# UM11810

## PN722X NFC controller

**Rev. 1 — 2 February 2024** **User manual**

**Document information**

| Information | Content |
|---|---|
| Keywords | PN722X NFCC, NFC, NFCC, NCI |
| Abstract | This is a user manual for the PN722X NFCC IC. The aim of this document is to describe the PN722X NFCC interfaces, modes of operation and possible configurations (NFCC view). |

# 1 Introduction

The PN722X is a NFC controller product that provides the most advanced generations of NXP Semiconductors NFC controller. The PN722X family is targeting Android based designs for PAYMENT, POS & mPOS TERMINAL applications.

The PN722X solution defines the state of the art in terms of NFC superior RF performance with support for both NFC Forum and EMVCo Mode with high RF output (up to 2.0 W) and high receiver sensitivity is a robust solution for payment terminals. This NFC controller allows to achieve EMVCo 3.x analog compliancy even in metal environment and for through-display applications.

As an product with NCI 2.2 interface, the PN7220 supports a secure switching between NFC Forum and EMVCo modes, allowing the fast development of terminals which need to support NFC Forum applications as well as EMVCo payment. A dedicated pre-configuration of the RF settings for each EMVCo and NFC Forum mode simplifies the design and takes care of the dedicated RF requirements of each mode without the need of frequent re-configurations of the RF functionality from a host. The internal processing of the device had been optimized for fast NCI data transfer with low latency, to meet EMVCo application timing requirements.

The PN722X is supported by an NCI 2.2 compliant middleware which allows to achieve both EMVCo3.x digital compliancy as well as NFC Forum compliancy. PN722X products provide support to connect to contact interface using TDA8035 which is the cost efficient integrated contact smart card reader IC.

Additionally, the PN7221 NFCC supports all features of PN7220 plus "Enhanced Contactless Polling" (ECP) by Apple. Note, that the ECP feature is available after formal authorization from Apple only.

The user manual describes the software interfaces (API), based on the NFC Forum standard, NCI, referencing the version 2.2.

As this document assumes pre-knowledge on certain technologies, check Section 3 to find the appropriate documentation.

For further information, refer to the PN722X data sheet [PN722X_DS].

# 2 Abbreviations

**Table 1. Abbreviations**

| Abbreviation | Definition |
|---|---|
| CE | Card emulation |
| CITO | controller input / target output (former MISO) |
| COTI | controller output / target input (former MOSI) |
| CLIF | contactless interface |
| COTS | commercially of the self |
| CPoC | contactless payment on COTS |
| CT | contact |
| DH | device host |
| DH-NFCEE | NFC execution environment running on the DH |
| DPC | dynamic power control |
| GID | group identifier |
| HDLL | host data link layer |
| LPCD | low power card detector |
| MT | message type |
| NCI | NFC controller interface |
| NFC | near field communication |
| NFCC | NFC controller |
| NFCEE | NFC execution environment |
| OID | opcode identifier |
| PRBS | pseudo random binary sequence |
| PBF | packet boundary flag |
| RF | radio frequency |
| RFU | reserved for future use |
| RSSI | received signal strength indication |

# 3 References

PN722X refers to PN7220

[PN722X_DS] PN7220 data sheet

[NCI] NFC Forum NFC controller Interface, version 2.2

[NCI_Table1] Status Codes table: table 129 in [NCI]

[NCI_Table2] RF technologies table: table 130 in [NCI]

[NCI_Table3] RF Technology and Mode table: table 131 in [NCI]

[NCI_Table4] Bit Rates table: table 132 in [NCI]

[NCI_Table5] RF protocols table: table 133 in [NCI]

[NCI_Table6] RF Interfaces table: table 134 in [NCI]

[NCI_Table8] Config. Parameters table: table 138 in [NCI]

[NCI_Table9] CORE_RESET_NTF table: table 5 in [NCI]

[NCI_Chap2] State Machine: chapter 5.2 in [NCI]

[DIGITAL] NFC Forum Digital Protocol Specification version 2.3

[ACTIVITY] NFC Forum Activity Specification version 2.2

[7816-4] ISO/IEC 7816-4

[14443-4] ISO/IEC 14443-4 2018-07

CTS configuration part of NFC Cockpit, https://www.nxp.com/products/:NFC-COCKPIT

# 4 PN722X architecture overview

The PN722X NFC controller simplified architecture view is depicted in Figure 1:

- The top part describes the Device Host architecture with Higher Layer Driver (i.e. Middleware stack) hosting:
  - the user applications (Reader/Writer, Card Emulation in the DH-NFCEE)
  - the NCI driver
  - and the transport layer driver.
- In addition to DH-NFCEE, PN722X supports NFCEEs for CT support.
- The middle part describes PN722X. PN722X is connected to the DH through an $I^2C$ physical interface. The PN722X NFCC firmware supports the NCI specification 2.2. PN722X supports NCI specification [NCI] and additional extensions. DH uses the NFCEE Interface to encapsulate the ISO 7816-4 APDU and sent to NFCC, allowing the communication with Contact Interface card connected over TDA 8035.
- The Bottom part of the figure contains the RF antenna connected to the PN722X NFCC, which can communicate over RF with a tag (card), a reader/writer or another NFC device.
  The PN722X NFCC firmware is stored both in ROM and in FLASH: This means that it can be updated when required. The PN722X NFCC User Data is stored in RAM and in FLASH. The data stored in FLASH can also be updated, after updating the data stored in FLASH, a POR is required to use these updated data.
  NXP provides a download mechanism for the binary content of the firmware stored in FLASH and the user data.
  All PN722X variants work in two types of Device Host configuration:
  **Single host device (HD) configuration:**
- HIF1-I2C interface connects PN722X to the device host. For contactless operation, NFC Forum and EMVCo modes of operation are supported. The host device controls the PN722X mode of operation by toggling Mode_Switch_GPI as described in Table 8.
  **Dual host device configuration:**
- PN722X NFCC can work in multi-host configuration, with a main and a target host. The main host acts as controller.
- For contactless operation, the main host supports NFC Forum mode, and the target host supports EMVCo mode.
- The main host device controls PN722X mode of operation, and the target host EMVCo mode of operation. Once the EMVCo operation is completed, the Main Host takes back the control to work in NFC Forum mode of operation.

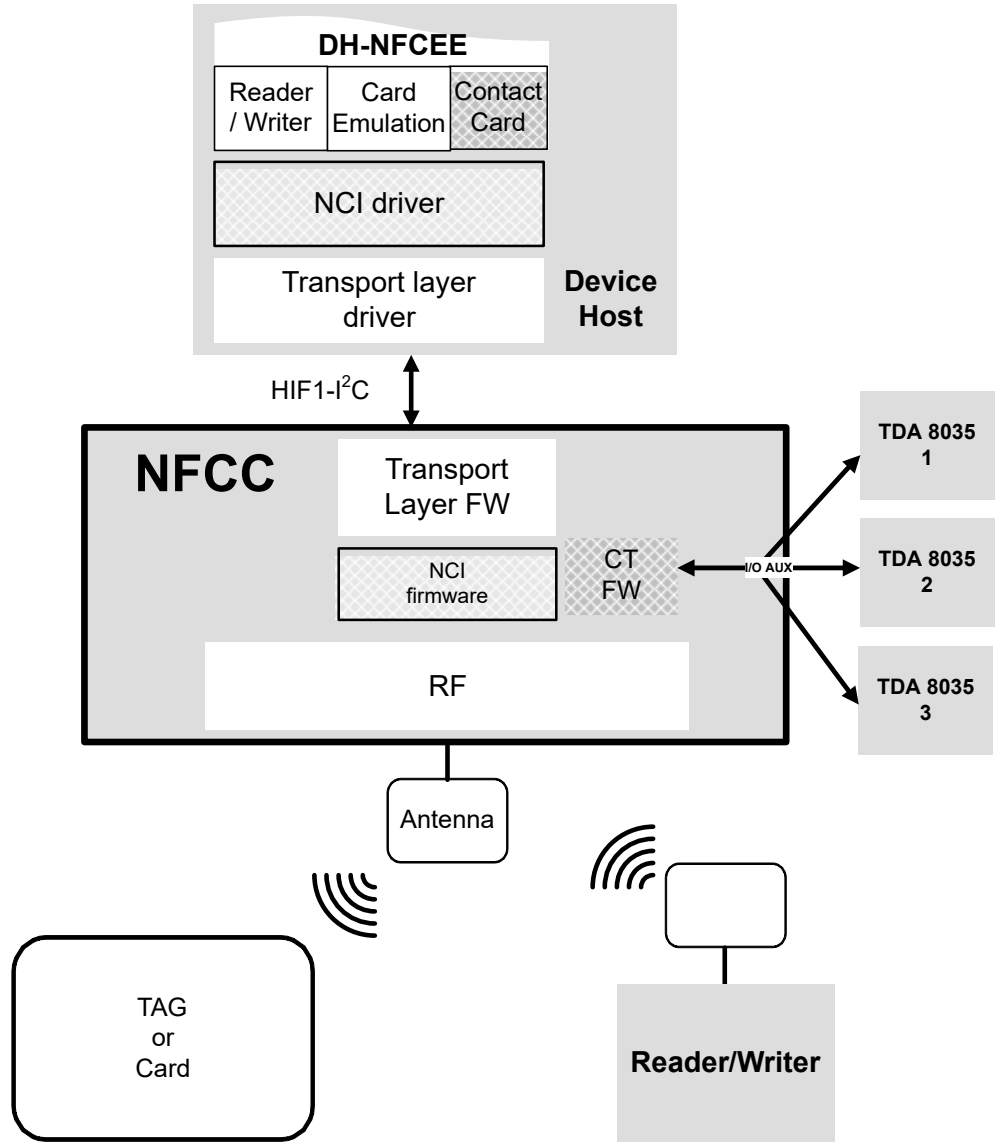



**Figure 1. PN722X system architecture with Single Host**

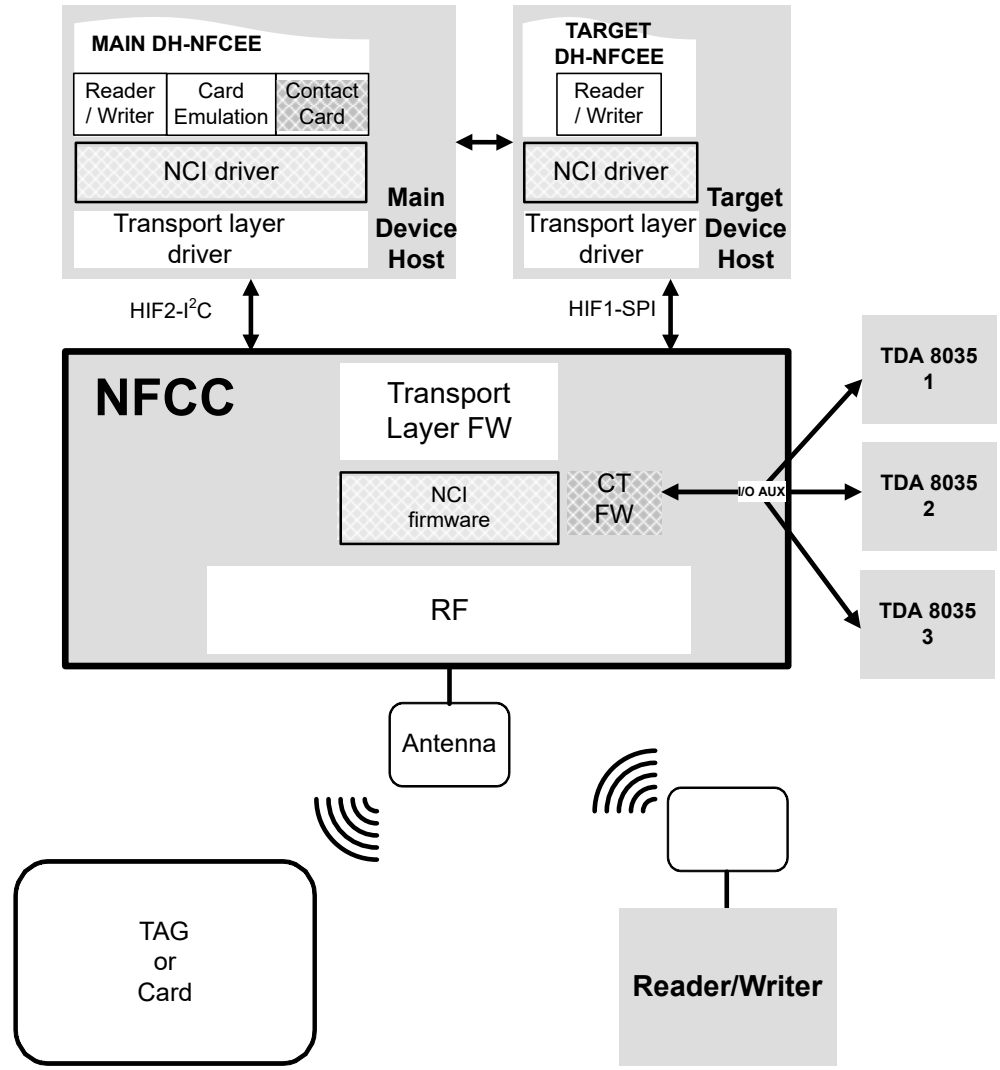



**Figure 2. PN722X system architecture with dual host**

## 4.1 Reader/writer operation in poll mode

### 4.1.1 Reader/writer hosted by the Main DH

The Reader/Writer application running on the Main DH is accessing a remote contactless Tag/Card, through the PN722X NFCC.

