AN13861

PN7160 card emulation

Rev. 1.0 — 26 April 2023 Application note

Document Information

Information	Content
Keywords	NFCC, NFC, CE, PN7160
Abstract	This document provides information about the PN7160 card emulation feature.

Revision history

Rev	Date	Description
1.0	20230426	Initial version

1 Introduction

The goal of this document is to give examples on how to properly set card emulation (CE) for specific CE scenario. For detailed explanation of CE architecture check user manual [5].

For hardware settings of card emulation, refer to [13] and [14].

Requirements:

- Knowledge of MCUXpresso and/or Android and/or Linux
- PN7160 Knowledge, for example, how to push configuration files on Linux/Android. [PN7160 Android porting guide [10], PN7160 Linux porting guide [11], NXP-NCI2.0 MCUXpresso examples guide [12]].

2 Card emulation by the DH-NFCEE

Scenario 1 (Card emulation by the DH-NFCEE) is a scenario, where the device host is responsible for emulating a card. An external Reader/Writer accesses the DH-NFCEE emulating the contactless card, through the PN7160.

Figure 1 shows the flow of communication. We can see that in this scenario PN7160 is just a device in the middle, responsible for forwarding the communication from the external Reader/Writer to DH-NFCEE. For more information, check UM [5].

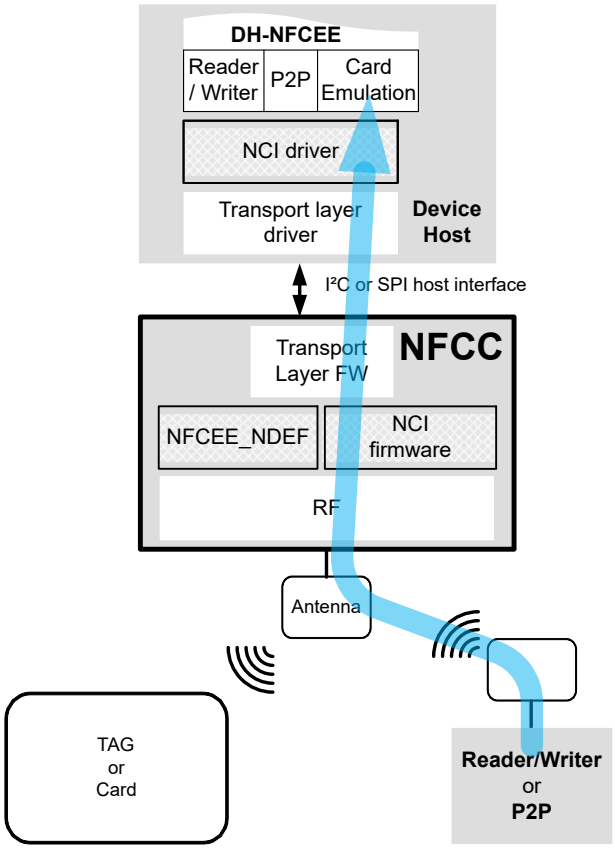

Figure 1. Card emulation by DH_NFCEE

3 Card emulation over NFCC

Figure 2 shows scenario 2 (card emulation over NFCC). In this scenario, the card is emulated on the NFCC. We can access to NFCEE_NDEF either via DH-NFCEE or from RF field. For more information, check UM [5].

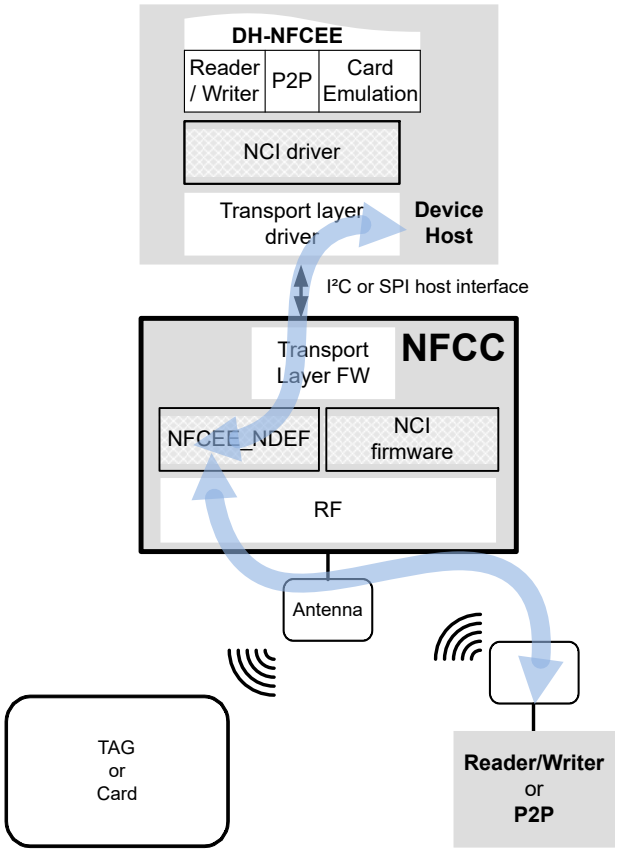

Figure 2. Card emulation over NFCC

4 Switching between scenarios

To switch between scenario 1 and scenario 2 on Android or Linux, we can use configuration file (libnfc-nxp.conf). There we must set the NXP_T4T_NFCEE_ENABLE flag.

