelay)
- void NFC_RW_T5T_getPacketStatus (uint8_t **pui8RXData, uint8_t *pui8Length)
- uint8_t NFC_RW_T5T_getT5TErrorCode (void)
- uint8_t NFC_RW_T5T_getT5TDSFID (void)
- uint8_t NFC_RW_T5T_getT5TAFI (void)
- void NFC_RW_T5T_getT5TUID (uint8_t **pui8NfcVId, uint8_t *ui8NfcVIdLength)
- uint16_t NFC_RW_T5T_getVICCBlockCount (void)
- uint8_t NFC_RW_T5T_getVICCBlockSize (void)

6.13.1 Detailed Description

A Type V Reader/Writer issues commands based on the ISO15693 specifications. The ISO15693 specification is for Vicinity Integrated Circuit Cards (VICCs) and communication with VICCs that are Type V tags occur at bitrates of 6.62 kbps (low data rate) or 26.48 kbps (high data rate).

For more information on Type V Readers and Tags please read the ISO15693 Specifications.

6.13.2 Function Documentation

6.13.2.1 tNfcRwT5TConnectionStatus NFC_RW_T5T_getGetSysInfoStatus (void)

NFC_RW_T5T_getGetSysInfoStatus - Checks for a Get System Info response.

This function checks if a Get System Info response was received, and returns the Type V Reader status of either a SUCCESS or a FAIL. If a Get System Info response has been received, the current status will be changed to IDLE regardless of whether the Type V Reader status was a SUCCESS or FAIL.

Returns

eStatusTemp is the Type V Reader status.

6.13.2.2 tNfcRwT5TConnectionStatus NFC_RW_T5T_getInventoryStatus (void)

NFC_RW_T5T_getInventoryStatus - Checks for an Inventory response

This function checks if an Inventory response was received, and returns the Type V Reader status of either a SUCCESS or a FAIL. If an Inventory response has been received, the current status will be changed to IDLE regardless of whether the Type V Reader status was a SUCCESS or FAIL.

Returns

eStatusTemp is the Type V Reader status

6.13.2.3 tNfcRwT5TConnectionStatus NFC_RW_T5T_getLockAFIStatus (void)

NFC_RW_T5T_getLockAFIStatus - Checks for a Lock AFI response.

This function checks if a Lock AFI response was received, and returns the Type V Reader status of either a SUCCESS or a FAIL. If a Lock AFI response has been received, the current status will be changed to IDLE regardless of whether the Type V Reader status was a SUCCESS or FAIL.

Returns

eStatusTemp is the Type V Reader status.

6.13.2.4 tNfcRwT5TConnectionStatus NFC_RW_T5T_getLockBlockStatus (void)

NFC_RW_T5T_getLockBlockStatus - Checks for a Lock Block response.

This function checks if a Lock Block response was received, and returns the Type V Reader status of either a SUCCESS or a FAIL. If a Lock Block response has been received, the current status will be changed to IDLE regardless of whether the Type V Reader status was a SUCCESS or FAIL.

Returns

eStatusTemp is the Type V Reader status.

6.13.2.5 void NFC_RW_T5T_getPacketStatus (uint8_t ** *pui8RXData,* uint8_t * *pui8Length*)

NFC_RW_T5T_getPacketStatus - Gets the last Type 5 data packet received.

Parameters

pui8RXData	this is a double pointer to receive the current pointer to the location of the Type V data bytes received buffer
pui8Length	this is a pointer to get the length of the received Type V data

This function returns the packet length and the location of the packet contents of the latest Type V data packet that was received.

Returns

None.

6.13.2.6 tNfcRwT5TConnectionStatus NFC_RW_T5T_getRawWriteStatus (void)

NFC_RW_T5T_getRawWriteStatus - Checks for any Type V tag response.

This function checks if a Type V tag response was received, and returns the Type V Reader status of either a SUCCESS or a FAIL. If a Type V tag response has been received, the current status will be changed to IDLE regardless of whether the Type V Reader status was a SUCCESS or FAIL.

Returns

eStatusTemp is the Type V Reader status.

6.13.2.7 tNfcRwT5TConnectionStatus NFC_RW_T5T_getReadMultipleStatus (void)

NFC_RW_T5T_getReadMultipleStatus - Checks for a Read Multiple response.

This function checks if a Read Multiple was received, and returns the Type V Reader status of either a SUCCESS or a FAIL. If a Read Multiple response has been received, the current status will be changed to IDLE regardless of whether the Type V Reader status was a SUCCESS or FAIL.