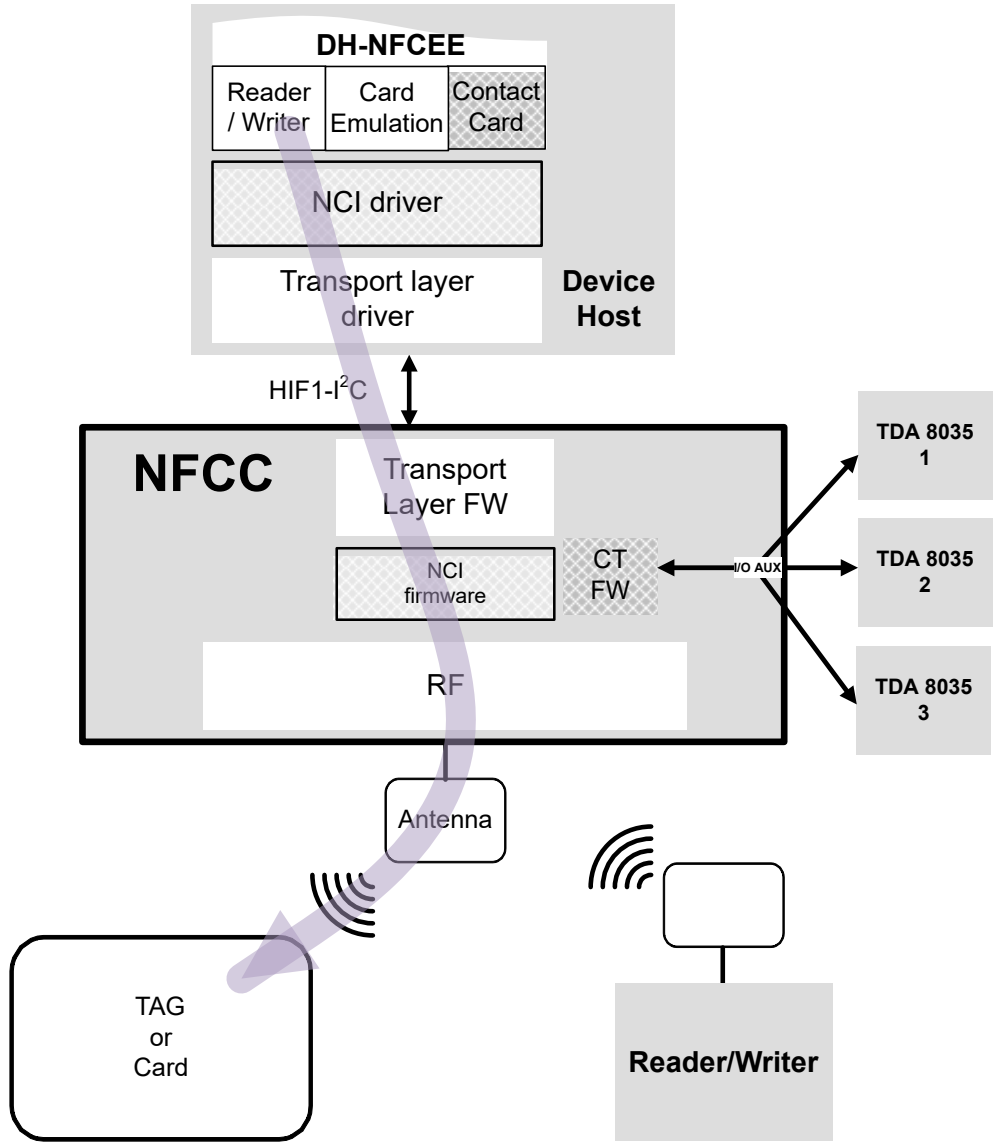



**Figure 3. Reader/Writer hosted by the Main DH**

### 4.1.2 Reader/Writer hosted by the Target [2] DH

The Reader/Writer application running on the Target DH is accessing a remote contactless Tag/Card, through the PN722X NFCC.

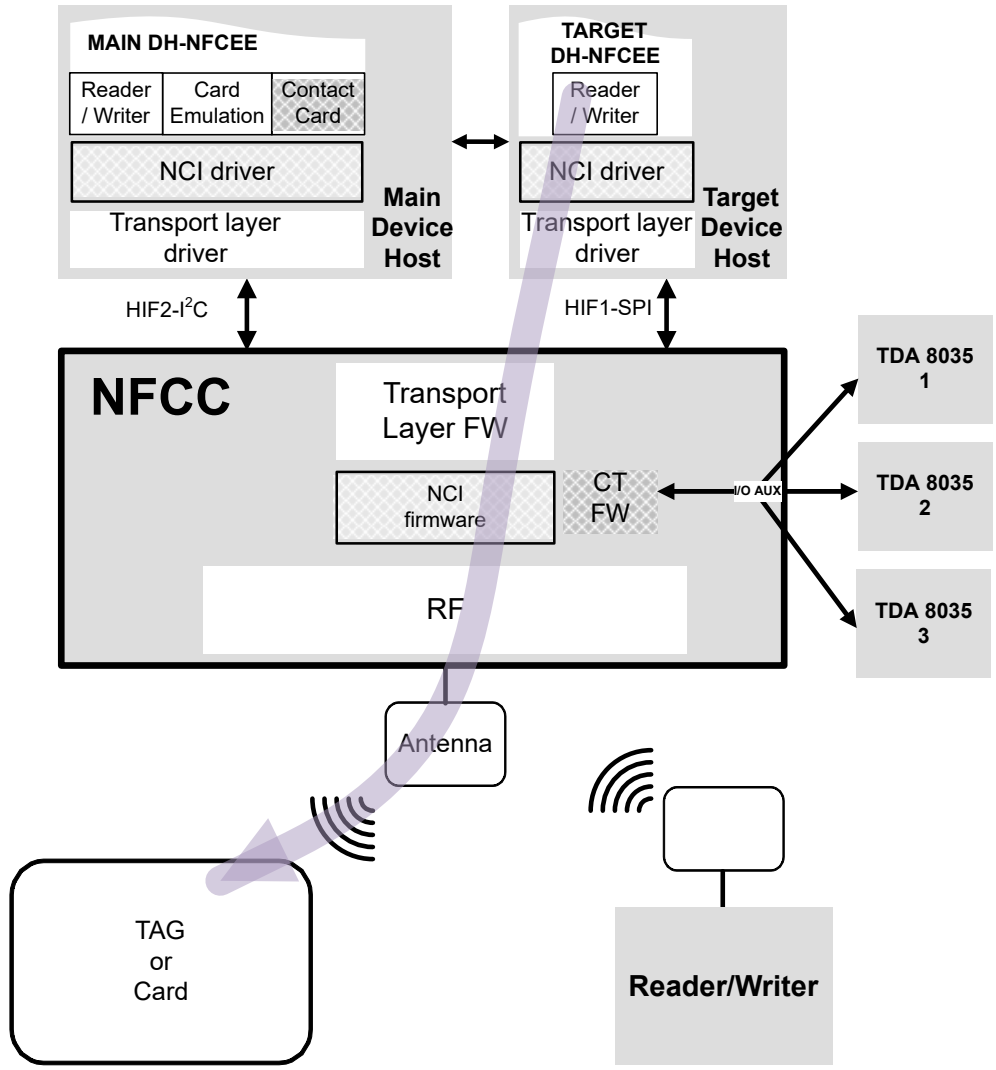



**Figure 4. Reader/Writer hosted by the Target DH**

2 The master/slave replacement into controller/target in this document follows the recommendation of the NXP - I2C standards organization.

## 4.2 Card emulation operation in listen mode

### 4.2.1 Card emulated by the Main DH-NFCEE

An external Reader/Writer accesses the Main DH-NFCEE by emulating a contactless card through the PN722X NFCC. PN7220 and PN7221 do not support Card emulation.

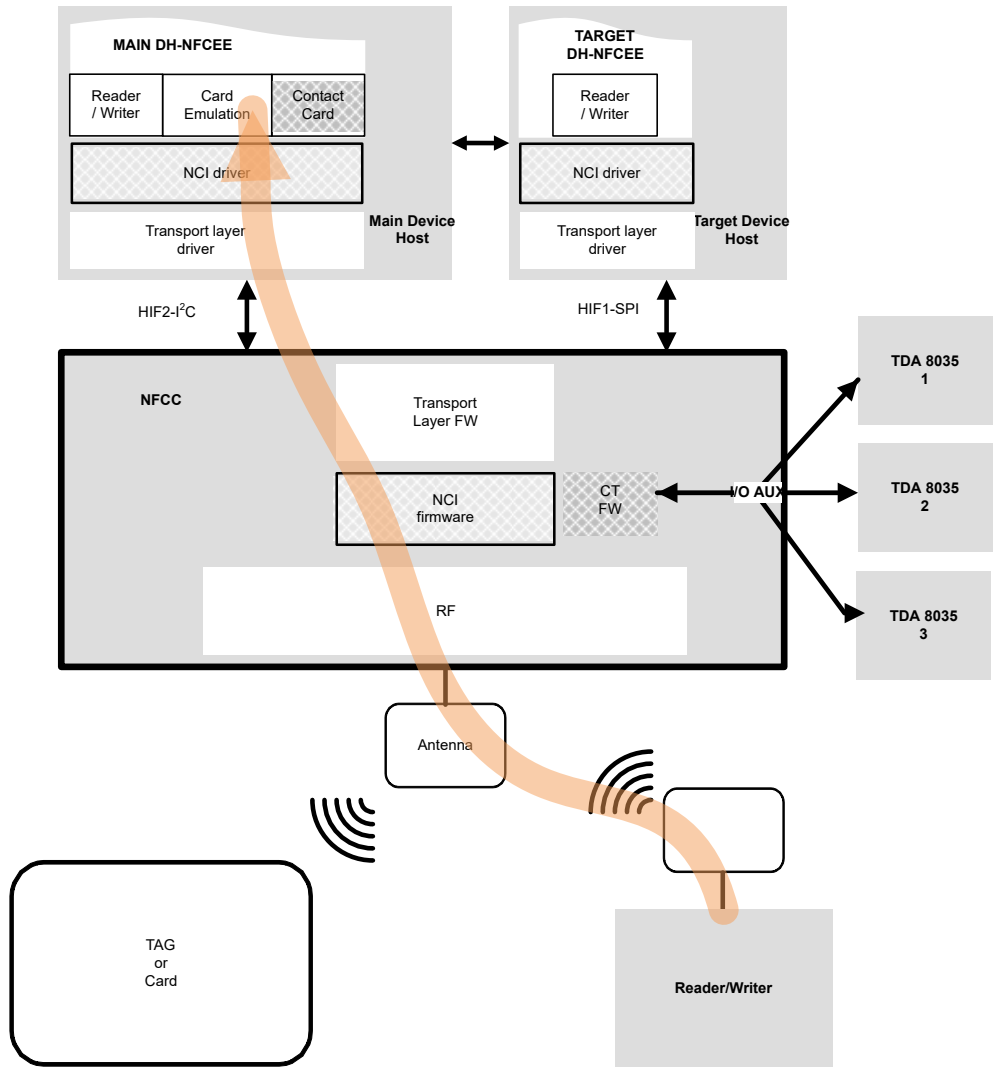



**Figure 5. Card emulated by the Main DH-NFCEE**

## 4.3 Combined modes of operation

The PN722X NFCC firmware combines the basic modes of operation described in Section 4.1 and Section 4.2, using the RF Discovery as defined in [NCI]. Since the PN722X NFCC offers more features than currently addressed in [NCI], NXP has defined some proprietary extensions.

The principle used to combine the various modes of operation is to build a cyclic activity, which sequentially activates various modes of operation:

1. Start a Polling sequence, to look for a remote Tag/Card. If several technologies are enabled by the DH, PN722X NFCC polls sequentially for all the enabled technologies.
2. If nothing was detected, enter a Listening sequence, to potentially be activated as a Card emulator by an external Reader/Writer.
3. If nothing happens after a programmable timeout, switch back to Poll Mode in step 1.

Figure 6 illustrates the cyclic activity specified in [ACTIVITY], where technologies NFC-A, NFC-B, NFC-F and NFC-V have been activated in Poll Mode.