If we want to use scenario 1 (card emulation by the DH-NFCEE): **NXP_T4T_NFCEE_ENABLE = 0x00**.

If we want to use scenario 2 (card emulation over NFCC): **NXP_T4T_NFCEE_ENABLE = 0x01**.

In the background, this flag is convert to NCI command. DH sends a CORE_SET_CONFIG with parameter 0xA095 in RF_IDLE_STATE. Figure 3 shows NXP_T4T_NFCEE_ENABLE flag.

```
################################################################################
#T4T NFCEE ENABLE
#bit pos 0 = Enable T4T NFCEE from NFCC (Only wired R/W activated, RF read  only activated)
NXP_T4T_NFCEE_ENABLE=0x00
```

Figure 3. T4T NFCEE enable in Linux or Android

In the MCUXpresso example, the process is a bit different. For enabling T4T_NFCEE, we must send an NCI command. Refer to Figure 4 for location. Example is available [8].

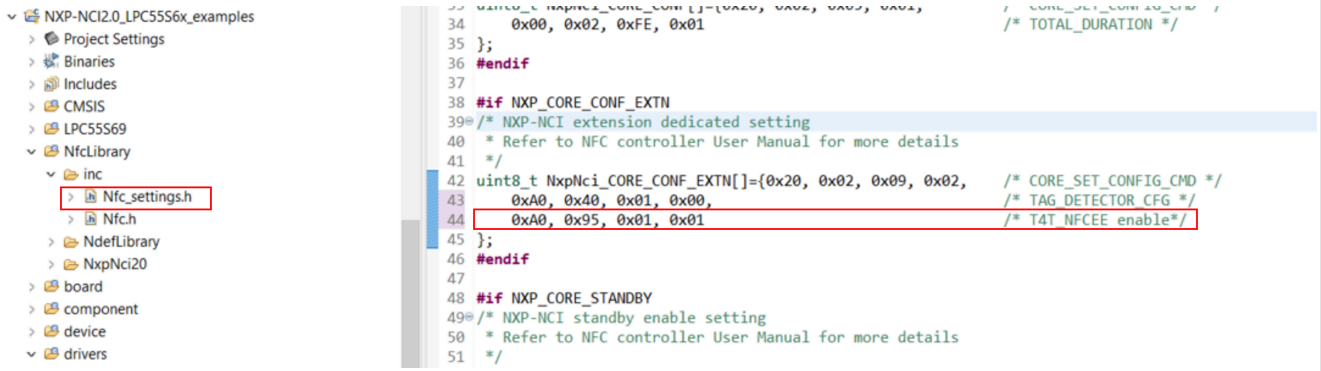

Figure 4. T4T NFCEE enable in MCUXpresso

5 Card emulation configuration

The goal of this chapter is to explain, how to configure CE. Since many settings are used between both versions of CE, every explained setting is valid for both scenario if not stated otherwise.

CE on either technology NFC-A or technology NFC-B, the PN7160 only supports the ISO-DEP protocol. ISO-DEP protocol always mapped to ISO-DEP interface. For more information, check UM [5].

To understand everything explained in following chapters, the knowledge of following standards is mandatory [NCI 2.0 [3], ISOIEC - 14443/3 [1], ISOEIC - 14443/4 [2], NFC Digital protocol [4]].

Together with settings described in following sections, NCI [3] provides other configurations, which can be used together with CE, for example, RF_FIELD_INFO and RF_NFCEE_ACTION. Explanation can be found in [3].

5.1 Type A configuration

Table is showing parameters and values for specific value for Type A configuration.

Table 1. Type A settings values

Name	Explanation	Value	Scenario 1	Scenario 2
LA_BIT_FRAME_SDD	Byte 1 of SENS_RES. SENS_RES explanation here: [4]	0x30	Supported	Supported
LA_PLATFORM_CFG	Byte 2 of SENS_RES. SENS_RES explanation here: [4]	0x31	Supported	Supported
LA_SEL_INFO	Value used to generate SEL_RES. Check [4]	0x32	Supported	Supported
LA_NFCID1	This sets ID. Check digital. [3] and [4]. Max supported size of UID is 7 bytes.	0x33	Supported	Supported
LI_A_RATS_TB1	For setting Frame Waiting Time (FWI) and startup Frame Guard Time (SFGU). For more details, check [1]	0x58	Supported	Supported
LI_A_HIST_BY	Setting Historical Bytes. For more details, check [1] and [4]	0x59	Supported	Supported
LI_A_BIT_RATE	Setting up maximum supported bit rates. For more details, check [3]	0x5B	Supported	Supported
LI_A_RATC_TC1	Support of CID. For more details, check [2] and [3]	0x5C	Supported	Supported

LA_BIT_FRAME_SDD and LA_PLATFORM_CFG are used to build SENS_RES. SENS_RES which is hardcoded to 0x0400 on Linux and Android.

LA_SEL_INFO is used to build SEL_RES. SEL_RES is hardcoded to 0x20 on Linux and Android.