Returns

eStatusTemp is the Type V Reader status.

6.13.2.8 tNfcRwT5TConnectionStatus NFC_RW_T5T_getReadSingleStatus (void)

NFC_RW_T5T_getReadSingleStatus - Checks for a Read Single response.

This function checks if a Read Single response was received, and returns the Type V Reader status of either a SUCCESS or a FAIL. If a Read Single response has been received, the current status will be changed to IDLE regardless of whether the Type V Reader status was a SUCCESS or FAIL.

Returns

eStatusTemp is the Type V Reader status.

6.13.2.9 tNfcRwT5TConnectionStatus NFC_RW_T5T_getResetToReadyStatus (void)

NFC_RW_T5T_getResetToReadyStatus - Checks for a Reset to Ready response.

This function checks if a Reset to Ready response was received, and returns the Type V Reader status of either a SUCCESS or a FAIL. If a Reset to Ready response has been received, the current status will be changed to IDLE regardless of whether the Type V Reader status was a SUCCESS or FAIL.

Returns

eStatusTemp is the Type V Reader status.

6.13.2.10 tNfcRwT5TConnectionStatus NFC_RW_T5T_getSelectStatus (void)

NFC_RW_T5T_getSelectStatus - Checks for a Select response.

This function checks if a Select response was received, and returns the Type V Reader status of either a SUCCESS or a FAIL. If a Select response has been received, the current status will be changed to IDLE regardless of whether the Type V Reader status was a SUCCESS or FAIL.

Returns

eStatusTemp is the Type V Reader status.

6.13.2.11 uint8_t NFC_RW_T5T_getT5TAFI (void)

NFC_RW_T5T_getT5TAFI - Returns the AFI read from the Type V tag.

This function returns the AFI from the Type V Tag. The AFI is acquired through the Get System Information command.

Returns

g_ui8T5TAFI is the AFI of the Type V Tag.

6.13.2.12 uint8_t NFC_RW_T5T_getT5TDSFID (void)

NFC_RW_T5T_getT5TDSFID - Returns the DSFID read from the Type V tag.

This function returns the DSFID from the Type V Tag. The DSFID is acquired through the Inventory and Get System Information commands.

Returns

g_ui8T5TDSFID is the DSFID of the Type V Tag.

6.13.2.13 uint8_t NFC_RW_T5T_getT5TErrorCode (void)

NFC_RW_T5T_getT5TErrorCode - Returns the current Type V error code.

This function returns the current Type V error code. This is important for for the application layer in order to handle errors in communication with Type V tags.

Returns

g_ui8T5TErrorCode is the current Type V error code.

6.13.2.14 void NFC_RW_T5T_getT5TUID (uint8_t ** *pui8NfcVId,* uint8_t * *ui8NfcVIdLength*)

NFC_RW_T5T_getT5TUID - Returns the UID read from the Type V tag.

This function returns the UID from the Type V Tag. The UID is acquired through the Inventory command, and must be used in order to issue any addressed Type V commands.

Returns

g_ui8T5TUID is the UID of the Type V Tag.

6.13.2.15 uint16_t NFC_RW_T5T_getVICCBlockCount (void)

NFC_RW_T5T_getVICCBlockCount - Returns number of blocks in the Type V tag.

This function returns the number of memory blocks that are contained within the Type V tag. This information is acquired through the Get System Information command. Using the block count along with the VICC block size it is possible to determine the total memory size of the Type V tag.

Returns

g_ui16T5TVICCBlockCount is the number of blocks in the Type V tag.

6.13.2.16 uint8_t NFC_RW_T5T_getVICCBlockSize (void)

NFC_RW_T5T_getVICCBlockSize - Returns the block size for the Type V tag.

This function returns the number of bytes per block for the Type V tag. This information is acquired through the Get System Information command. Using the size per block along with the VICC block count it is possible to determine the total memory size of the Type V tag. Note that the size returned by this function is the one less than the actual size. Therefore a size returned as 0x03 would indicate 4 bytes per block.

Returns

g_ui8T5TVICCBlockSize is the number of bytes per block in the Type V tag.

6.13.2.17 tNfcRwT5TConnectionStatus NFC_RW_T5T_getWriteAFIStatus (void)

NFC_RW_T5T_getWriteAFIStatus - Checks for a Write AFI response.