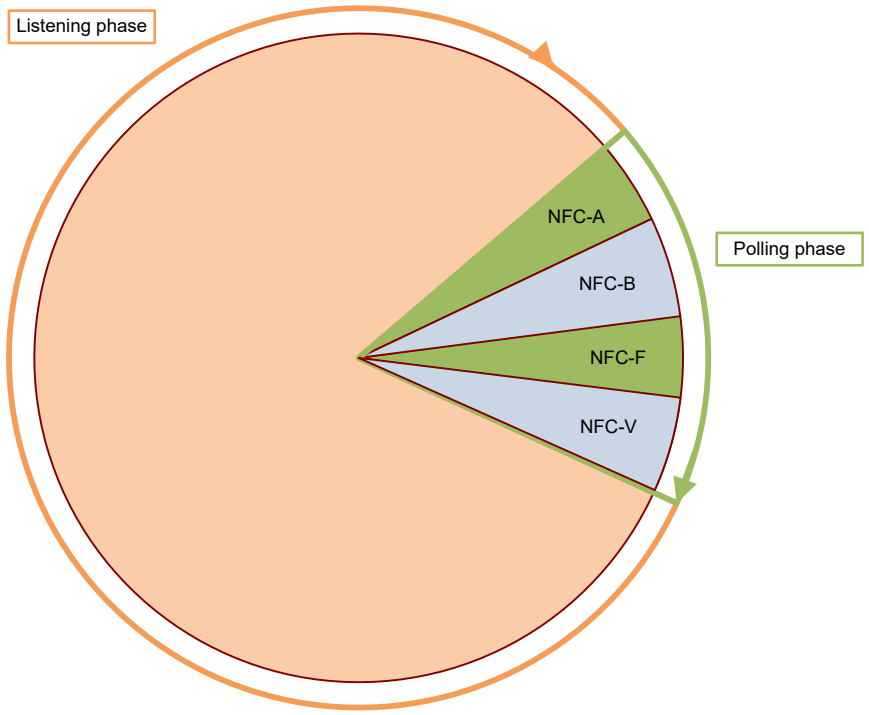



**Figure 6. RF discovery NFC Forum profile**

***Note:*** *When the PN722X NFCC is polling in Reader/Writer operation, it can consume a significant amount of current (depending on the antenna characteristics: >100mA). This applies for the four polling phases shown in Figure 6, and is because the PN722X NFCC has to generate the RF carrier (13.56 MHz). But during the Listen phase, the PN722X NFCC current consumption is reduced to few 10's of µA, since PN722X is waiting for the detection of an externally generated RF carrier (standby power mode).*

Figure 7 illustrates the cyclic RF Discovery where polling is enabled only for NFC-A and NFC-B, for simplicity.

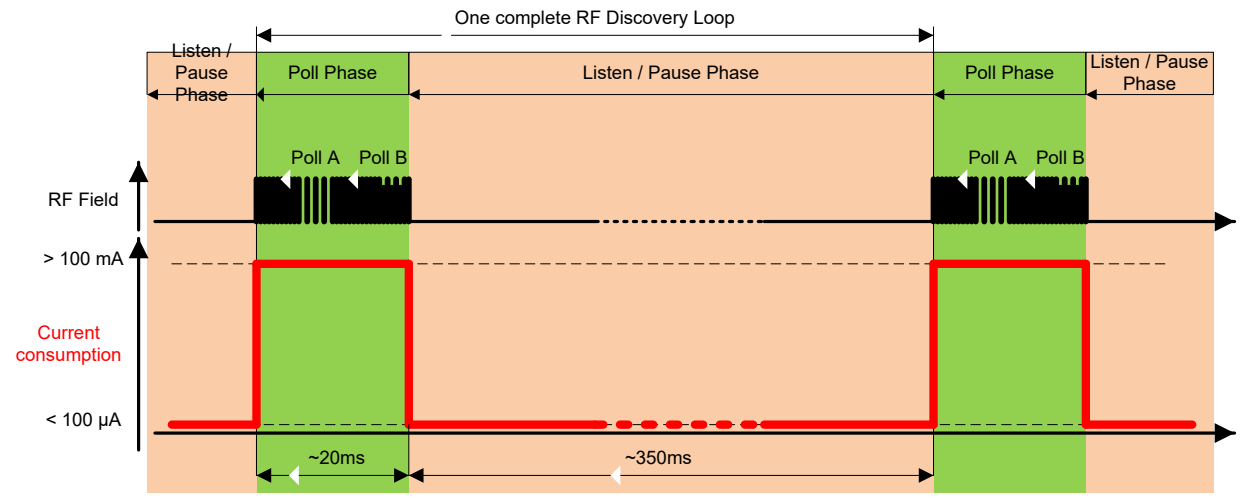



**Figure 7. Power consumption during RF discovery**

In a typical set-up, the polling phase is approximately 50 ms long while the listening phase is approximately 350 ms long (this set-up is configured thanks to the NCI parameter called TOTAL_DURATION).

This average consumption can be further optimized, using PN722X NFCC Low-Power Card Detector (LPCD) feature.

***Note:*** *[NCI] defines the TOTAL_DURATION of the discovery period independently of the reader phases applied. To simplify the implementation for the PN722X NFCC, a timer is applied only during the Listen/pause phase. So, depending on the polling phase configuration (one technology or more), the total duration varies.*

For further details on the RF discovery activity, refer to Section 12.

## 4.4 DH access to CT Card over TDA

CT support with PN722X uses NFCEE interface defined by NCI Specification.

NXP commonly names this mode of communication the "CT mode", since the communication is to interact with Contact Cards (for example SAM, Contact Payment Card). FIGURE illustrates the data exchange flow between the DH and the CT Card over NFCC via TDA. This mode is useful for two different use cases:

- The application running on the DH wants to interact with a SAM connected over TDA to perform Secure Key Store or Crypto Operations. A typical example is a DH application that encrypts the Data using Crypto engine on SAM and then communicates the data when a contactless payment takes place.
- The application running on the DH wants to accept any Contact Payment Card that can be inserted dynamically and perform Payment transaction.

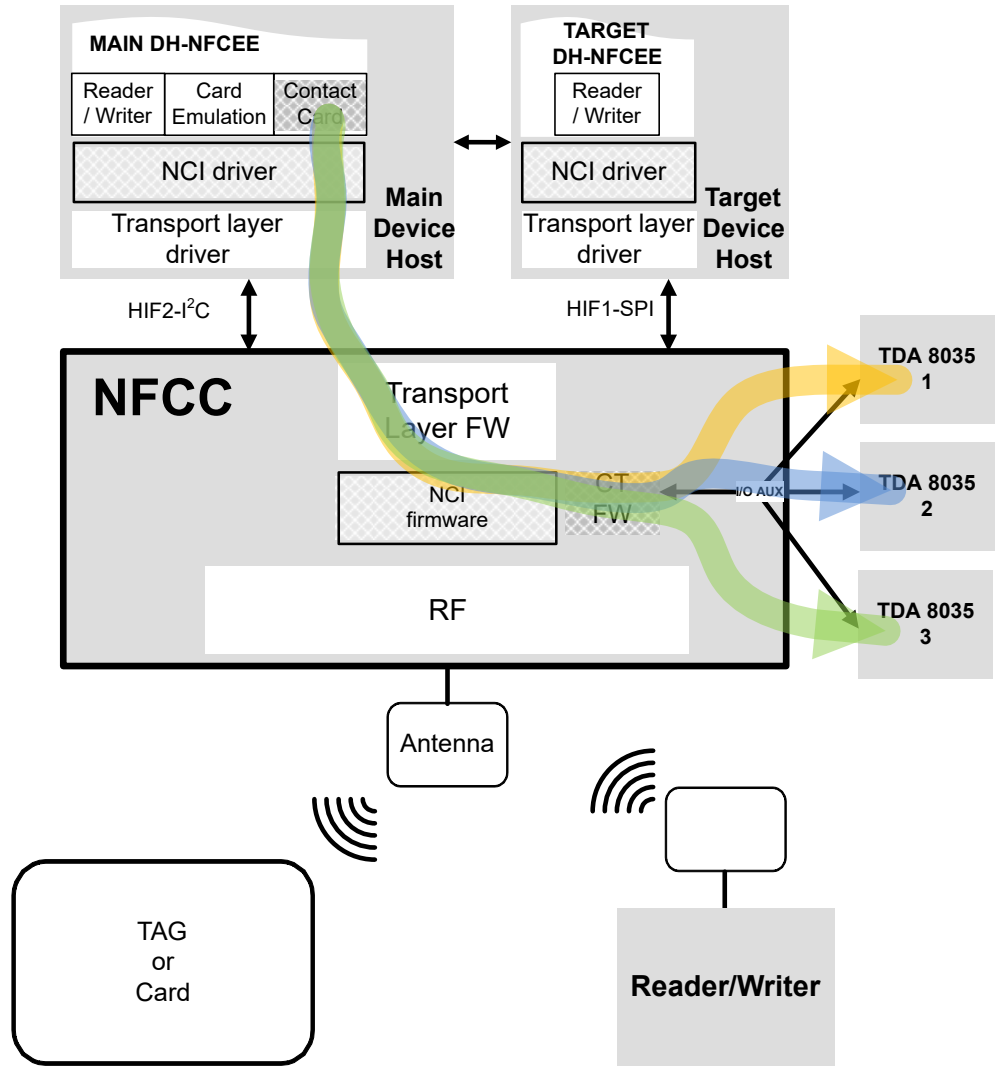



**Figure 8. Exchanges between the DH and the CT Card over TDA**

To start CT activation for the two use cases, the DH should send `NFCEE_DISCOVER_CMD` to check the presence of any CT Card, and notify NFCC with `NFCEE_DISCOVER_NTF`. The NFCC does not check any activity related to card presence and/or card removal without `NFCEE_DISCOVER_NTF` from the DH.

The first use case works when SAM is connected before DH boot process and during boot. SAM cards are initialized and the information is conveyed as notifications to DH upon receiving NFCEE_DISCOVER_CMD.

In the second use case, the contact payment card can be inserted dynamically. After the activation and identification of payment card, the PN722X sends an event to the DH only upon receiving NFCEE_DISCOVER_CMD.

For both use cases, the general mechanism used to communicate with the CT Cards over TDA from the DH is an ISO7816-4 APDU wrapped using a proprietary layer running on the DH on top of NCI Transport layer. DH needs to generate ISO7816-4 APDU packets, which are then encapsulated in NCI data packets and forwarded to the NFCC. The NFCC forwards the ISO7816-4 APDU packets to the appropriate CT Card over TDA. A similar mechanism is used for the communication from a CT Card to the DH.

For further details on the CT Card Management, refer to Section 8.

# 5 NCI overview

The section gives an overview of the key points of the [NCI] specification.

## 5.1 NCI components

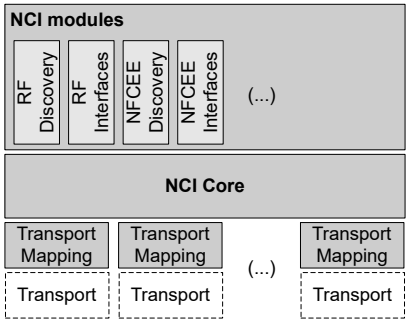



**Figure 9. NCI components**

### 5.1.1 NCI modules

NCI modules are built on top of the functionality provided by the NCI Core. Each module provides a well-defined functionality to the DH. NCI modules provide the functionality to configure the NFCC and to discover and communicate with Remote NFC Endpoints or with local NFCEEs.

Some NCI modules are mandatory parts of an NCI implementation, others are optional. There can also be dependencies between NCI modules in the sense that a module may only be useful if there are other modules implemented as well. For example all modules that deal with communication with a Remote NFC Endpoint (the RF Interface modules) depend on the RF Discovery to be present.

### 5.1.2 NCI core

The NCI core defines the basic functionality of the communication between a Device Host (DH) and an NFC controller (NFCC). This enables Control Message (Command, Response, and Notification) and Data Message exchange between an NFCC and a DH.

### 5.1.3 Transport mappings

Transport mappings define how the NCI messaging is mapped to an underlying NCI transport, which is a physical connection (and optional associated protocol) between the DH and the NFCC. Each transport mapping is associated with a specific NCI transport.

## 5.2 NCI concepts

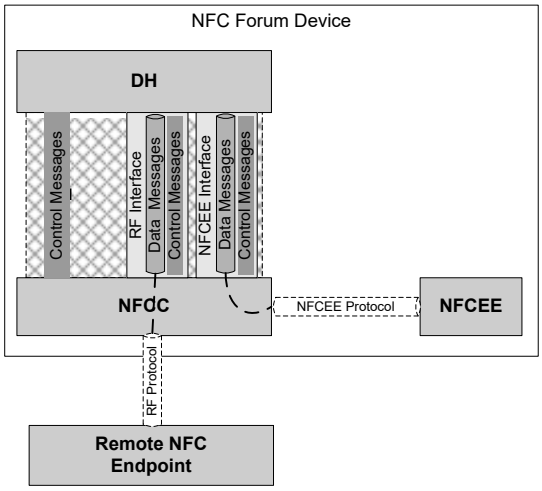



**Figure 10. NCI concepts**

### 5.2.1 Control messages

A DH uses NCI Control messages to control and configure an NFCC. Control Messages consist of Commands, Responses, and Notifications. Commands are only allowed to be sent in the direction from DH to NFCC, Responses and Notifications are only allowed in the other direction. Control Messages are transmitted in NCI Control Packets, NCI supports segmentation of Control Messages into multiple Packets.

The NCI Core defines a basic set of Control Messages, for example to set and retrieve NFCC configuration parameters. NCI Modules can define additional Control Messages.