In MCUXpresso project configurations are set in Nfc_settings.h. Codeblock below, shows exact variable.

```
uint8_t NxpNci_CORE_CONF[]={ 0x20, 0x02, 0x31, 0x0F,
    0x85, 0x01, 0x01,
    0x28, 0x01, 0x00,
    0x21, 0x01, 0x00,
    0x30, 0x01, 0x08,
    0x31, 0x01, 0x03,
    0x32, 0x01, 0x60,
    0x38, 0x01, 0x01,
    0x33, 0x04, 0x01, 0x02, 0x03, 0x04,
    0x54, 0x01, 0x06,
    0x50, 0x01, 0x02,
    0x5B, 0x01, 0x00,
    0x80, 0x01, 0x01,
    0x81, 0x01, 0x01,
    0x82, 0x01, 0x0E,
    0x18, 0x01, 0x01
};
```

Configurations are configured in libnfc-nxp.conf for Android and Linux. Codeblock below shows exact location.

```
################################################################################
# Core configuration settings
NXP_CORE_CONF={ 20, 02, 31, 0F,
 85, 01, 01,
 28, 01, 00,
 21, 01, 00,
 30, 01, 08,
 31, 01, 03,
 32, 01, 60,
 38, 01, 01,
 33, 04, 01, 02, 03, 04,
 54, 01, 06,
 50, 01, 02,
 5B, 01, 00,
 80, 01, 01,
 81, 01, 01,
 82, 01, 0E,
 18, 01, 01
}
```

We can see most of settings when we read our emulated card with TagInfo application. Figure 5 shows TagInfo read for scenario 1 and Figure 6 shows TagInfo read for scenario 2.

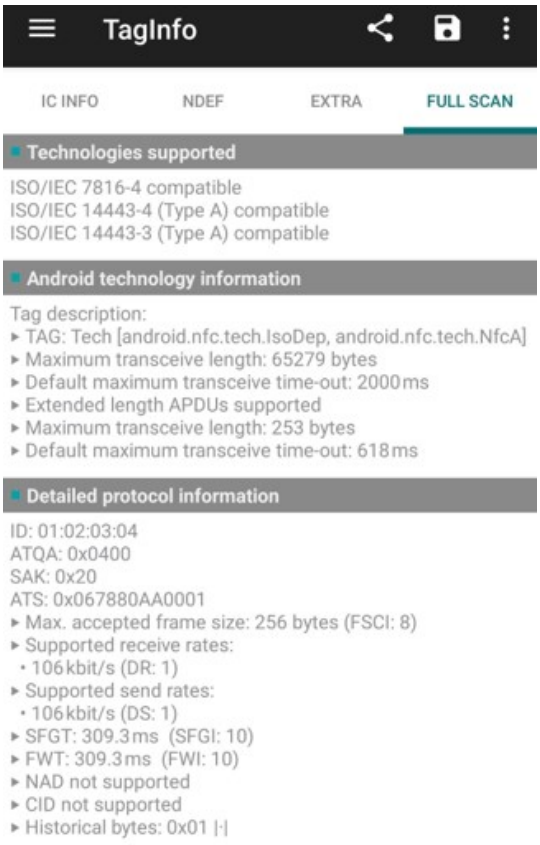

Figure 5. TagInfo read - Scenario 1

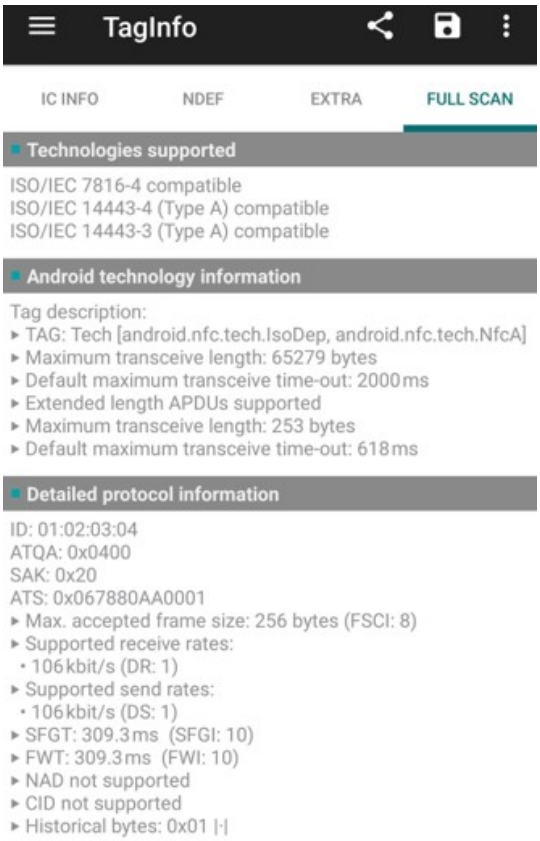

Figure 6. TagInfo read - Scenario 2

5.2 Type B configuration

Table is showing parameters and values for Type B configuration.