This function checks if a Write AFI response was received, and returns the Type V Reader status of either a SUCCESS or a FAIL. If a Write AFI response has been received, the current status will be changed to IDLE regardless of whether the Type V Reader status was a SUCCESS or FAIL.

Returns

eStatusTemp is the Type V Reader status.

6.13.2.18 tNfcRwT5TConnectionStatus NFC_RW_T5T_getWriteSingleStatus (void)

NFC_RW_T5T_getWriteSingleStatus - Checks for a Write Single response.

This function checks if a Write Single response was received, and returns the Type V Reader status of either a SUCCESS or a FAIL. If a Write Single response has been received, the current status will be changed to IDLE regardless of whether the Type V Reader status was a SUCCESS or FAIL.

Returns

eStatusTemp is the Type V Reader status.

6.13.2.19 void NFC_RW_T5T_init (void)

NFC_RW_T5T_init - Initializes the variables for the Type V Reader/Writer.

This function must be called prior to any other NFC_RW_T5T function in in order to properly initialize all variables which have been declared. Failure to do so may cause the software to enter unknown states.

Returns

None.

6.13.2.20 tNfcRwT5TConnectionStatus NFC_RW_T5T_processReceivedData (uint8_t * *pui8RxBuffer,* uint8_t *ui8Length*)

NFC_RW_T5T_processReceivedData - Processes the Type V Data Received.

Parameters

pui8RxBuffer	is the pointer to where T5T data can be received.
ui8Length	is the length of the T5T data packet received.

This function is used to handle the processing of all Type V data packets that have been received from Type V tags. This function includes a timeout monitor to ensure the software will not be locked up if the tag is unexpectedly removed from the RF field.

This function will wait for replies specific to the command that was issued. If it receives an expected reply then it will indicated a successful response and the application layer would then have to determine what command to issue next. This gives developers full control over the sequencing of Type V Reader commands. If an unexpected reply is received, it will be treated as a protocol error and the Initiator state machine will reset.

Returns

g_eNfcRwT5TCommStatus returns the current status of the Type V Reader based on what response was received.

6.13.2.21 uint8_t NFC_RW_T5T_sendGetSysInfoCmd (uint8_t *ui8ReqFlag,* uint8_t *ui8CommandCode*)

NFC_RW_T5T_sendGetSysInfoCmd - Sends the Get System Information Command.

Parameters

ui8ReqFlag	is the Request Flag to be sent with the Type V command
ui8CommandCode	is the command code for the issued ISO15693 command

This function is used set the correct state for sending a Get System Info command, and also to process any application based variables that have been passed into the Type V Reader layer.

Returns

ui8Status indicates if the state was successfully changed or not.

6.13.2.22 uint8_t NFC_RW_T5T_sendInventoryCmd (uint8_t *ui8ReqFlag,* uint8_t *ui8AFI,* bool *bSendAFI*)

NFC_RW_T5T_sendInventoryCmd - Sends the Inventory Command.

Parameters

ui8ReqFlag	is the Request Flag to be sent with the Type V command
ui8AFI	is the AFI that may be sent by with the Inventory command
bSendAFI	is a boolean to determine whether the AFI is sent or not

This function is used set the correct state for sending an Inventory command, and also to process any application based variables that have been passed into the Type V Reader layer.

Returns

ui8Status indicates if the state was successfully changed or not.

6.13.2.23 uint8_t NFC_RW_T5T_sendLockAFICmd (uint8_t *ui8ReqFlag*)

NFC_RW_T5T_sendLockAFICmd - Sends the Lock AFI Command.

Parameters

ui8ReqFlag	is the Request Flag to be sent with the Type V command

This function is used set the correct state for sending a Lock AFI command, and also to process any application based variables that have been passed into the Type V Reader layer.

Returns

ui8Status indicates if the state was successfully changed or not.

6.13.2.24 uint8_t NFC_RW_T5T_sendLockBlockCmd (uint8_t *ui8ReqFlag,* uint16_t *ui16BlockNumber*)

NFC_RW_T5T_sendLockBlockCmd - Sends the Lock Block Command.

Parameters

ui8ReqFlag	is the Request Flag to be sent with the Type V command
ui16BlockNumber	is the Block Number to be read from the tag

This function is used set the correct state for sending a Lock Block command, and also to process any application based variables that have been passed into the Type V Reader layer.

Returns

ui8Status indicates if the state was successfully changed or not.