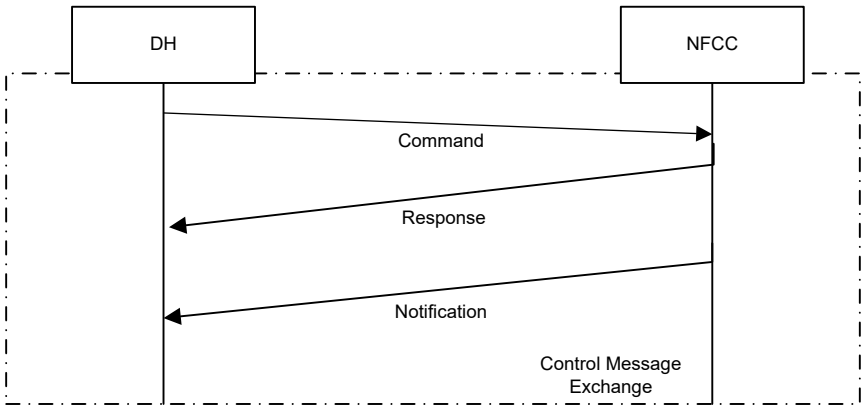



**Figure 11. Control Message Exchange**

### 5.2.2 Data messages

Data messages are used to transport data to either a Remote NFC Endpoint (named RF Communication in NCI) or to an NFCEE (named NFCEE Communication). NCI defines Data Packets enabling the segmentation of Data Messages into multiple Packets.

Data Messages can only be exchanged in the context of a Logical Connection. As a result, a Logical Connection must be established before any Data Message is sent. One Logical Connection, the Static RF Connection, is always established during initialization of NCI. The Static RF Connection is dedicated for RF Communication. Additional Logical Connections can be created for RF and/or NFCEE Communication.

Since [NCI2.2], another logical connection, the Static HCI Connection, is always established during initialization of NCI. But PN722X does not create this Static HCI Connection for NFCEE communicating to CT device.

Logical Connections provide the flow control for Data Messages from DH to NFCC (Figure 12).

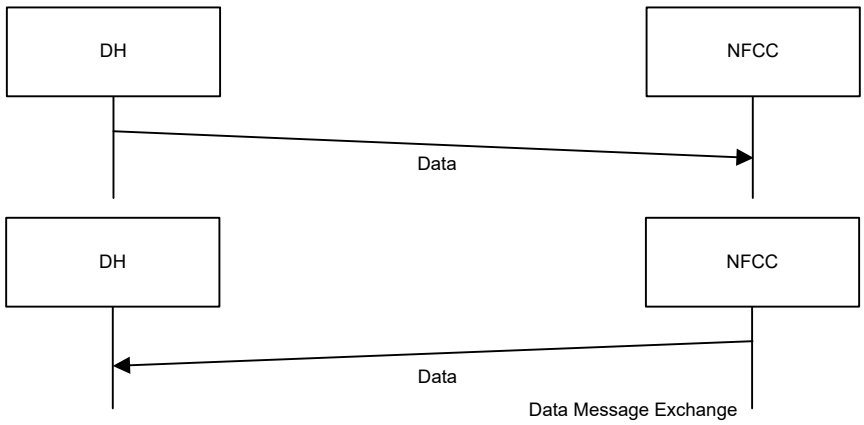



**Figure 12. Data message exchange**

### 5.2.3 Interfaces

An NCI module might contain a single Interface. Each Interface defines how a DH can communicate via NCI with a Remote NFC Endpoint or NFCEE. Each Interface is defined to support specific protocols and can only be used for those protocols (most Interfaces support exactly one protocol). NCI defines two types of Interfaces: RF Interfaces and NFCEE Interfaces.

Protocols that are used to communicate with a Remote NFC Endpoint are called RF Protocols. Protocols that are used to communicate with an NFCEE are called NFCEE Protocols.

An NFCEE Interface has a one-to-one relationship to an NFCEE Protocol. However, there might be multiple RF Interfaces for one RF Protocol. The multiple RF Interfaces allow NCI to support different splits of the protocol implementation between the NFCC and DH. An NCI implementation on an NFCC includes the RF Interfaces that match the functionality implemented on the NFCC.

Interfaces need to be activated before they can be used and deactivated when they are no longer used.

An Interface can define its own configuration parameters and Control Messages. But, most important, it defines both the mapping of the Data Message to the payload of the respective RF or NFCEE Protocol and, in the case of RF Communication, whether the Static RF Connection and/or Dynamic Logical Connections are used to exchange those Data Messages between the DH and the NFCC.

### 5.2.4 RF interface extensions

An RF Interface Extension adds a specific, clearly defined set of functions to one or more RF Interfaces. Each RF Interface Extension defines which RF Interfaces it can extend.

The availability of an RF Interface Extension is bound to the time at which one of these RF Interfaces is activated. The availability can additionally depend on other conditions, e.g. the protocol being currently used in the NFC RF Communication. If an RF Interface is active, available RF Interface Extensions can be started and stopped by the DH. RF Interface Extensions are never started automatically but are stopped automatically when the RF Interface is deactivated. Each RF Interface Extension defines the Control Messages for starting and stopping its functionality.

While started, an RF Interface Extension can overrule definitions of the active RF Interface to provide its own functionality. RF Interface Extensions can define the interface's own configuration parameters and Control Messages. They can also provide a format for Data Messages that is different from the one defined by the RF Interface. However, RF Interface Extensions cannot overrule the deactivation behavior of the active Interface.

For a given RF Interface there can be multiple RF Interface Extensions. It is possible to start multiple RF Interface Extensions at the same time, as long as they are not mutually exclusive. Each RF Interface Extension defines which other RF Interface Extension it can be used with.

However, the extension process cannot be compounded: RF Interface Extensions are not intended to extend the functionality of other RF Interface Extensions.

### 5.2.5 RF communication

RF Communication is started by configuring and running the RF Discovery process. The RF Discovery is an NCI module that discovers and enumerates Remote NFC Endpoints.

For each Remote NFC Endpoint, the RF Discovery Process provides the DH with the Remote NFC Endpoint information that was gathered during the RF Discovery Process. This information includes the RF Protocol that is used to communicate with the Remote NFC Endpoint. During RF discovery configuration the DH sets up a mapping that associates an RF Interface for each RF Protocol. If only a single Remote NFC Endpoint is detected during a discovery cycle, the RF Interface for this Endpoint is automatically activated. If there are multiple Remote NFC Endpoints detected in Poll Mode, the DH can select the Endpoint it wants to communicate with. This selection also triggers activation of the mapped Interface.

After an RF Interface has been activated, the DH can communicate with the Remote NFC Endpoint using the activated RF Interface. An activated RF Interface can be deactivated by either the DH or the NFCC (e.g., on behalf of the Remote NFC Endpoint). However, each RF Interface can define which of those methods are allowed. The deactivation options vary, depending on which part of the protocol stack is executed on the DH. For example, if a protocol command to tear down the communication is handled on the DH, the DH will deactivate the RF Interface. If such a command is handled on the NFCC, the NFCC will deactivate the Interface.

This specification describes the possible Control Message sequences for RF Communication in the form of a state machine.

### 5.2.6 NFCEE communication

The DH can learn about the NFCEEs connected to the NFCC by employing the NFCEE Discovery module. During NFCEE Discovery, the NFCC assigns an identifier for each NFCEE. When the DH wants to communicate with an NFCEE, it opens a Logical Connection to the NFCEE that includes the corresponding identifier and specifies the NFCEE Protocol to be used.

Opening a Logical Connection to an NFCEE automatically activates the NFCEE Interface associated with the protocol specified. As there is always a one-to-one relationship between an NFCEE Protocol and Interface, there is no mapping step required (unlike RF Interface activation).

After the Interface has been activated, the DH can communicate with the NFCEE using the activated Interface.

Closing the connection to an NFCEE Interface deactivates the NFCEE Interface.

NCI also includes functionality to allow the DH to enable or disable the communication between an NFCEE and the NFCC.

### 5.2.7 Identifiers

NCI uses different identifiers for Remote NFC Endpoints and NFCEEs. These identifiers are dynamically assigned by the NFCC. The DH learns them in the contexts of RF Discovery and NFCEE Discovery. The identifiers for Remote NFC Endpoints are called RF Discovery IDs. They usually have a short lifetime as they are only valid for the time the DH wants to be able to communicate with the Remote NFC Endpoint. In contrast, the identifiers for NFCEEs have a longer lifetime, since NFCEEs usually are not frequently added to or removed from an NFC Forum Device. The identifiers for NFCEEs are called “NFCEE IDs”. There is one reserved and static NFCEE ID, value 0, which represents the DH-NFCEE.

Logical Connections take a third type of identifier, Destination Type, as a first parameter to identify the destination for the data. Depending on the Destination Type, there can be a second parameter for identifying the data destination. For example, if the Destination Type is ‘Remote NFC Endpoint’, the second parameter will be an RF Discovery ID.

## 5.3 NCI packet format

### 5.3.1 Common packet header

All packets have a common header, consisting of an MT field and a Packet Boundary Flag (PBF) field:

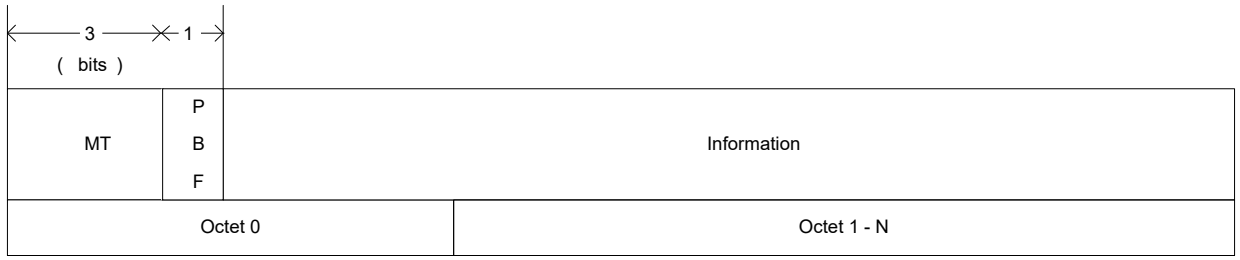



**Figure 13. NCI core packet format**

- **Message Type (MT)**

The MT field indicates the contents of the Packet and SHALL be a 3-bit field containing one of the values listed in Table 1, below. The content of the Information field is dependent on the value of the MT field. The receiver of an MT designated as RFU SHALL silently discard the packet.

**Table 2. MT values**

| MT | Description |
|---|---|
| 000b | Data Packet |
| 001b | Control Packet - Command Message as a payload |
| 010b | Control Packet - Response Message as a payload |
| 011b | Control Packet – Notification Message as a payload |
| 100b-111b | RFU |

- **Packet Boundary Flag (PBF)**

The Packet Boundary Flag (PBF) is used for Segmentation and Reassembly and SHALL be a 1-bit field containing one of the values listed in [NCI] specification.

**Table 3. PBF Value**

| PBF | Description |
|---|---|
| 0b | The Packet contains a complete Message, or the Packet contains the last segment of a segmented Message |
| 1b | The Packet contains a segment of a Message which is not the last segment. |

The following rules apply to the PBF flag in Packets:

- If the Packet contains a complete Message, the PBF SHALL be set to 0b.
- If the Packet contains the last segment of a segmented Message, the PBF SHALL be set to 0b.
- If the packet does not contain the last segment of a segmented Message, the PBF SHALL be set to 1b.

### 5.3.2 Control packets

Figure 14 shows the control packet structure.

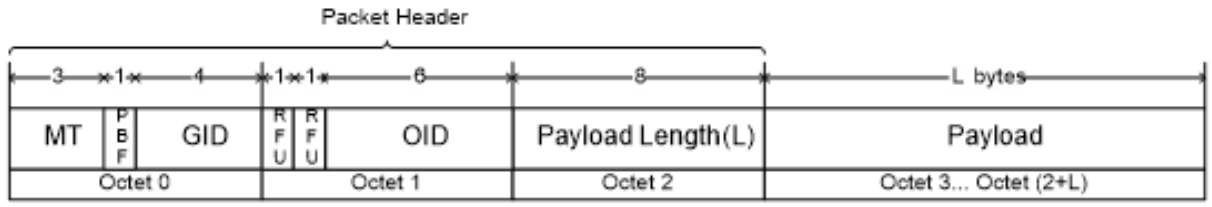



**Figure 14. Control packet format**

Each Control Packet SHALL have a 3 octet Packet Header and MAY have additional payload for carrying a Control Message or a segment of Control Message.

***Note:*** *In the case of an 'empty' Control Message, only the Packet Header is sent.*

- **Message Type (MT)**

Refer to Section 5.3.1 for details of the MT field.

- **Packet Boundary Flag (PBF)**

Refer to Section 5.3.1 for details of the PBF field.

- **Group Identifier (GID)**

NCI supports Commands, Responses, and Notifications which are categorized according to their individual groups. The Group Identifier (GID) indicates the categorization of the message and SHALL be a 4-bit field containing one of the values listed in [NCI] specification.

All GID values not defined in [NCI] specification are RFU.

- **Opcode Identifier (OID)**

The opcode Identifier (OID) indicates the identification of the Control Message and SHALL be a 6-bit field which is a unique identification of a set of Command, Response, or Notification Messages within the group (GID). OID values are defined along with the definition of the respective Control Messages described in [NCI] specification.

- **Payload Length (L)**

The Payload Length SHALL indicate the number of octets present in the payload. The Payload Length field SHALL be an 8-bit field containing a value from 0 to 255.

### 5.3.3 Data packets

Figure 15 shows the data packet structure.