Table 2. Type A settings values

Name	Explanation	Value	Scenario 1	Scenario 2
LB_SENSB_INFO	Used to generate Byte 2 of Protocol info within SENSB_RES. Check [4] and [3]	0x38	Supported	Supported
LB_NFCID0	NFC-B identifier of the NFC Forum Device. NFCID0 is always 4 bytes long. Check [4]	0x39	Supported	Supported
LB_APPLICATION_DATA	Application data (Bytes 6-9) of SENSB_RES. Check [4]	0x3A	Supported	Supported
LB_SFGI	Startup Frame Guard Time. Check [4]	0x3B	Supported	Supported
LB_FWI_ADC_FO	Byte 3 of Protocol info within SENSB_RES. Check [4]	0x3C	Supported	Supported
LB_BIT_RATE	Setting up maximum supported bit rates. For more details, check [3]	0x3E	Supported	Supported
LI_B_H_INFO_RESP	Higher Layer - Response field of the ATTRIB. Check [4]	0x5A	Supported	Supported

LB_SENSB_INFO is on Linux and Android hardcoded to 0x01.

Section 5.1 shows location, where we set those parameters.

6 Configuration sequence

After configuration parameters are set, couple of commands must be sent that the parameters are taken into consideration. This document is explaining how commands look like, to understand them into details check UM [5] and NCI specification [3].

6.1 RF_DISCOVER_MAP_CMD

Since PN7160 CE supports only ISO-DEP protocol and RF interface, this command has only one option for set CE mode. Figure 7 shows example of RF_DISCOVER_MAP_CMD. RF protocol and RF interface for listen mode (CE) is always set to ISO-DEP. For detailed explanation of this command, check [3].

```
[NCI][COMMAND][21 00 0A 03 04 03 02 05 03 03 80 01 08]
   RF_DISCOVER_MAP_CMD
      * Number of Mapping Configurations = 3 [0x03]
         --> Mapping Configuration N° 1
            - RF Protocol  = {Std} PROTOCOL_ISO_DEP [0x04]
            - Mode         = the RF Interface is mapped to the RF Protocol both in Listen Mode and Poll Mode [0x03]
            - RF Interface = {Std} ISO-DEP RF Interface [0x02]

         --> Mapping Configuration N° 2
            - RF Protocol  = {Std} PROTOCOL_NFC_DEP [0x05]
            - Mode         = the RF Interface is mapped to the RF Protocol both in Listen Mode and Poll Mode [0x03]
            - RF Interface = {Std} NFC-DEP RF Interface [0x03]

         --> Mapping Configuration N° 3
            - RF Protocol  = {NXP PN547-PN548} MIFARE Classic [0x80]
            - Mode         = the RF Interface is mapped to the RF Protocol in Poll Mode [0x01]
            - RF Interface = RF-IF = RFU [0x08]
```

Figure 7. RF_DISCOVER_MAP_CMD example

This command cannot be changed from configuration files. If someone want to change RF_DISCOVER_MAP_CMD command in Linux or Android, change must be done in HAL layer.

6.2 CORE_SET_CONFIG_CMD

With CORE_SET_CONFIG_CMD command parameters from Section 5 are set. For detailed explanation of this command, check [3].

6.3 RF_SET_LISTEN_MODE_ROUTING_CMD

Description of this command is in UM [5] and in NCI specification [3]. For range of this application note, it is only important, that this command is different between both CE scenarios.

Figure 8 and Figure 9 shows difference in command between both scenarios.

```
[NCI][COMMAND][21 01 17 00 04 03 04 00 00 FE FE 01 03 00 11 04 01 03 00 01 05 00 03 00 39 00]
   RF_SET_LISTEN_MODE_ROUTING_CMD
      * More                        = Last Message [0x00]
      * Number of Routing Entries = 4 [0x04]
         --> Routing Entry N° 1
            - Type   = RFU [0x03]
            - Length = 4 [0x04]
            - Value  = [0x00 0x00 0xFE 0xFE]
               + Route       = DH NFCEE ID [0x00]
               + Power State = RFU [0x00]


         --> Routing Entry N° 2
            - Type   = Protocol-based routing entry [0x01]
            - Length = 3 [0x03]
            - Value  = [0x00 0x11 0x04]
               + Route       = DH NFCEE ID [0x00]
               + Power State = RFU [0x11]
               + Protocol    = {Std} PROTOCOL_ISO_DEP [0x04]


         --> Routing Entry N° 3
            - Type   = Protocol-based routing entry [0x01]
            - Length = 3 [0x03]
            - Value  = [0x00 0x01 0x05]
               + Route       = DH NFCEE ID [0x00]
               + Power State = Switched on [0x01]
               + Protocol    = {Std} PROTOCOL_NFC_DEP [0x05]


         --> Routing Entry N° 4
            - Type   = Technology-based routing entry [0x00]
            - Length = 3 [0x03]
            - Value  = [0x00 0x39 0x00]
               + Route       = DH NFCEE ID [0x00]
               + Power State = RFU [0x39]
               + Technology  = {Std} NFC_RF_TECHNOLOGY_A [0x00]
```