6.13.2.25 uint8_t NFC_RW_T5T_sendRawWriteCmd (uint8_t *ui8ReqFlag,* uint8_t *ui8CommandCode,* const uint8_t * *pui8Buffer,* uint8_t *ui8Length,* bool *bOptionDelay*)

NFC_RW_T5T_sendRawWriteCmd - Sends a raw ISO15693 formatted data packet.

Parameters

ui8ReqFlag	is the Request Flag to be sent with the Type V command
ui8CommandCode	is the command code for the issued ISO15693 command
pui8Buffer	is a pointer to additional data for the ISO15693 command
ui8Length	is the length of the data stored at pui8Buffer
bOptionDelay	is used to indicate if a delay is required when the option flag is set as described in the ISO15693 specifications //! This function is used set the correct state for sending a Raw data packet for ISO15693, and also to process any application based variables that have been passed into the Type V Reader layer.

Returns

ui8Status indicates if the state was successfully changed or not.

6.13.2.26 uint8_t NFC_RW_T5T_sendReadMultipleCmd (uint8_t *ui8ReqFlag,* uint16_t *ui16BlockNumber,* uint8_t *ui8BlockCount*)

NFC_RW_T5T_sendReadMultipleCmd - Sends the Read Multiple Command.

Parameters

ui8ReqFlag	is the Request Flag to be sent with the Type V command
ui16BlockNumber	is the starting Block Number to be read from the tag
ui8BlockCount	is the Total Number of Blocks to be read from the tag

This function is used set the correct state for sending a Read Multiple command, and also to process any application based variables that have been passed into the Type V Reader layer.

Returns

ui8Status indicates if the state was successfully changed or not.

6.13.2.27 uint8_t NFC_RW_T5T_sendReadSingleCmd (uint8_t *ui8ReqFlag,* uint16_t *ui16BlockNumber*)

NFC_RW_T5T_sendReadSingleCmd - Sends the Read Single Command.

Parameters

ui8ReqFlag	is the Request Flag to be sent with the Type V command
ui16BlockNumber	is the Block Number to be read from the tag

This function is used set the correct state for sending a Read Single command, and also to process any application based variables that have been passed into the Type V Reader layer.

Returns

ui8Status indicates if the state was successfully changed or not.

6.13.2.28 uint8_t NFC_RW_T5T_sendResetToReady (uint8_t *ui8ReqFlag*)

NFC_RW_T5T_sendResetToReady - Sends the Reset to Ready Command.

Parameters

ui8ReqFlag	is the Request Flag to be sent with the Type V command

This function is used set the correct state for sending a Reset to Ready command, and also to process any application based variables that have been passed into the Type V Reader layer.

Returns

ui8Status indicates if the state was successfully changed or not.

6.13.2.29 uint8_t NFC_RW_T5T_sendSelect (uint8_t *ui8ReqFlag*)

NFC_RW_T5T_sendSelect - Sends the Select Command.

Parameters

ui8ReqFlag	is the Request Flag to be sent with the Type V command

This function is used set the correct state for sending a Select command, and also to process any application based variables that have been passed into the Type V Reader layer.

Returns

ui8Status indicates if the state was successfully changed or not.

6.13.2.30 uint8_t NFC_RW_T5T_sendStayQuietCmd (uint8_t *ui8ReqFlag*)

NFC_RW_T5T_sendStayQuietCmd - Sends the Stay Quiet Command.

Parameters

ui8ReqFlag	is the Request Flag to be sent with the Type V command

This function is used set the correct state for sending a Stay Quiet command, and also to process any application based variables that have been passed into the Type V Reader layer.

Returns

ui8Status indicates if the state was successfully changed or not.

6.13.2.31 uint8_t NFC_RW_T5T_sendWriteAFICmd (uint8_t *ui8ReqFlag,* uint8_t *ui8NewAFI*)

NFC_RW_T5T_sendWriteAFICmd - Sends the Write AFI Command.

Parameters

ui8ReqFlag	is the Request Flag to be sent with the Type V command
ui8NewAFI	is the new AFI value to be written to the tag

This function is used set the correct state for sending a Write AFI command, and also to process any application based variables that have been passed into the Type V Reader layer.

Returns

ui8Status indicates if the state was successfully changed or not.

6.13.2.32 uint8_t NFC_RW_T5T_sendWriteSingleCmd (uint8_t *ui8ReqFlag,* uint16_t *ui16BlockNumber,* const uint8_t * *pui8Buffer,* uint8_t *ui8Length*)

NFC_RW_T5T_sendWriteSingleCmd - Sends the Write Single Command.