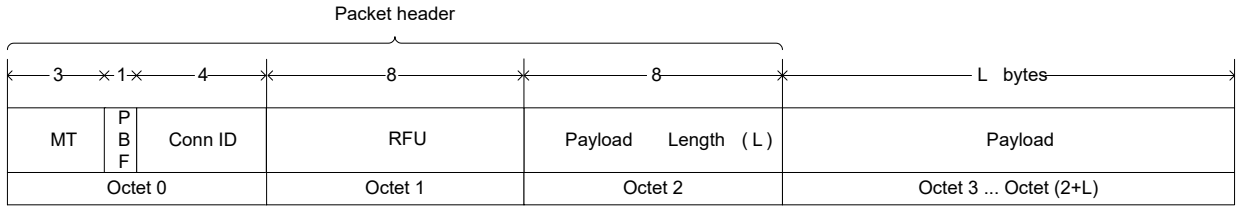



**Figure 15. Data packet structure**

Each Data Packet has a 3 octet Packet Header and may have an additional Payload to carry a Data Message or a segment of a Data Message.

***Note:*** *If a Data Message is empty, only the Packet Header is sent.*

- **Message Type (MT)**

Refer to Section 5.3.1 for details of the MT field.

- **Packet Boundary Flag (PBF)**

Refer to section Section 5.3.1 for details of the PBF field.

- **Connection Identifier (Conn ID)**

The Connection Identifier (Conn ID) is used to indicate the previously setup Logical Connection to which this data belongs. The Conn ID is a 4-bit field containing a value from 0 to 15.

- **Payload Length (L)**

The Payload Length field indicates the number of Payload octets present. The Payload Length field is an 8-bit field containing a value from 0 to 255.

### 5.3.4 Segmentation and reassembly

The Segmentation and Reassembly functionality SHALL be supported by both the DH and the NFCC.

Segmentation and Reassembly of Messages SHALL be performed independently for Control Packets and Data Packets of each Logical Connection.

Any NCI Transport Mapping is allowed to define a fixed Maximum Transmission Unit (MTU) size in octets. If such a Mapping is defined and used, then if either DH or NFCC must transmit a Message (either Control or Data Message) that would generate a Packet (including Packet Header) larger than the MTU, the Segmentation and Reassembly (SAR) feature SHALL be used on the Message.

**The following rules apply to segmenting Control Messages**:

- For each segment of a Control Message, the header of the Control Packet SHALL contain the same MT, GID, and OID values.
- **From DH to NFCC:** The Segmentation and Reassembly feature SHALL be used when sending a Command Message from the DH to the NFCC that would generate a Control Packet with a payload larger than the “Max Control Packet Payload Size” reported by the NFCC at initialization. Each segment of a Command Message except for the last SHALL contain a payload with the length of “Max Control Packet Payload Size”.
- **From NFCC to DH:** When an NFCC sends a Control Message to the DH, regardless of the length, it MAY segment the Control Message into smaller Control Packets if needed for internal optimization purposes.

**The following rules apply to segmenting Data Messages**:

- For each segment of a Data Message, the header of the Data Packet SHALL contain the same MT and Conn ID.
- **From DH to NFCC:** if a Data Message payload size exceeds the max. Data Packet Payload Size, of the connection then the Segmentation and Reassembly feature SHALL be used on the Data Message.
- **From NFCC to DH:** when an NFCC sends a Data Message to the DH, regardless of the payload length it MAY segment the Data Message into smaller Data Packets for any internal reason, for example for transmission buffer optimization.

***Note:*** *PN722X uses modified NFCEE Discovery and Interface mechanism to communicate with CT Card using TDA. Thereby reducing the impact on Command, Response, and Notification changes required to be handled by DH.*

# 6 DH interface

## 6.1 Overview

The PN722X supports HIF1-I2C as physical interface to connect to the Main DH in Single Host Device Configuration and will support HIF2-I2C as physical interface to connect to the Main DH and HIF1-SPI as physical interface to connect to the Target DH (typically a security MCU) in Dual Host Device Configuration.

Independent of the physical interface, the PN722X has two main modes of operation to communicate with the DH:

1. NCI-based communications
2. HDLL-Based communications, only used when the PN722X is triggered to enter the “download mode”, to update its firmware.

The description of the transport layer in the next chapters is limited to NCI-based communications. For further information on the HDLL-based communications, please refer to chapter on PN722X NFCC encrypted secured firmware upload mode.

## 6.2 HIF1 interface

### 6.2.1 $I^2C$ interface

#### 6.2.1.1 Introduction

The HIF1-$I^2C$ interface of the PN722X is compliant with the [$I^2C$] Bus Specification, including device ID and Soft Reset. It is target-only, i.e. the SCL signal is an input driven by the host. PN722X does not use SCL clock stretching (i.e. maintain SCL low to pause the transmission).

***Note:*** *NCI packets length can be up to 258 bytes in both directions. The DH must consider this constraint when implementing the $I^2C$ driver.*

The PN722X HIF1-$I^2C$ interface supports standard (up to 100 kbit/s), fast-Speed mode (up to 400 kbit/s), fast-Plus mode (1 Mbit/s) and High-Speed mode (up to 3.4 Mbit/s).

The PN722X HIF1-$I^2C$ supports the 7-bit addressing mode, first 5-bits are fixed and is decimal 40, last two bits are configurable using ADDR0 and ADDR1 pins as per below table:

| ADDR1 Pin | ADDR0 Pin | I2C Address |
|---|---|---|
| 0 | 0 | 40 (0x28) |
| 0 | 1 | 41 (0x29) |
| 1 | 0 | 42 (0x2A) |
| 1 | 1 | 43 (0x2B) |

The following names are used in the document:

**Table 4. HIF1-$I^2C$ pins correspondence**

| Bus signal name | Correspondence in PN722X |
|---|---|
| I2C1_SDA | Equivalent to pin Host Interface 1 (I2C / SPI) : I2C1_SDA in PN722X Datasheet |
| I2C1_SCL | Equivalent to pin Host Interface 1 (I2C / SPI) : I2C1_SCL in PN722X Datasheet |

#### 6.2.1.2 NCI transport mapping

In the PN722X, there is no additional framing added for I$^2$C: an NCI packet (either data or control message, as defined in [NCI]) is transmitted over I$^2$C “as is”, i.e. without any additional byte (no header, no CRC etc.).

#### 6.2.1.3 Write sequence from the DH

As the HIF1-I$^2$C clock is controlled by the DH, only the DH can initiate an I$^2$C exchange.

A DH write sequence always starts with the sending of the PN722X HIF1-I$^2$C target address followed by the write bit (logical ‘0’: 0b). Then the PN722X HIF1-I$^2$C interface sends an I$^2$C ACK back to the DH for each data byte written by the DH.

It may send an I$^2$C NACK (negative acknowledge) when none of the reception buffers used by the NCI core in the PN722X is free and also when PN722X is in standby state. If one single byte of a complete NCI frame is NACKed by the PN722X, the DH has to resend the complete NCI frame and not only this single byte.

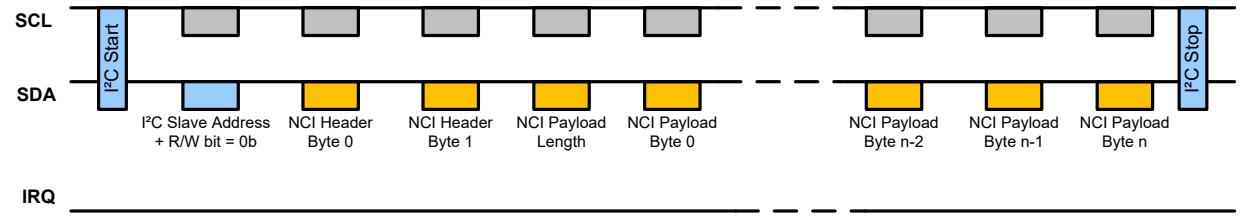



**Figure 16. I$^2$C write sequence**

***Note:*** *If PN722X has an NCI Message ready to be sent to the DH while it is receiving another NCI Message from the DH, the NFC_IRQ pin is raised during the Write Sequence. This is not an error and has to be accepted by the DH. Once the Write Sequence is completed, the DH has to start a Read Sequence (see Section "Read sequence from the DH").*

### 6.2.1.4 Read sequence from the DH

The DH shall never initiate a spontaneous I$^2$C read request. The DH shall wait until it is triggered by the PN722X. To trigger the DH, the PN722X generates a logical transition from Low to High on its IRQ pin. So after writing any NCI command, the DH shall wait until the PN722X raises its IRQ pin. The DH can then transmit a Read request to fetch the NCI answer from the PN722X . When the PN722X needs to send a spontaneous notification to the DH (for instance an RF Interface activation notification), the PN722X raises the IRQ pin and the DH performs a normal read as described above.

A DH Read Sequence always starts by sending the PN722X HIF1-I$^2$C Target Address followed by the read bit (logical '1'). Then the DH I$^2$C interface sends an ACK back to the PN722X for each data byte received.

Figure 17 is an example where the IRQ is raised so the DH can proceed a read.

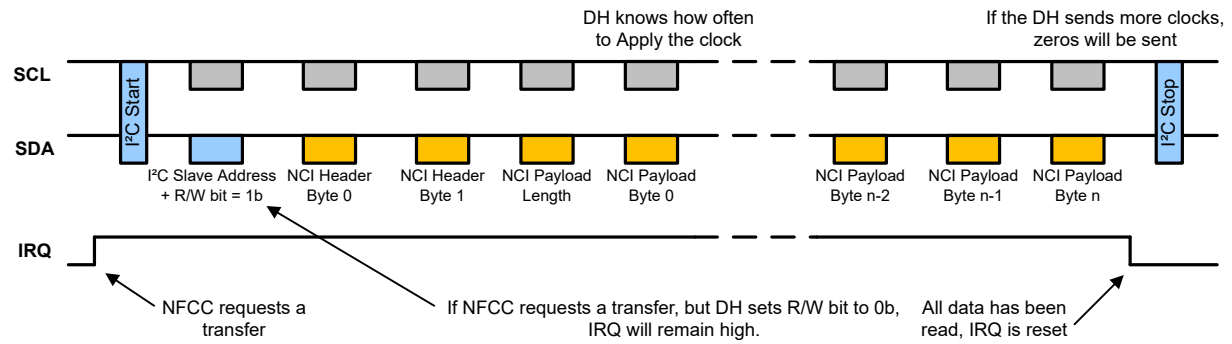



**Figure 17. >I$^2$C read sequence**

As indicated on the figure above, in case the PN722X requests a data transfer by raising the NFC_IRQ1 pin and the DH tries to initiate a write sequence by positioning the write bit to 0b, the PN722X keeps the NFC_IRQ1 active until the DH starts a read sequence. The DH is not allowed to proceed with a write sequence once the PN722X has set the NFC_IRQ1 pin to its active value (logical '1' in the figure above).

If PN722X has another message ready to be sent to the DH before the end of the on-going Read Sequence, the NFC_IRQ1 pin will be first deactivated at the end of the on-going Read Sequence and then reactivated to notify to the DH that a new message has to be read.

If PN722X has no more data to be sent but still the controller keeps SCL line active, PN722X keeps sending dummy bytes '0xFF'.

***Note:*** *IRQ in Figure 17 refers to NFC_IRQ1.*

#### 6.2.1.5 Split mode

The PN722X supports the interruption of a frame transfer, as defined in [I2C]. This feature is only available in Read Mode; it is forbidden to use it in Write Mode.

This can be useful in a system where the I2C bus is shared between several peripherals: It allows the host to stop an on-going exchange, to switch to another peripheral (with a different target address) and then to resume the communication with the PN722X.

Another typical use case for the split mode is to have the DH reading first the NCI packet header, to know what the Payload length is. The DH can then allocate a buffer with an appropriate size and read the payload data to fill this buffer. Figure 18 represents this use case.

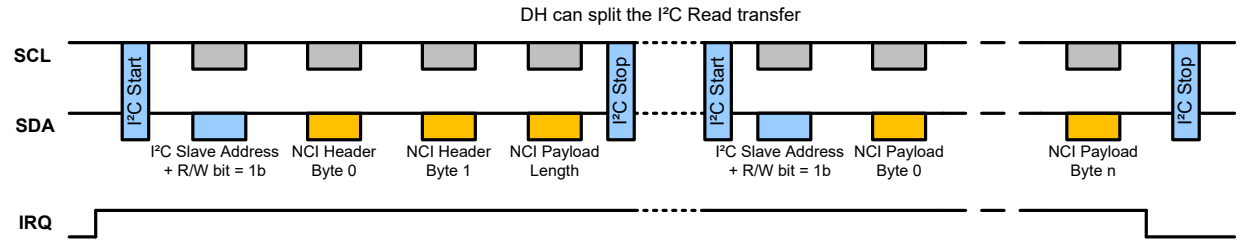



**Figure 18. I2C read sequence with split mode**

***Note:*** *After a hardware reset (power-on reset or RSTN pin reset), NFCC is ready to receive a frame after Tboot (see value in [PN722X _DS]). If no frame is received during 5 seconds, NFCC boots in NCI applicative mode followed by an entry in Active mode.*

### 6.2.2 SPI interface

#### 6.2.2.1 Introduction

The PN722X target-only SPI interface is compliant with the Freescale standard (see [SPI]).

- SPI speed up to 15 Mbit/s
- 8-bit data format only
- Supports all 4 modes of SPI (CPOL and CPHA)
- If no data is available, the CITO line is kept idle high (sends 0xFF bytes)
- Toggling the NSS line indicates a new frame

It is restricted to half-duplex communications.