Figure 8. RF_SET_LISTEN_MODE_ROUTING_CMD for scenario 1

```
[NCI][COMMAND][21 01 22 00 05 02 09 10 3B D2 76 00 00 85 01 01 03 04 00 00 FE FE 01 03 00 11 04 01 03 00 01 05 00 03 00 39 00]
   RF_SET_LISTEN_MODE_ROUTING_CMD
      * More                      = Last Message [0x00]
      * Number of Routing Entries = 5 [0x05]
         --> Routing Entry N° 1
            - Type   = AID-based routing entry [0x02]
            - Length = 9 [0x09]
            - Value  = [0x10 0x3B 0xD2 0x76 0x00 0x00 0x85 0x01 0x01]
               + Route       = Dynamically assigned by the NFCC [0x10]
               + Power State = RFU [0x3B]
               + AID         = [0xD2 0x76 0x00 0x00 0x85 0x01 0x01]


         --> Routing Entry N° 2
            - Type   = RFU [0x03]
            - Length = 4 [0x04]
            - Value  = [0x00 0x00 0xFE 0xFE]
               + Route       = DH NFCEE ID [0x00]
               + Power State = RFU [0x00]


         --> Routing Entry N° 3
            - Type   = Protocol-based routing entry [0x01]
            - Length = 3 [0x03]
            - Value  = [0x00 0x11 0x04]
               + Route       = DH NFCEE ID [0x00]
               + Power State = RFU [0x11]
               + Protocol    = {Std} PROTOCOL_ISO_DEP [0x04]


         --> Routing Entry N° 4
            - Type   = Protocol-based routing entry [0x01]
            - Length = 3 [0x03]
            - Value  = [0x00 0x01 0x05]
               + Route       = DH NFCEE ID [0x00]
               + Power State = Switched on [0x01]
               + Protocol    = {Std} PROTOCOL_NFC_DEP [0x05]


         --> Routing Entry N° 5
            - Type   = Technology-based routing entry [0x00]
            - Length = 3 [0x03]
            - Value  = [0x00 0x39 0x00]
               + Route       = DH NFCEE ID [0x00]
               + Power State = RFU [0x39]
               + Technology  = {Std} NFC_RF_TECHNOLOGY_A [0x00]
```

Figure 9. RF_SET_LISTEN_MODE_ROUTING_CMD for scenario 2

6.4 RF_DISCOVER_CMD

RF_DISCOVER_CMD [3] responsibility is starting discovery loop. To select for which technologies we want to poll and/or listen, we must set parameters. For Linux and Android, parameters are located in libnfc-nci.conf.

For selecting listen mode, HOST_LISTEN_TECH_MASK must be set. Following values are supported:

1. 0x01 for Type A only
2. 0x02 for Type B only
3. 0x03 for Type A and Type B

In MCUXpresso project, selection of technologies is done in source file (for example nfc_example_RWandCE.c), where DiscoveryTechnologies variable need to be set.

Figure 10 shows example of RF_DISCOVER_CMD.

```
[NCI][COMMAND][21 03 03 01 80 01]
   RF_DISCOVER_CMD
      * Number of Configurations = 1 [0x01]
         --> Configuration N° 1
            - RF Technology and Mode = {Std} NFC_A_PASSIVE_LISTEN_MODE [0x80]
            - Discovery Frequency    = RF Technology and Mode will be executed in every discovery period [0x01]
```

Figure 10. RF_DISCOVER_CMD example

7 Example description

NXP provides example for both scenarios. Scenario 2 is done in two different examples, since access to CE is possible either from DH or from RF.

Examples can be found here: MCUXpresso [8], Linux (scenario 1) [7], Linux (scenario 2) [9].

7.1 Scenario 1 description

Like mentioned in Section 7.1, DH hosts CE. PN7160 forwards communication to DH, for that we must develop a state machine on DH. The state machine in MCUXpresso example can be found in T4T_NDEF_emu.c, Linux example can be found here: [7].

When external reader is in range, PN7160 reports RF_INTF_ACTIVATED_NTF ([3]) with all information.

Figure 11 shows example of RF_INTF_ACTIVATED_NTF. After this command, external reader will start sending commands for read/write NDEF and so on.

```
[NCI][NOTIFICATION][61 05 0c 01 02 04 80 ff 01 00 80 00 00 01 80]
   RF_INTF_ACTIVATED_NTF
      * RF Discovery ID                    = 1 [0x01]
      * RF Interface                       = {Std} ISO-DEP RF Interface [0x02]
      * RF Protocol                        = {Std} PROTOCOL_ISO_DEP [0x04]
      * Activation RF Technology and Mode  = {Std} NFC_A_PASSIVE_LISTEN_MODE [0x80]
      * Max Data Packet Payload Size       = 255 [0xFF]
      * Initial Number of Credits          = 1 [0x01]
      * Length of RF Technology Specific Parameters = 0 [0x00]
      * RF Technology Specific Parameters = []

      * Data Exchange RF Technology and Mode = {Std} NFC_A_PASSIVE_LISTEN_MODE [0x80]
      * Data Exchange Transmit Bit Rate      = {Std} NFC_BIT_RATE_106: 106 Kbit/s [0x00]
      * Data Exchange Receive Bit Rate       = {Std} NFC_BIT_RATE_106: 106 Kbit/s [0x00]
      * Length of Activation Parameters      = 1 [0x01]
      * Activation Parameters                = [0x80]
         --> RATS Command Param = [0x80]
            - FSD = 256 (Frame Size for proximity coupling Device) [FSDI = 0x8]
            - DID = (Device ID) [0x0]
```

Figure 11. Example of RF_INTF_ACTIVATED_NTF

In Linux and MCUXpresso examples, we must develop a state machine on the DH. To achieve this, some rules must be followed. Check [6] for command set.