Parameters

ui8ReqFlag	is the Request Flag to be sent with the Type V command
ui16BlockNumber	is the Block Number to be read from the tag

This function is used set the correct state for sending a Write Single command, and also to process any application based variables that have been passed into the Type V Reader layer.

Returns

ui8Status indicates if the state was successfully changed or not.

6.13.2.33 void NFC_RW_T5T_stateMachine (void)

NFC_RW_T5T_stateMachine - Conditions and transmits the data packets for Type V Reader commands.

This function constructs the data packets for Type V commands based on the current status. The packet is then transmitted at the end of the state machine. Currently the supported Type V commands are: Inventory, Stay Quiet, Read Single Block, Write Single Block, Lock Block, Read Multiple Blocks, Select, Reset to Ready, Write AFI, Lock AFI, and Get System Information. A raw packet write command is also available to handle any commands that are not directly supported by the software.

Returns

None.

6.14 NFC Target API Functions

Functions

- void NFC_Target_Init ()
- uint16_t NFC_Target_GetPCDTimeOut (void)
- void NFC_Target_SetState (tNfcTargetState eState)
- tNfcTargetState NFC_Target_StateMachine (uint8_t ∗pui8Payload, uint8_t ui8Length, t_sTRF79x0_TargetMode sTargetMode)
- tNfcTargetState NFC_Target_getState (void)
- bool NFC_Target_CompareBuffers (uint8_t ∗puiBuffer1, uint8_t ∗puiBuffer2, uint8_t ui8Length)

6.14.1 Detailed Description

NFC Target Statemachine based on the NFC Forum Activity Spec Ver 1.0.

6.14.2 Function Documentation

6.14.2.1 bool NFC_Target_CompareBuffers (uint8_t ∗ *puiBuffer1,* uint8_t ∗ *puiBuffer2,* uint8_t *ui8Length*)

NFC_Target_CompareBuffers - compares if two buffers are the same.

Parameters

puiBuffer1	is the pointer for source buffer 1.
puiBuffer2	is the pointer for source buffer 2.
ui8Length	is the number of bytes to compare between buffer 1 and 2.

This function checks if two buffers have the same values for ui8Length values.

Returns

bRet is true if both buffers have the same value, false otherwise.

6.14.2.2 uint16_t NFC_Target_GetPCDTimeOut (void)

NFC_Target_GetPCDTimeOut - returns current PCD timeout.

This function returns the current PCD timeout.

Returns

g_ui16PCDTimeOut the current PCD timeout.

6.14.2.3 tNfcTargetState NFC_Target_getState (void)

NFC_Target_getState - returns the current target state.

This function returns g_eCurrentTargetState.

Returns

g_eCurrentTargetState the current target state.

6.14.2.4 void NFC_Target_Init ( )

NFC_Target_Init - Initializes the variables for the target statemachine.

This function must be called prior to any other NFC_Target function in in order to properly initialize all variables which have been declared. Failure to do so may cause the software to enter unknown states. It will initialize the NFC_A, NFC_B, NFC_F, NFC_DEP, and ISO_DEP layers.

Returns

None.

6.14.2.5 void NFC_Target_SetState (tNfcTargetState *eState*)

NFC_Target_SetState - sets the current target state.

Parameters

eState	the target state

This function sets g_eCurrentTargetState with a new target state (eState), used inside the NFC_Target_StateMachine() function.

Returns

None.

6.14.2.6 tNfcTargetState NFC_Target_StateMachine (uint8_t * *pui8Payload,* uint8_t *ui8Length,* t_sTRF79x0_TargetMode *sTargetMode*)

NFC_Target_StateMachine - Handles incoming commands from PCD, and sets the PCD timeout based on the NFC technologies that get activated.

Parameters

pui8Payload	is the pointer to data received from the PCD.
ui8Length	is the number of bytes received from the PCD.
sTargetMode	is the Target mode enabled in the transceiver.

This function handles the target statemachine for the application where the transceiver is in target / listen mode. It handles the anti-collision / activation of NFC-A, NFC-B and NFC-F. Once the technologies are activated it calls the NFCDEP_processReceivedRequest() function for P2P mode, and ISODEP_processReceivedRequest() function for T4T tags.

Returns

None.