**Table 5. SPI pins correspondence [1]**

| | |
|---|---|
| NSS (HIF1) | Equivalent to pin I2CADR0_SPINSS of the PN722X when using SPI |
| COTI (HIF2) | Equivalent to pin I2CADR1_SPIMOSI of the PN722X when using SPI. |
| CITO (HIF3) | Equivalent to pin I2CSDA_SPIMISO of the PN722X when using SPI |
| SCK (HIF4) | Equivalent to pin I2CSCL_SPISCK of the PN722X when using SPI |

[1] The "MOSI/MISO" replacement to "COTI/CITO" in this document follows the recommendation of the NXP - I2C standards organization.

#### 6.2.2.2 NCI transport mapping

A header byte is added to the NCI packets (either data or control message, as defined in Section 5.3).

Each data transfer over SPI starts with a header byte called “transfer direction detector”. The meaning of the header byte, depends on its value ( Table 6).

**Table 6. PN722X transfer direction detector**

| Byte value | Meaning |
|---|---|
| 0XXXXXXXb | DH write-access |
| 11111111b (0xFF) | DH read-access |

The “transfer direction detector” header byte restricts the full-duplex capabilities of SPI to half-duplex.

***Note:*** *PN722X use a different scheme of SPI Data transfer as compared SPI Data transfer scheme mentioned in NCI Specification. Write and Read sequence is as defined in this section.*

### 6.2.2.3 Write sequence from the DH

As the SPI clock is mastered by the DH, only the DH can initiate an SPI exchange.

A DH write sequence always starts with the sending of the Transfer Direction Detector Byte = 0XXXXXXXb. The PN722X considers all the following Bytes as part of an NCI packet.

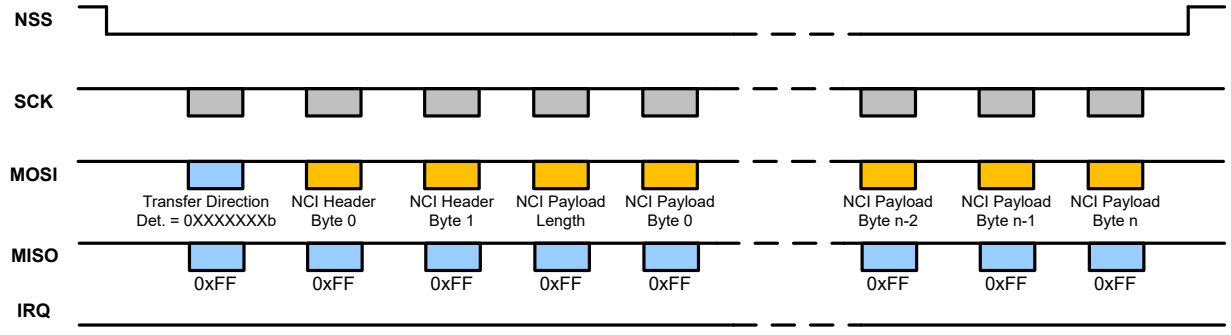



**Figure 19. Example of a SPI write-access**

***Note:*** *If PN722X has an NCI Message ready to be sent to the DH while it is receiving another NCI Message from the DH, the IRQ pin is raised somewhere during the Write Sequence. This is not an error and has to be accepted by the DH. Once the Write Sequence is completed, the DH has to start a Read Sequence.*

Issues may happen, in case PN722X is in Standby mode when the DH is writing a new frame: in such a condition, PN722X is not be able to catch the received frame.

In order to detect that PN722X is not ready to receive a frame, the DH has to monitor the CITO [3] line, to check the value of the first byte received on CITO while it is writing data on the COTI line. When the Write sequence starts:

- If the first Byte received on CITO is 0xFF, PN722X is ready, and the Write sequence proceeds with no issue.
- If the first Byte received on CITO is any other value than 0xFF, PN722X is not ready to receive a new frame. The DH has to resend the whole NCI frame, after a typical timeout of 5 ms.

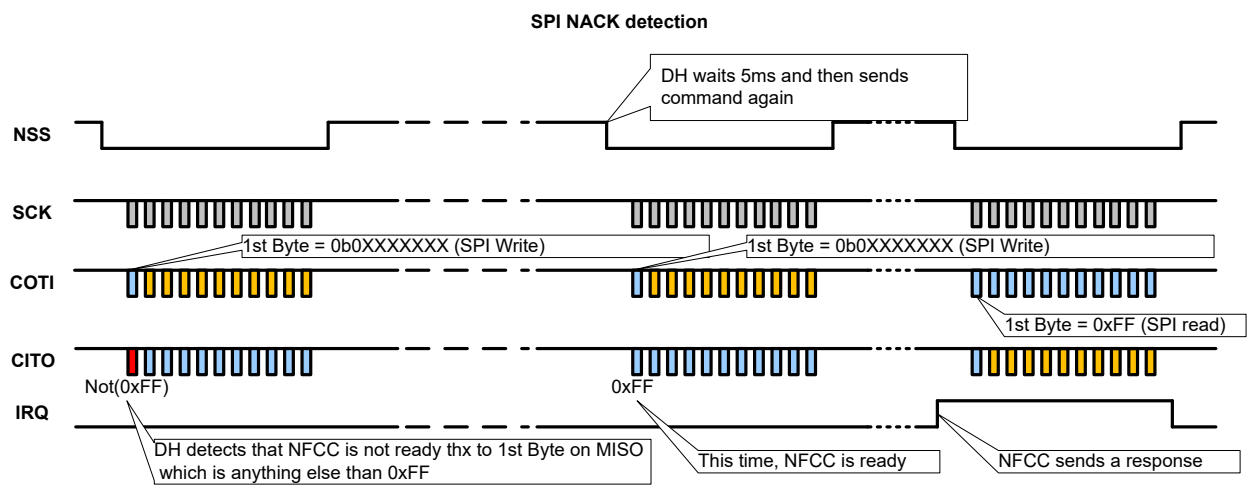



**Figure 20. SPI write access in case PN722X is in standby**

3 The "MOSI/MISO" replacement to "COTI/CITO" in this document follows the recommendation of the NXP - I2C standards organization.

#### 6.2.2.4 Read sequence from the DH

The DH shall never initiate a spontaneous SPI read request. The DH shall wait until it is triggered by the PN722X. To trigger the DH, the PN722X generates a logical transition from Low to High on its IRQ pin. So after writing any NCI command, the DH shall wait until the PN722X raises its IRQ pin. The DH can then transmit a Read request to fetch the NCI answer from the PN722X.

A DH Read Sequence always starts by the sending of the Transfer Direction Detector Byte = 0xFF.

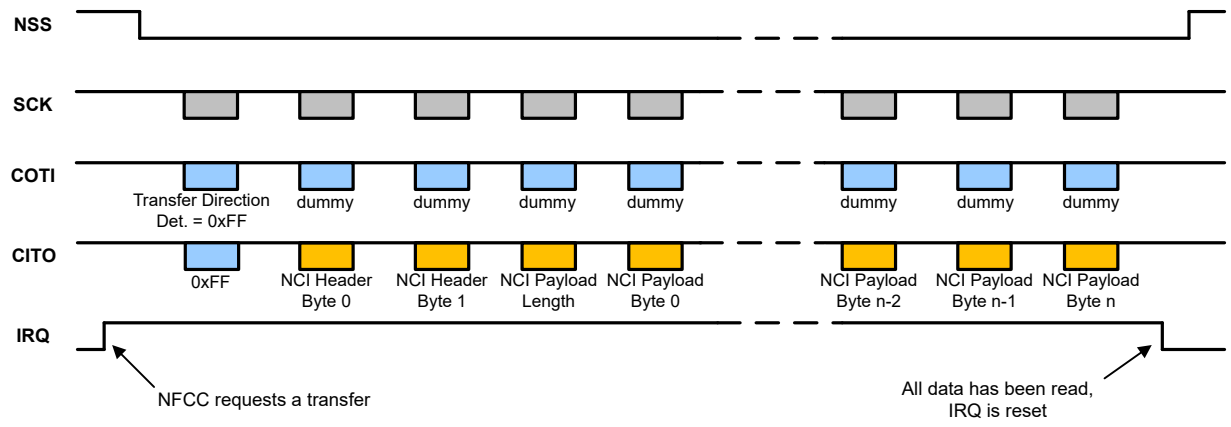



**Figure 21. Example of a SPI read access**

In Figure 21, PN722X requests a data transfer by raising the NFC_IRQ1 pin. The DH tries to initiate a write sequence by positioning the Transfer Direction Byte to 0b0XXXXXXX. PN722X keeps the NFC_IRQ1 to logical '1' until the DH starts a read sequence. The DH is not allowed to proceed with a write sequence once the PN722X has set the NFC_IRQ1 pin to its active value (logical '1' in Figure 21).

If PN722X has another message ready to be sent to the DH before the end of the on-going Read Sequence, the IRQ pin is first deactivated at the end of the on-going Read Sequence and then reactivated to notify to the DH that a new message has to be read.

***Note:*** *IRQ in Figure 21 refers to NFC_IRQ1.*

### 6.2.2.5 Split mode

The PN722X supports the interruption of a frame transfer over SPI. This feature is only available in Read Mode; it is forbidden to use it in Write Mode.

A typical use case for the split mode is to have the DH reading first the NCI packet header, to know what the Payload length is. The DH can then allocate a buffer with an appropriate size and read the payload data to fill this buffer. Figure 22 represents this use case.

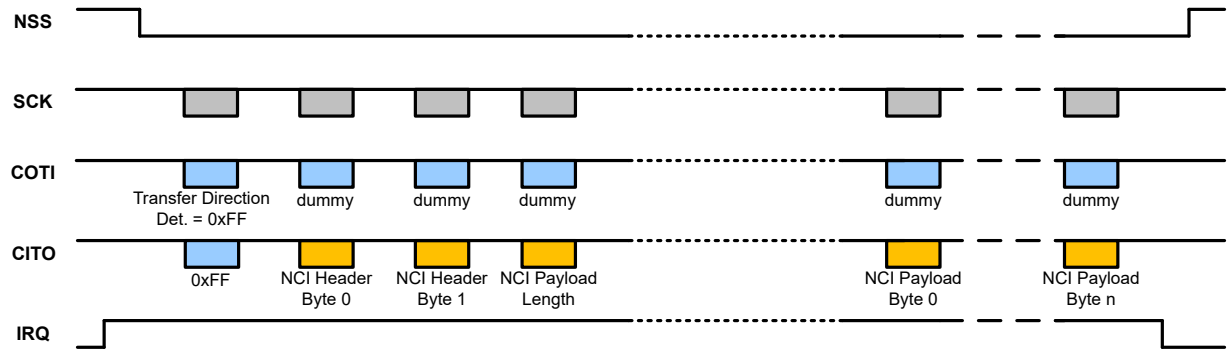



**Figure 22. SPI read sequence with split mode**

### 6.2.2.6 Invalid sequence from the DH

Any SPI data transfer starting by a Transfer Direction Detector Byte different from either 0XXXXXXXb or 11111111b is discarded by PN722X, as this is an invalid frame (Figure 23).

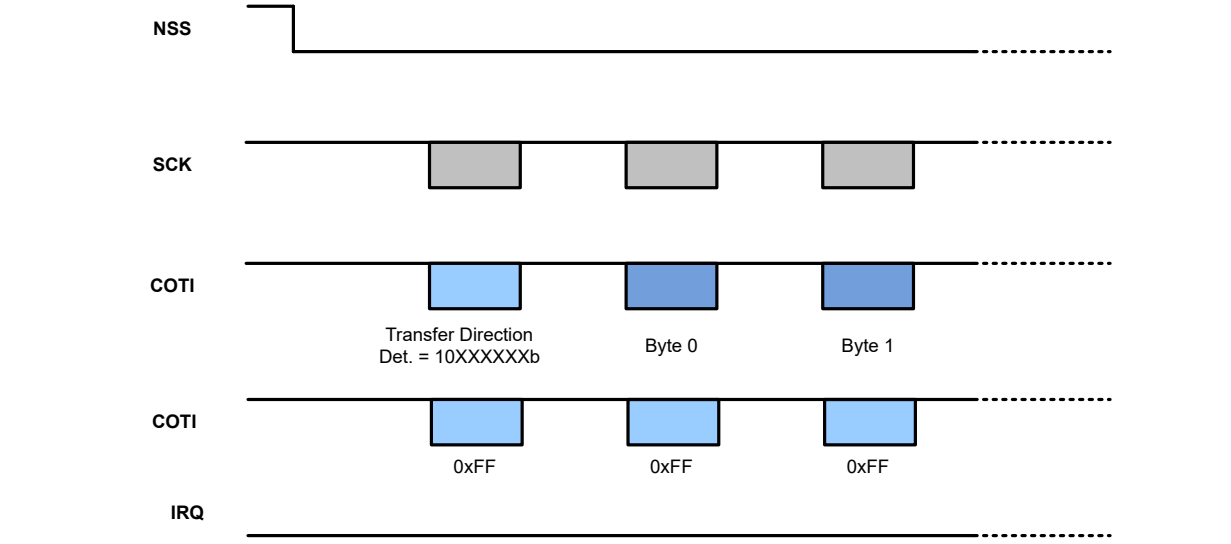



**Figure 23. Example of an invalid SPI access**

## 6.3 HIF2 interface

### 6.3.1 I²C interface

HIF2-I²C interface of the PN722X is compliant with the [I2C] Bus specification, including device ID and Soft Reset. It is target-only, i.e. the SCL signal is an input driven by the host. PN722X does not use SCL clock stretching (i.e. maintain SCL low to pause the transmission).