To test it, we can use **TagInfo** for read NDEF and **TagWrite** for write NDEF.

Codeblock below shows state machine for MCUXpresso example (similar must be done for Linux).

```
void T4T_NDEF_EMU_Next(unsigned char *pCmd, unsigned short Cmd_size, unsigned char *pRsp,
 unsigned short *pRsp_size)
{
    bool eStatus = false;

    if (!memcmp(pCmd, T4T_NDEF_EMU_APP_Select, sizeof(T4T_NDEF_EMU_APP_Select)))
    {
        *pRsp_size = 0;
        eStatus = true;
        eT4T_NDEF_EMU_State = NDEF_Application_Selected;
    }
```

```
    else if (!memcmp(pCmd, T4T_NDEF_EMU_CC_Select, sizeof(T4T_NDEF_EMU_CC_Select)))
    {
        if(eT4T_NDEF_EMU_State == NDEF_Application_Selected)
        {
            *pRsp_size = 0;
            eStatus = true;
            eT4T_NDEF_EMU_State = CC_Selected;
        }
    }
    else if (!memcmp(pCmd, T4T_NDEF_EMU_NDEF_Select, sizeof(T4T_NDEF_EMU_NDEF_Select)))
    {
        *pRsp_size = 0;
        eStatus = true;
        eT4T_NDEF_EMU_State = NDEF_Selected;
    }
    else if (!memcmp(pCmd, T4T_NDEF_EMU_Read, sizeof(T4T_NDEF_EMU_Read)))
    {
        if(eT4T_NDEF_EMU_State == CC_Selected)
        {
            unsigned short offset = (pCmd[2] << 8) + pCmd[3];
            unsigned char length = pCmd[4];

            if(length <= (sizeof(T4T_NDEF_EMU_CC) + offset + 2))
            {
                memcpy(pRsp, &T4T_NDEF_EMU_CC[offset], length);
                *pRsp_size = length;
                eStatus = true;
            }

        }
        else if (eT4T_NDEF_EMU_State == NDEF_Selected)
        {
            unsigned short offset = (pCmd[2] << 8) + pCmd[3];
            unsigned char length = pCmd[4];

            if(length <= (T4T_NdefMessage_size + offset + 2))
            {
                T4T_NDEF_EMU_FillRsp(pRsp, offset, length);
                *pRsp_size = length;
                eStatus = true;
            }
        }
    }
    else if (!memcmp(pCmd, T4T_NDEF_EMU_Write, sizeof(T4T_NDEF_EMU_Write)))
    {
        if (eT4T_NDEF_EMU_State == NDEF_Selected)
        {

            unsigned short offset = (pCmd[2] << 8) + pCmd[3];
            unsigned char length = pCmd[4];
            if(offset + length <= sizeof(T4T_NdefMessageWritten))
            {
                memcpy(&T4T_NdefMessageWritten[offset-2], &pCmd[5], length);
                pT4T_NdefMessage = T4T_NdefMessageWritten;
                T4T_NdefMessage_size = (pCmd[5] << 8) + pCmd[6];
                *pRsp_size = 0;
                eStatus = true;
            }
        }
    }

    if (eStatus == true)
    {
        memcpy(&pRsp[*pRsp_size], T4T_NDEF_EMU_OK, sizeof(T4T_NDEF_EMU_OK));
        *pRsp_size += sizeof(T4T_NDEF_EMU_OK);
    } else
    {
```

```
        memcpy(pRsp, T4T_NDEF_EMU_NOK, sizeof(T4T_NDEF_EMU_NOK));
        *pRsp_size = sizeof(T4T_NDEF_EMU_NOK);
        T4T_NDEF_EMU_Reset();
    }
}
```

On Android, everything is done in AOSP MW code and usually no need to change anything.

7.2 Scenario 2 description

Scenario 2 is a bit more complex, since access to NFCEE_NDEF can be made from DH or from RF field. Following chapters describe, how to access from DH and from RF.

7.2.1 DH access

DH access example can be found here (Linux ([7], [9]), MCUXpresso ([8])). On Android, everything is handled by AOSP middleware and T4TDemo app ([15]). In this section, MCUXpresso codeblocks are used. To access NFCEE_NDEF through DH, specific command order must be followed and state need to be "RFST_IDLE" check [5].

1. Selection of Scenario 2 must be done, like described in Section 4
2. Send NFCEE_DISCOVER_CMD
3. Send NFCEE_MODE_SET_CMD
4. Open communication channel with CORE_CONN_CREATE_CMD