6.15 NFC SNEP API Functions

Functions

- void SNEP_init (void)
- void SNEP_setMaxPayload (uint8_t ui8MaxPayload)
- uint8_t SNEP_stateMachine (uint8_t *pui8BufferPtr)
- uint8_t SNEP_setupPacket (uint8_t *pui8PacketPtr, uint32_t ui32PacketLength, uint8_t ui8FragmentLength)
- uint8_t SNEP_getTxBufferStatus (void)
- uint8_t SNEP_enqueue (uint8_t *pui8PacketPtr, uint8_t ui8Length)
- uint8_t SNEP_sendRequest (uint8_t *pui8DataPtr, tSNEPCommands eRequestCmd)
- uint8_t SNEP_sendResponse (uint8_t *pui8DataPtr, tSNEPCommands eResponseCmd)
- void SNEP_processReceivedData (uint8_t *pui8RxBuffer, uint16_t ui16RxLength)
- void SNEP_getReceiveStatus (tPacketStatus *peReceiveFlag, uint16_t *pui16length, uint8_t **pui8DataPtr, uint32_t *ui32NdefTotalLength)
- tSNEPConnectionStatus SNEP_getProtocolStatus (void)
- void SNEP_setProtocolStatus (tSNEPConnectionStatus eProtocolStatus)

Variables

- uint32_t **g_ui32SNEPPacketLength**
- uint8_t **g_ui8FragQueue**
- uint8_t **g_ui8TFragLength**
- uint8_t **g_pui8SNEPTxBuffer** [SNEP_MAX_BUFFER]
- uint8_t * **g_pui8SNEPRxPacketPtr**
- uint32_t **g_ui32SNEPRemainingRxPayloadBytes**
- uint16_t **g_ui16SNEPReceivedBytes**
- tPacketStatus **g_eRxPacketStatus**
- tSNEPConnectionStatus **g_eSNEPConnectionStatus**
- uint8_t **g_ui8MaxPayload**

6.15.1 Detailed Description

Simple NDEF Exchange Protocol is an application protocol used by the LLCP layer to send / receive NDEFs between two NFC Forum Devices operating in Peer to Peer Mode (1 Target and 1 Initiator). For more information on SNEP please read the NFC Simple NDEF Exchange Protocol Specification Version 1.0.

6.15.2 Function Documentation

6.15.2.1 uint8_t SNEP_enqueue (uint8_t * *pui8PacketPtr,* uint8_t *ui8Length*)

SNEP_enqueue - Used to queue fragments of the remaining of the packet.

Parameters

pui8PacketPtr	is the starting pointer of the data to be transmitted.
ui8Length	is the length of the fragment. It must not greater than SNEP_getTxBufferStatus(). If the remaining packet size is greater than SNEP_getTxBufferStatus(), then it must be equal to ui8FragmentLength for best throughput.

This function must be called by main application after calling SNEP_getTxBufferStatus() to send the remaining of the packet.

Returns

ui8PacketSetupStatus is STATUS_SUCCESS if the fragment was queued or STATUS_FAIL if ui8Length was greater than the available fragment size.

6.15.2.2 tSNEPConnectionStatus SNEP_getProtocolStatus (void)

SNEP_getProtocolStatus - returns current SNEP Connection Status enumeration

This function returns the current SNEP status flag. It must be called inside LLCP_processReceivedData() to decide if further I-PDUs are required - this occurs when there are requests/responses queued.

Returns

g_eSNEPConnectionStatus the current connection status flag.

6.15.2.3 void SNEP_getReceiveStatus (tPacketStatus * *peReceiveFlag,* uint16_t * *pui16length,* uint8_t ** *pui8DataPtr,* uint32_t * *ui32NdefTotalLength*)

SNEP_getReceiveStatus - Get RxStatus flag, Clear packet status flag, retrieve length of data, retrieve data

Parameters

peReceiveFlag	is a pointer to store the rx state status.
pui8length	is a pointer to store the number of received bytes.
pui8DataPtr	is a double pointer to store the pointer of data received.

The *peReceiveFlag* parameter can be any of the following:

- **RECEIVED_NO_FRAGMENT** - No Fragment has been received
- **RECEIVED_FIRST_FRAGMENT** - First fragment has been received.
- **RECEIVED_N_FRAGMENT** - N Fragment has been received.
- **RECEIVED_FRAGMENT_COMPLETED** - End of the fragment has been received.