***Note:*** *NCI packets length can be up to 258 Bytes in both directions. The DH must consider this constraint when implementing the I²C driver.*

The PN722X HIF2-I²C interface supports standard (up to 100 kbit/s), fast-Speed mode (up to 400 kbit/s), fast-Plus mode (1 Mbit/s).

The PN722X supports the 7-bit addressing mode, which is FIXED at production.

***Note:*** *The PN722X I²C 7-bit address is fixed to 0x38.*

### 6.3.2 NCI transport mapping

In the PN722X, there is no additional framing added for I²C: An NCI packet (either data or control message, as defined in [NCI]) is transmitted over I²C “as is”, i.e. without any additional byte (no header, no CRC etc.).

### 6.3.3 Write sequence from the DH

As the HIF2-I²C clock is controlled by the DH, only the DH can initiate an I²C exchange.

A DH write sequence always starts with sending of the PN722X HIF2-I²C target address followed by the write bit (logical ‘0’: 0b). Then the PN722X HIF2-I²C interface sends an I²C ACK back to the DH for each data byte written by the DH.

It may send an I²C NACK (negative acknowledge) when none of the reception buffers used by the NCI core in the PN722X is free and also when PN722X is in standby state. If one single byte of a complete NCI frame is NACKed by the PN722X, the DH has to resend the complete NCI frame and not only this single byte.

When standby feature of PN722X is supported and HIF2-I²C interface, GPIO3 needs to be used as wake-up source to PN722X to exit from standby and the I²C NACK shall be sent.

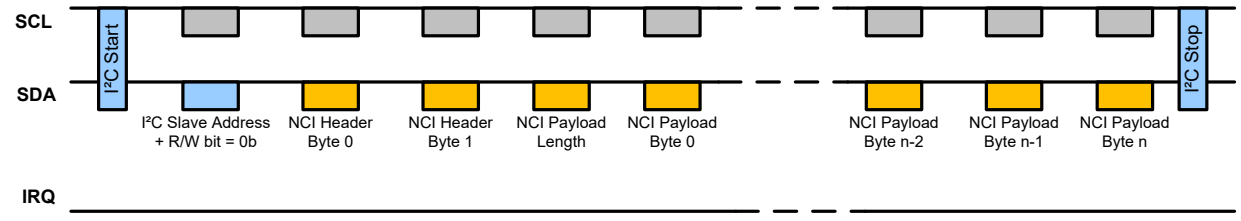



**Figure 24. I²C write sequence**

***Note:*** *With HIF2-I²C it may happen that PN722X raised IRQ and an NCI Message is ready to be read by DH. If DH tries to send an NCI Message, then it might be NACKed by PN722X. In such a condition, the DH shall complete the Read Sequence and shall retry to perform the Write Sequence. This is not an error and has to be accepted by the DH.*

### 6.3.4 Read sequence from the DH

The DH shall never initiate a spontaneous HIF2-I$^2$C read request. The DH shall wait until the PN722X triggers the read request. To trigger the DH, the PN722X generates a logical transition from Low to High on its IRQ pin. After writing any NCI command, the DH shall wait until the PN722X raises its IRQ pin. The DH can then transmit a Read request to fetch the NCI answer from the PN722X . When the PN722X needs to send a spontaneous notification to the DH (for instance an RF Interface activation notification), the PN722X raises the IRQ pin and the DH performs a normal read as described above.

A DH Read Sequence always starts by sending the PN722X HIF2-I$^2$C Target Address followed by the read bit (logical '1'). Then the DH I$^2$C interface sends an ACK back to the PN722X for each data byte received.

Figure 25 shows the example where the IRQ is raised so the DH can proceed a read.

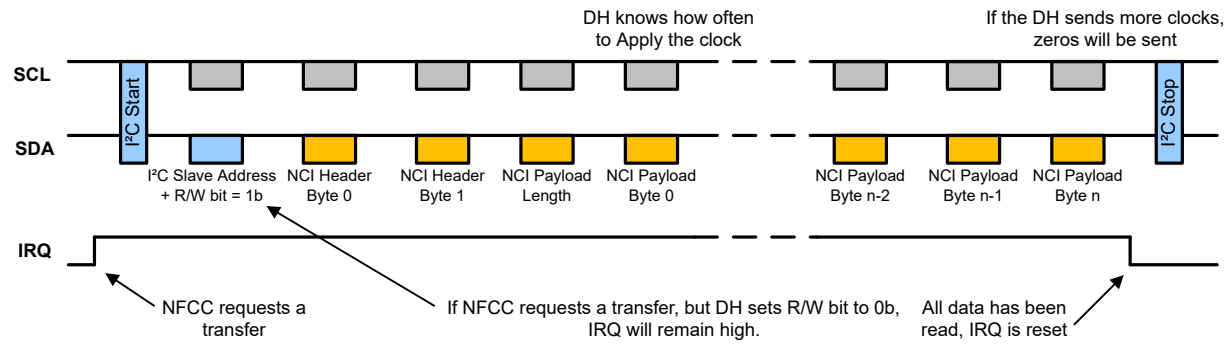



**Figure 25. I$^2$C read sequence**

In Figure 25, PN722X requests a data transfer by raising the NFC_IRQ2 pin. The DH tries to initiate a write sequence by positioning the write bit to 0b. PN722X keeps the NFC_IRQ2 active until the DH starts a read sequence. The DH is not allowed to proceed with a write sequence once the PN722X has set the NFC_IRQ2 pin to its active value (logical '1' in Figure 25).

If PN722X has another message ready to be sent to the DH before the end of the on-going Read Sequence, first the NFC_IRQ2 pin is deactivated at the end of the on-going Read Sequence. Second, NFC_IRQ2 pin is reactivated to notify to the DH that a new message has to be read.

If PN722X has no more data to be sent but the controller keeps SCL line active, PN722X keeps sending dummy bytes '0xFF'.

***Note:*** *IRQ in Figure 25 refers to NFC_IRQ2.*

### 6.3.5 Split mode

The PN722X supports the interruption of a frame transfer, as defined in [I2C]. This feature is only available in Read Mode and is forbidden in Write Mode.

This can be useful in a system where the I2C bus is shared between several peripherals: It allows the host to stop an on-going exchange, to switch to another peripheral (with a different target address) and then to resume the communication with the PN722X.

Another typical use case for the split mode is to have the DH reading first the NCI packet header, to know what the Payload length is. The DH can then allocate a buffer with an appropriate size and read the payload data to fill this buffer. Figure 26 illustrates this use case.

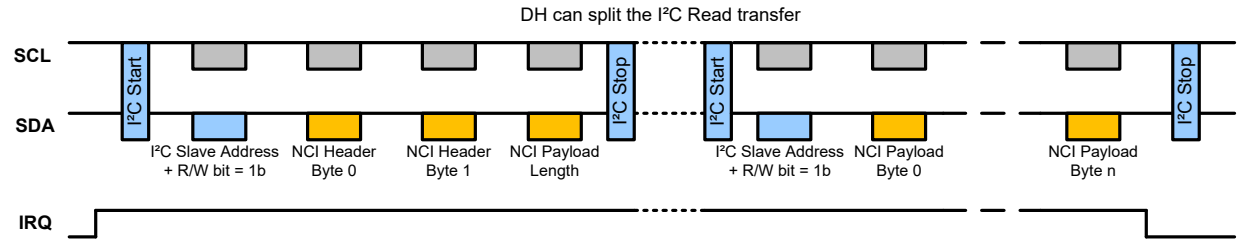



**Figure 26. I2C Read sequence with split mode**

## 6.4 GPIO and GPI interface

The PN722X supports multiple GPIOs and one GPI interface, each one having a specific functionality as described below:

**Table 7. PN722X GPIO usage**

| GPIO name | Description |
|---|---|
| TDA_CS0, TDA_CS1 and TDA_CS2 | Used to connect to chip-select of TDA's |

**Table 8. PN722X GPI usage**

| GPI name | Description |
|---|---|
| Mode_Switch_GPI | Used to switch between NFC Forum and EMVCo mode in Single Main DH Configuration for all PN722X variants using HIF1-I2C Interface.<br>Used to switch between NFC Forum execution with Main DH using HIF2-I2C Interface and EMVCo mode execution with Target DH using HIF1-SPI Interface .<br>'0' – NFC Forum Mode<br>'1' – EMVCo Mode<br>***Note:*** *When DH toggles the Mode_Switch_GPI, the DH must wait for CORE_RESET_NTF and validate the state.* |

***Note:*** *A spike on Mode_Switch_GPI may lead PN722X to enter into NFC Forum or EMVCo Mode based on the first GPI state transition that PN722X observed. DH must ensure that PN722X is in correct state by reading the Proprietary reason code in CORE_RESET_NTF indicating boot due to Switch Mode reset.*

***Note:*** *Dual Host Device Configuration: Main Host device should make sure both PN722X and Target Host are in the correct state to avoid deadlocks in the system. Once Main host toggles Mode_Switch_GPI of PN722X to high to enter into EMVCo mode, Main Host device shall also indicate Target Host to take over the communication with NFCC and also check if the connection establishment is successful.*

# 7 Compliance to [NCI] and PN722X NFCC extensions

The PN722X NFCC is a complex contactless System on Chip with numerous features. Yet, [NCI] as defined by the NFC Forum does not give full access to all these features. Therefore, NXP had to extend [NCI] with proprietary extensions. The PN722X NFCC DH interface which includes [NCI] plus the PN722X NFCC extensions is referenced in the present document as [PN722X NFCC-NCI].

***Note:*** *NXP uses [NCI] as much as possible and tries to limit the use of proprietary extensions.*

## 7.1 Feature-based comparison of [NCI] and [PN722X NFCC-NCI]

The table below represents the features overview of the PN722X NFCC. It highlights the main differences between the NCI standard ([NCI]) and [PN722X NFCC-NCI]. The chapter column contains shortcuts to the section in the document where the feature is described in detail.

**Table 9. Features overview**

| Features | [NCI 2.2] | [PN722X NFCC-NCI] |
|---|---|---|
| RF Discovery activity based on NFC Forum | ☑ | ☑ |
| RF Discovery activity based on EMVCo | ☒ | ☑ |
| Reader/Writer ISO-DEP for NFC-A and NFC-B, T2T, T3T, T4T, T5T | ☑ | ☑ |
| Reader/Writer MIFARE Classic, MIFARE Plus | ☒ | ☑ |
| Card Emulation ISO-DEP for NFC-A | ☑ | ☑ |
| Card Emulation for other technologies | ☑ | ☒ |
| Testing: PRBS test | ☒ | ☑ |
| Configuration: Power and clock management, RF Settings, APC | ☒ | ☑ |
| Others: Encrypted Secured FW upload | ☒ | ☑ |

☑ ... covered

☒ … not covered

## 7.2 [NCI] Implementation in the PN722X NFCC

[NCI] defines several features which are optional or configurable. For instance, data exchange can use an optional flow control, for which the number of credits is defined by the NFCC. The maximum number of simultaneous Dynamic logical connections is also up to the NFCC.

This section describes [NCI] features which are optional or configured by the NFCC, to highlight how they are implemented in the PN722X NFCC.

### 7.2.1 Logical connections and credits

Figure 27 shows a simplified overview of an NFC device as defined in the NFC Forum.

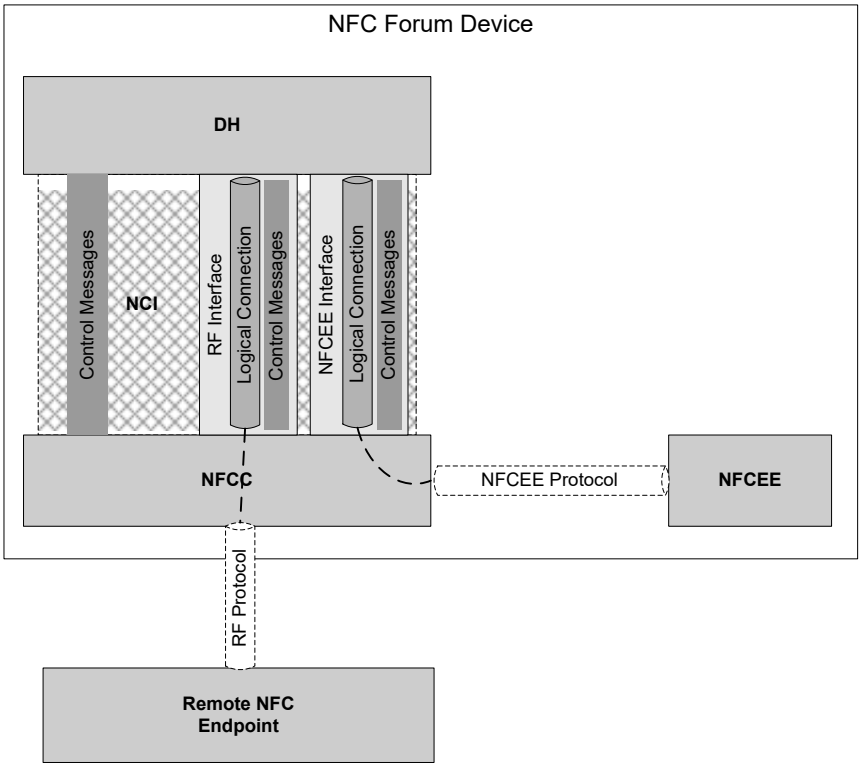



**Figure 27. NFC Forum device architecture**

Logical connections are used to transport data between the DH and the NFCC. Although optional in [NCI], [PN722X NFCC-NCI] implements data flow control based on credits management. To minimize the required buffer/memory size, the **number of credits is limited to 1** on each logical connection.