Codeblock below shows MCUXpresso project, with commands mentioned above:

```
void Configure_NFCEE_NDEF_and_Open_Logical_connection(){
 uint8_t NFCEE_DISCOVER_CMD[] = {0x22, 0x00, 0x00};
 uint8_t NFCEE_MODE_SET_CMD[] = {0x22, 0x01, 0x02, 0x10, 0x01};
 uint8_t CORE_CONN_CREATE_CMD[] = {0x20, 0x04, 0x06, 0x03, 0x01, 0x01, 0x02, 0x10, 0x00};

 uint8_t Answer[MAX_NCI_FRAME_SIZE];
 uint16_t AnswerSize;

 NxpNci_HostTransceive(NFCEE_DISCOVER_CMD, sizeof(NFCEE_DISCOVER_CMD), Answer,
 sizeof(Answer), &AnswerSize);
 NxpNci_WaitForReception(Answer, sizeof(Answer), &AnswerSize, TIMEOUT_INFINITE);

 NxpNci_HostTransceive(NFCEE_MODE_SET_CMD, sizeof(NFCEE_MODE_SET_CMD), Answer,
 sizeof(Answer), &AnswerSize);
 NxpNci_WaitForReception(Answer, sizeof(Answer), &AnswerSize, TIMEOUT_INFINITE);
 NxpNci_WaitForReception(Answer, sizeof(Answer), &AnswerSize, TIMEOUT_INFINITE);

 NxpNci_HostTransceive(CORE_CONN_CREATE_CMD, sizeof(CORE_CONN_CREATE_CMD), Answer,
 sizeof(Answer), &AnswerSize);
 NxpNci_WaitForReception(Answer, sizeof(Answer), &AnswerSize, TIMEOUT_INFINITE);
}
```

After the communication channel is open, there is read and write-access. Command set is explained in [6].

IMPORTANT NOTICE:

After all actions are finished, communication channel must be closed with CORE_CONN_CLOSE_CMD (0x20, 0x05, 0x01, 0x05)!

7.2.2 RF access

For RF access on Android and Linux, the only change is to switch between scenarios in configuration file. In MCUXpresso project three modifications are needed:

1. Switch between scenarios
2. RF_SET_LISTEN_MODE_ROUTING_CMD must be modified inside NxpNci20.c in function NxpNci_ConfigureMode. For more details, check example ([8]) and Section 6.3.
3. Send NFCEE_DISCOVER_CMD and NFCEE_MODE_SET_CMD

After all changes, RF_DISCOVER_CMD can be send and RF access to NFCEE_NDEF is established.

If TagInfo is used, the state machine must be reused, but the DH will always reports "6A 82" (NOK). Check the example for more details.

8 Abbreviations

Table 3.

Abbr.	Meaning
AN	application note
CE	card emulation
DH	device host
DH-NFCEE	NFC Execution Environment running on the DH
FW	firmware
ISO/IEC	International Standard Organization / International Electrotechnical Community
MW	middleware
NCI	NFC controller interface
NFC	near-field communication
NFCC	NFC controller (i.e. PN7160)
RF	radio frequency
RFU	reserved for future use
UM	user manual

9 References

[1] ISO/IEC14443 - 3 standard – Cards and security devices for personal identification — Contactless proximity objects - Part 3 Initialization and anticollision

[2] ISO/IEC14443 - 4 standard – Cards and security devices for personal identification — Contactless proximity objects - Part 4 Transmission protocol

[3] NFC Forum – NFC Controller Interface (NCI) - Version 2.0

[4] NFC Forum – Digital Protocol

[5] UM11495 - PN7160 NFC controller, user manual, available on https://www.nxp.com/products/:PN7160

[6] NFC Forum - Type 4 Tag

[7] Linux CE Emulation on DH - https://github.com/NXPNFCLinux/linux_libnfc-nci_examples/tree/master/ndef-emulation_example

[8] MCUXpresso examples - https://www.nxp.com/downloads/en/software/SW6705.zip

[9] Linux CE example (scenario 2) - https://github.com/NXPNFCLinux/linux_libnfc-nci_examples/tree/master/t4t-ndef-emulation_example

[10] PN7160 Android Porting Guide - https://www.nxp.com/docs/en/application-note/AN13189.pdf

[11] PN7160 Linux Porting Guide - https://www.nxp.com/docs/en/application-note/AN13287.pdf

[12] NXP-NCI2.0 MCUXpresso examples guide - https://www.nxp.com/docs/en/application-note/AN13288.pdf

[13] PN7160 RF Settings guide - https://www.nxp.com/docs/en/application-note/AN13218.pdf

[14] PN7160 Antenna design guide and matching guide - https://www.nxp.com/docs/en/application-note/AN13219.pdf

[15] PN7160 Android13 release - https://github.com/NXPNFCLinux/nxpnfc_android13

10 Legal information

10.1 Definitions

Draft — A draft status on a document indicates that the content is still under internal review and subject to formal approval, which may result in modifications or additions. NXP Semiconductors does not give any representations or warranties as to the accuracy or completeness of information included in a draft version of a document and shall have no liability for the consequences of use of such information.

10.2 Disclaimers

Limited warranty and liability — Information in this document is believed to be accurate and reliable. However, NXP Semiconductors does not give any representations or warranties, expressed or implied, as to the accuracy or completeness of such information and shall have no liability for the consequences of use of such information. NXP Semiconductors takes no responsibility for the content in this document if provided by an information source outside of NXP Semiconductors.

In no event shall NXP Semiconductors be liable for any indirect, incidental, punitive, special or consequential damages (including - without limitation - lost profits, lost savings, business interruption, costs related to the removal or replacement of any products or rework charges) whether or not such damages are based on tort (including negligence), warranty, breach of contract or any other legal theory.

Notwithstanding any damages that customer might incur for any reason whatsoever, NXP Semiconductors' aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the Terms and conditions of commercial sale of NXP Semiconductors.