This function must be called in the main application after the NFC_P2P_stateMachine() is called to ensure the data received is moved to another buffer / handled when a fragment is received.

Returns

None

6.15.2.4 uint8_t SNEP_getTxBufferStatus (void)

SNEP_getTxBufferStatus - Returns the size of the available tx fragment.

This function must be called by main application before calling SNEP_setupPacket() to check for the available tx fragment size.

Returns

g_ui8FragQueue is the size of the available tx fragment.

6.15.2.5 void SNEP_init (void)

SNEP_init - Initialize the Simple NDEF Exchange Protocol driver.

This function must be called prior to any other function offered by the SNEP driver. This function initializes the SNEP status, tx/rx packet length and maximum payload size. This function must be called by the LLCP_init().

Returns

None.

6.15.2.6 void SNEP_processReceivedData (uint8_t * *pui8RxBuffer,* uint16_t *ui16RxLength*)

SNEP_processReceivedData - Processes the data received from a client/server.

Parameters

pui8RxBuffer	is the starting pointer of the SNEP request/response received.
ui8RxLength	is the length of the SNEP request/response received.

This function handles the requests/responses received inside an I-PDU in the LLCP layer. This function must be called inside LLCP_processReceivedData().

Returns

None

6.15.2.7 uint8_t SNEP_sendRequest (uint8_t * *pui8DataPtr,* tSNEPCommands *eRequestCmd*)

SNEP_sendRequest - Sends request to the server.

Parameters

pui8DataPtr	is the start pointer where the request will be written.
eRequestCmd	is the request command to be sent.

The *eRequestCmd* parameter can be any of the following:

- **SNEP_REQUEST_CONTINUE** - Send remaining fragments
- **SNEP_REQUEST_GET** - Return an NDEF message
- **SNEP_REQUEST_PUT** - Accept an NDEF message
- **SNEP_REQUEST_REJECT** - Do not send remaining fragments

This function sends a SNEP request from the SNEP client to a SNEP server. It must be called from the SNEP_stateMachine().

Returns

ui8offset is the length of the request written starting at pui8DataPtr.

6.15.2.8 uint8_t SNEP_sendResponse (uint8_t * *pui8DataPtr,* tSNEPCommands *eResponseCmd*)

SNEP_sendResponse - Sends response to the client.

Parameters

pui8DataPtr	is the start pointer where the response will be written.
eResponseCmd	is the response command to be sent.

The *eResponseCmd* parameter can be any of the following:

- **SNEP_RESPONSE_CONTINUE** - Continue send remaining fragments
- **SNEP_RESPONSE_SUCCESS** - Operation succeeded
- **SNEP_RESPONSE_NOT_FOUND** - Resource not found
- **SNEP_RESPONSE_EXCESS_DATA** - Resource exceeds data size limit
- **SNEP_RESPONSE_BAD_REQUEST** - Malformed request not understood
- **SNEP_RESPONSE_NOT_IMPLEMENTED** - Unsupported functionality requested
- **SNEP_RESPONSE_UNSUPPORTED_VER** - Unsupported protocol version
- **SNEP_RESPONSE_REJECT** - Do not send remaining fragments

This function sends a SNEP response from the SNEP server to a SNEP client. It must be called from the SNEP_stateMachine().

Returns

ui8offset is the length of the response written starting at pui8DataPtr.

6.15.2.9 void SNEP_setMaxPayload (uint8_t *ui8MaxPayload*)

SNEP_setMaxPayload - Set the Maximum size of each fragment.

Parameters

ui8MaxPayload	is the maximum size of each fragment.

This function must be called inside LLCP_processTLV() to define the maxium size of each fragment based on the Maximum Information Unit (MIU) supported by the target/initiator.

Returns

None.

6.15.2.10 void SNEP_setProtocolStatus (tSNEPConnectionStatus *eProtocolStatus*)

SNEP_setProtocolStatus - sets current SNEP Connection Status enumeration

Parameters

eProtocolStatus	is the status flag used by the SNEP_stateMachine() to send request/response. Only when it is set to SNEP_CONNECTION_IDLE, new send transactions are allowed.