Right to make changes — NXP Semiconductors reserves the right to make changes to information published in this document, including without limitation specifications and product descriptions, at any time and without notice. This document supersedes and replaces all information supplied prior to the publication hereof.

Suitability for use — NXP Semiconductors products are not designed, authorized or warranted to be suitable for use in life support, life-critical or safety-critical systems or equipment, nor in applications where failure or malfunction of an NXP Semiconductors product can reasonably be expected to result in personal injury, death or severe property or environmental damage. NXP Semiconductors and its suppliers accept no liability for inclusion and/or use of NXP Semiconductors products in such equipment or applications and therefore such inclusion and/or use is at the customer's own risk.

Applications — Applications that are described herein for any of these products are for illustrative purposes only. NXP Semiconductors makes no representation or warranty that such applications will be suitable for the specified use without further testing or modification.

Customers are responsible for the design and operation of their applications and products using NXP Semiconductors products, and NXP Semiconductors accepts no liability for any assistance with applications or customer product design. It is customer's sole responsibility to determine whether the NXP Semiconductors product is suitable and fit for the customer's applications and products planned, as well as for the planned application and use of customer's third party customer(s). Customers should provide appropriate design and operating safeguards to minimize the risks associated with their applications and products.

NXP Semiconductors does not accept any liability related to any default, damage, costs or problem which is based on any weakness or default in the customer's applications or products, or the application or use by customer's third party customer(s). Customer is responsible for doing all necessary testing for the customer's applications and products using NXP Semiconductors products in order to avoid a default of the applications and the products or of the application or use by customer's third party customer(s). NXP does not accept any liability in this respect.

Terms and conditions of commercial sale — NXP Semiconductors products are sold subject to the general terms and conditions of commercial sale, as published at http://www.nxp.com/profile/terms, unless otherwise agreed in a valid written individual agreement. In case an individual agreement is concluded only the terms and conditions of the respective agreement shall apply. NXP Semiconductors hereby expressly objects to applying the customer's general terms and conditions with regard to the purchase of NXP Semiconductors products by customer.

Export control — This document as well as the item(s) described herein may be subject to export control regulations. Export might require a prior authorization from competent authorities.

Evaluation products — This product is provided on an "as is" and "with all faults" basis for evaluation purposes only. NXP Semiconductors, its affiliates and their suppliers expressly disclaim all warranties, whether express, implied or statutory, including but not limited to the implied warranties of non-infringement, merchantability and fitness for a particular purpose. The entire risk as to the quality, or arising out of the use or performance, of this product remains with customer.

In no event shall NXP Semiconductors, its affiliates or their suppliers be liable to customer for any special, indirect, consequential, punitive or incidental damages (including without limitation damages for loss of business, business interruption, loss of use, loss of data or information, and the like) arising out the use of or inability to use the product, whether or not based on tort (including negligence), strict liability, breach of contract, breach of warranty or any other theory, even if advised of the possibility of such damages.

Notwithstanding any damages that customer might incur for any reason whatsoever (including without limitation, all damages referenced above and all direct or general damages), the entire liability of NXP Semiconductors, its affiliates and their suppliers and customer's exclusive remedy for all of the foregoing shall be limited to actual damages incurred by customer based on reasonable reliance up to the greater of the amount actually paid by customer for the product or five dollars (US$5.00). The foregoing limitations, exclusions and disclaimers shall apply to the maximum extent permitted by applicable law, even if any remedy fails of its essential purpose.

Translations — A non-English (translated) version of a document, including the legal information in that document, is for reference only. The English version shall prevail in case of any discrepancy between the translated and English versions.

Security — Customer understands that all NXP products may be subject to unidentified vulnerabilities or may support established security standards or specifications with known limitations. Customer is responsible for the design and operation of its applications and products throughout their lifecycles to reduce the effect of these vulnerabilities on customer's applications and products. Customer's responsibility also extends to other open and/or proprietary technologies supported by NXP products for use in customer's applications. NXP accepts no liability for any vulnerability. Customer should regularly check security updates from NXP and follow up appropriately.

Customer shall select products with security features that best meet rules, regulations, and standards of the intended application and make the ultimate design decisions regarding its products and is solely responsible for compliance with all legal, regulatory, and security related requirements concerning its products, regardless of any information or support that may be provided by NXP.

NXP has a Product Security Incident Response Team (PSIRT) (reachable at PSIRT@nxp.com) that manages the investigation, reporting, and solution release to security vulnerabilities of NXP products.

NXP B.V. - NXP B.V. is not an operating company and it does not distribute or sell products.

10.3 Licenses

Purchase of NXP ICs with NFC technology — Purchase of an NXP Semiconductors IC that complies with one of the Near Field Communication (NFC) standards ISO/IEC 18092 and ISO/IEC 21481 does not convey an implied license under any patent right infringed by implementation of any of those standards. Purchase of NXP Semiconductors IC does not include a license to any NXP patent (or other IP right) covering combinations of those products with other products, whether hardware or software.

10.4 Trademarks

Notice: All referenced brands, product names, service names, and trademarks are the property of their respective owners.

NXP — wordmark and logo are trademarks of NXP B.V.

EdgeVerse — is a trademark of NXP B.V.

Tables

Figures

Contents

For more information, please visit: http://www.nxp.com

Date of release: 26 April 2023
Document identifier: AN13861