The *eProtocolStatus* parameter can be any of the following:

- **SNEP_CONNECTION_IDLE** - No ongoing tx/rx
- **SNEP_WRONG_VERSION_RECEIVED** - Wrong Version Received
- **SNEP_CONNECTION_RECEIVED_FIRST_PACKET** - Received First Fragment
- **SNEP_CONNECTION_RECEIVING_N_FRAGMENTS** - Received N Fragment
- **SNEP_CONNECTION_WAITING_FOR_CONTINUE** - Waiting for Continue Rsp.
- **SNEP_CONNECTION_WAITING_FOR_SUCCESS** - Waiting for Success Rsp.
- **SNEP_CONNECTION_SENDING_N_FRAGMENTS** - Sending N Fragment
- **SNEP_CONNECTION_SEND_COMPLETE** - Send Completed
- **SNEP_CONNECTION_RECEIVE_COMPLETE** - Receive Completed
- **SNEP_CONNECTION_EXCESS_SIZE** - Received Excess Size Req.

This function is called inside LLCP_processReceivedData(), to set the g_eSNEPConnectionStatus flag to SNEP_CONNECTION_IDLE after the a send transaction is completed to allow for further send transactions.

Returns

None

6.15.2.11 uint8_t SNEP_setupPacket (uint8_t * *pui8PacketPtr,* uint32_t *ui32PacketLength,* uint8_t *ui8FragmentLength*)

SNEP_setupPacket - Setup a packet to be transmitted via the SNEP transport layer.

Parameters

pui8PacketPtr	is the starting pointer of the data to be transmitted.
ui32PacketLength	is the total length of the packet.
ui8FragmentLength	is the length of the first fragment. It must not greater than SNEP_getTxBufferStatus(). If the total packet size is greater than SNEP_getTxBufferStatus(), then ui8FragmentLength must be equal to SNEP_getTxBufferStatus() for best throughput.

This function must be called by main application after calling SNEP_getTxBufferStatus() to initialize the packet to be sent to the SNEP server. If the total packet is greater than SNEP_getTxBufferStatus(), the remaining of the packet must be queued with SNEP_enqueue().

Returns

ui8PacketSetupStatus is STATUS_SUCCESS if the packet was queued or STATUS_FAIL if there was a current transfer ongoing.

6.15.2.12 uint8_t SNEP_stateMachine (uint8_t * *pui8BufferPtr*)

SNEP_stateMachine - Decides which request/response to fill inside the LLCP packet.

Parameters

pui8BufferPtr	is the starting pointer where the SNEP request / response should be written to.

This function must be called inside LLCP_sendI() to check if there are any queued fragment to be transmitted or if

there is a queued response. If the packet length returned is equal to 0, then there are no commands to be sent.

Returns

ui8PacketLength is the length of the request/response written to pui8BufferPtr.

Chapter 7

Data Structure Documentation

7.1 NfcAStatus Struct Reference

Data Fields

- bool **bNfcDepSupport**
- bool **bISODepSupport**
- tUidLength **eUidLength**

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

- nfc_a.h

7.2 NfcFStatus Struct Reference

Data Fields

- bool **bNfcDepSupoort**

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

- nfc_f.h

7.3 t_sNfcDEP_P2PSetup Union Reference

Data Fields

- struct {
    uint8_t **bP2PSupportLLCP**: 1
    uint8_t bP2PSupportLoopback: 1
    uint8_t ui3P2PMaxTimeouts: 3
    uint8_t ui3P2PMaxProtocolErrors: 3
  } **bits**

- uint8_t **ui8byte**

7.3.1 Field Documentation

7.3.1.1 uint8_t t_sNfcDEP_P2PSetup::bP2PSupportLoopback

Support LLCP Protocol

7.3.1.2 uint8_t t_sNfcDEP_P2PSetup::ui3P2PMaxProtocolErrors

Maximum number of Timeouts allowed before reset

7.3.1.3 uint8_t t_sNfcDEP_P2PSetup::ui3P2PMaxTimeouts

Support Loopback for DEP packets

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

- nfc_dep.h

7.4 tNfcMode Union Reference

Data Fields

- struct {
  uint8_t **bNfcModePoll**: 1
  uint8_t **bNfcModeListen**: 1
  uint8_t **bReserved**: 1
  } **bits**

- uint8_t **ui8Byte**

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

- nfc_controller.h

7.5 tT3TPacketData Struct Reference

Data Fields

- uint8_t * **pui8Services**
- uint8_t **ui8ServiceCount**
- uint8_t * **pui8Blocks**
- uint8_t **ui8BlockLen**
- uint8_t * **pui8Data**
- uint8_t **ui8DataLen**
- bool **bThreeByteBlock**

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

- nfc_rw_t3t.h