In addition to the mandatory static RF logical connection, [PN722X NFCC-NCI] allows **to create only 1 additional** Dynamic logical connection. So, the “Max Logical Connections” parameter reported in **CORE_INIT_RSP** equals **0x01** for [PN722X NFCC-NCI] and the number of credits of the NFCEE Connection equals to 0x01**.** That means that when the DH must create a new logical connection, it has to first close the currently opened one, if any.

Table 10 gives an overview of the logical connection and credits available in the [PN722X NFCC-NCI].

**Table 10. Logical Connections/Credits configuration**

| Logical connection | Number of connections | Number of credits | Max. Data Packet payload Size |
|---|---|---|---|
| Static connection | - 1 for RF End Point<br>- 1 for NFCEE Connection | 1<br>1 | [255]<br>[255] |
| Dynamic connection | - 1 for NCI loopback testing | 1 | [255] |

To optimize the number of [PN722X NFCC-NCI] internal buffers, all the logical connections shall not be used at the same time. Therefore, only the following scenarios are possible at the same time depending on the RF State Machine:

**Table 11. Logical Connections supported depending on RF State Machine**

| NCI RF States | NCI CMD | Static RF EndPoint | Loopback | CT NFCEE |
|---|---|---|---|---|
| RFST_IDLE | ☑ | ☒ | ☑ | ☑ |
| Others RF States | ☑ | ☑ | ☒ | ☑ |

### 7.2.2 Compliance to [NCI] control messages

**Table 12. Status on the compliance to [NCI] control messages**

| Group | Control messages | Status |
|---|---|---|
| CORE | CORE_RESET_CMD / RSP / NTF | **Full support**[1] |
| | CORE_INIT_CMD / RSP | **Full support** |
| | CORE_SET_CONFIG_CMD / RSP | **Full support** |
| | CORE_GET_CONFIG_CMD / RSP | **Full support** |
| | CORE_CONN_CREATE_CMD / RSP | **Full support** |
| | CORE_CONN_CLOSE_CMD / RSP | **Full support** |
| | CORE_CONN_CREDITS_NTF | **Full support**[2] |
| | CORE_GENERIC_ERROR_NTF | **Full support** |
| | CORE_INTERFACE_ERROR_NTF | **Full support** |
| | CORE_SET_POWER_SUB_STATE_CMD/RSP | **No support** |
| RF | RF_DISCOVER_MAP_CMD / RSP | **Full support** |
| | RF_SET_LISTEN_MODE_ROUTING_CMD / RSP | **Full support** |
| | RF_GET_LISTEN_MODE_ROUTING_CMD / RSP / NTF | **Full support** |
| | RF_DISCOVER_CMD / RSP / NTF | **Full support** |
| | RF_DISCOVER_SELECT_CMD / RSP | **Full support** |
| | RF_INTF_ACTIVATED_NTF | **Full support** |
| | RF_DEACTIVATE_CMD / RSP / NTF | **Full support** |
| | RF_FIELD_INFO_NTF | **Full support** |
| | RF_T3T_POLLING_CMD / RSP / NTF | **Full support** |
| | RF_NFCEE_ACTION_NTF | **No support** |
| | RF_NFCEE_DISCOVERY_REQ_NTF | **No support** |
| | RF_PARAMETER_UPDATE_CMD / RSP | **Partial support**[3] |
| | RF_INTF_EXT_START_CMD / RSP | **No support** |
| | RF_INTF_EXT_STOP_CMD / RSP | **No support** |
| | RF_EXT_AGG_ABORT_CMD / RSP | **No support** |
| | RF_NDEF_ABORT_CMD / RSP | **No support** |
| | RF_ISO_DEP_NAK_PRESENCE_CMD / RSP / NTF | **Full support** |
| | RF_SET_FORCE_NFCEE_ROUTING_CMD / RSP | **No support** |
| NFCEE | NFCEE_DISCOVER_CMD / RSP / NTF | **Full support** |
| | NFCEE_MODE_SET_CMD / RSP / NTF | **Full support** |
| | NFCEE_STATUS_NTF | **No support** |
| | NFCEE_POWER_AND_LINK_CNTRL_CMD / RSP | **No support** |

[1] CORE_RESET_NTF has sometimes an additional field, not compliant to [NCI]. See Section 9

[2] PN722X may send a CORE_CONN_CREDIT_NTF with 0 credit indicating that NFCC received a data packet but could not release the data immediately, due to a flow control on the NFCEE/RF interface preventing NFCC to release the credit. In such situation (for example during extended APDU in wired mode), a subsequent CORE_CONN_CREDIT_NTF with 1 credit is sent once NFCC sends the packet.

[3] RF_PARAMETER_UPDATE_CMD is not fully supported as corresponding scenarios can automatically be fulfilled when using ISO-DEP RF interface.

### 7.2.3 Compliance to [NCI] RF interfaces

Figure 28 shows the RF interfaces available in [NCI].

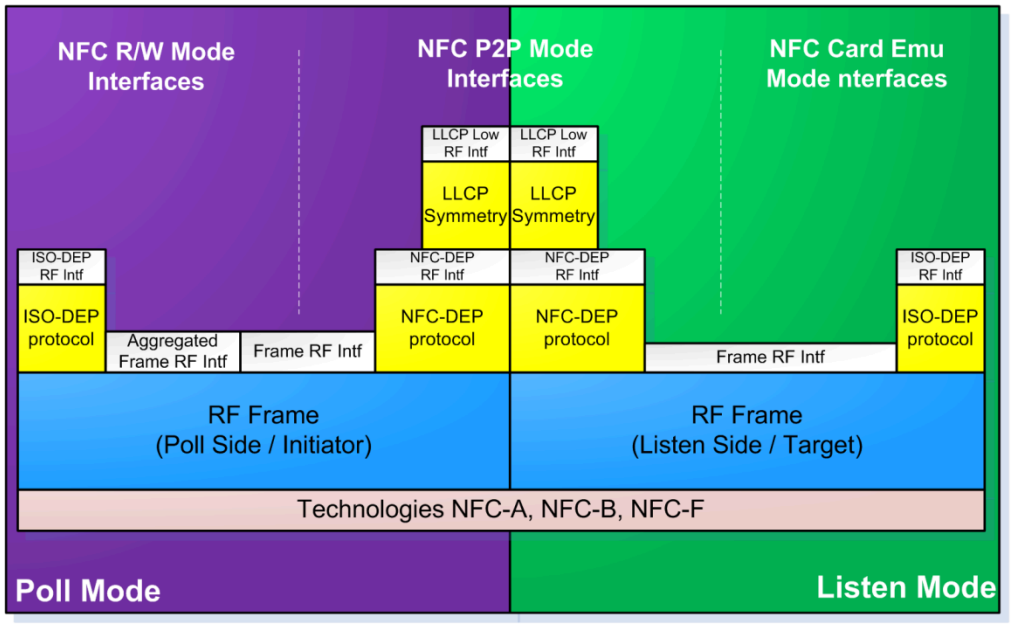



**Figure 28. [NCI] RF interface architecture**

Table 13 shows which RF interfaces PN722X NFCC supports and reports into CORE_INIT_RSP.

**Table 13. NCI interface limitations**

| RF Interface present in [NCI] | Status |
|---|---|
| Poll side Frame RF interface | **Full support** |
| Listen side Frame RF interface (3A/3B) | **No Support** |
| Poll side ISO-DEP interface | **Full support** |
| Listen side ISO-DEP interface | **Full support** |
| Poll side and Listen side NFC-DEP interface | **No Support** |
| Poll side Frame RF interface with frame Aggregated extension | **No Support** |

### 7.2.4 Compliance to [NCI] RF Discovery

Table 14 lists the phases that PN722X NFCC supports.

**Table 14. Supported NCI Discovery phases**

| RF Interface present in [NCI] | Value | Status |
|---|---|---|
| NFC_A_PASSIVE_POLL_MODE | 0x00 | **Full support** |
| NFC_B_PASSIVE_POLL_MODE | 0x01 | **Full support** |
| NFC_F_PASSIVE_POLL_MODE | 0x02 | **Full support** |
| NFC_ACTIVE_POLL_MODE | 0x03 | **No Support** |
| NFC_V_PASSIVE_POLL_MODE | 0x06 | **Full support** |
| NFC_A_PASSIVE_LISTEN_MODE | 0x80 | **Full support** |
| NFC_B_PASSIVE_LISTEN_MODE | 0x81 | **No Support** |
| NFC_F_PASSIVE_LISTEN_MODE | 0x82 | **No Support** |
| NFC_ACTIVE_LISTEN_MODE | 0x83 | **No Support** |

### 7.2.5 Compliance to [NCI] configuration parameters

[NCI] defines a set of configuration parameters in NCI Spec [Configuration Parameter Tags]. PN722X NFCC supports most of the parameters, apart from one subset. PN722X supports device resolution of two devices in each technology and activation of the first two device IDs using RF_DISCOVER_SELECT_CMD.

Table 15 lists [NCI] configuration parameters, with the default value in PN722X.

**Table 15. Compliance to [NCI] configuration parameters**

| Configuration parameter | Status on PN722X | Coming from | Default value | Behavior if partial/no support |
|---|---|---|---|---|
| TOTAL DURATION | **Full support** | [NCI] | 0x03E8 (1 s) | Minimum Value: 350 ms<br>Maximum Value: 65 seconds (without LPCD)<br>Maximum Value: 2.6 seconds (with LPCD)<br>*NOTE: Step size is 10 milli seconds.* |
| CON_DISCOVERY_ PARAM | **No support** | [ACTIVITY] | — | — |
| PA_BAIL_OUT | **Full support** | [ACTIVITY] | 0x01 | Bail Out is enabled by default in Poll/NFC-A |
| PA_DEVICES_LIMIT | **Full support** | [ACTIVITY] | 0x02 | Value supported from 0x01 to 0x02 |
| PB_AFI | **Full support** | [DIGITAL] | 0x00 | — |
| PB_BAIL_OUT | **Full support** | [ACTIVITY] | 0x01 | Bail Out is enabled by default in Poll/NFC-B |
| PB_ATTRIB_PARAM1 | **Full support** | [DIGITAL] | 0x00 | — |
| PB_SENSB_REQ_ PARAM | **Full support** | [DIGITAL] | 0x00 | — |
| PB_DEVICES_LIMIT | **Full support** | [ACTIVITY] | 0x02 | Value supported from 0x01 to 0x02 |
| PF_BIT_RATE | **Full support** | [DIGITAL] | 0x01 (212 kB/ s) | — |
| PF_BAIL_OUT | **Full support** | [ACTIVITY] | 0x01 | Bail Out is enabled by default in Poll/NFC-F |
| PF_DEVICES_LIMIT | **Full support** | [ACTIVITY] | 0x02 | Value supported from 0x01 to 0x02 |

**Table 15. Compliance to [NCI] configuration parameters***...continued*

| Configuration parameter | Status on PN722X | Coming from | Default value | Behavior if partial/no support |
|---|---|---|---|---|
| PI_B_H_INFO | **Full support** | [DIGITAL] | empty | — |
| PI_BIT_RATE | **Full support** | [DIGITAL] | 0x00 | 106 kB/s by default but can be up to 848 kB/s |
| PN_NFC_DEP_PSL | **No Support** | [DIGITAL] | — | — |
| PN_ATR_REQ_GEN_BYTES | **No Support** | [DIGITAL] | — | — |
| PN_ATR_REQ_CONFIG | **No Support** | [DIGITAL] | — | — |
| PV_DEVICES_LIMIT | **Full support** | [ACTIVITY] | 0x02 | Value supported from 0x01 to 0x02 |
| LA_BIT_FRAME_SDD[1] | **No support** | [DIGITAL] | — | Use Proprietary 0xA2 configuration to update these parameters in EEPROM. |
| LA_PLATFORM_CONFIG[1] | **No support** | [DIGITAL] | — | |
| LA_SEL_INFO[1] | **No support** | [DIGITAL] | — | |
| LA_NFCID1 | **No support** | [DIGITAL] | — | — |
| LB_SENSB_INFO | **No Support** | [DIGITAL] | — | — |
| LB_NFCID0 | **No Support** | [DIGITAL] | — | — |
| LB_APPLICATION_DATA | **No Support** | [DIGITAL] | — | — |
| LB_SFGI | **No Support** | [DIGITAL] | — | — |
| LB_FWI_ADC_FO | **No Support** | [DIGITAL] | — | — |
| LB_BIT_RATE | **No Support** | [NCI] | — | — |
| LF_T3T_IDENTIFIERS_1..4 | **No Support** | [DIGITAL] | — | — |
| LF_T3T_IDENTIFIERS_5..16 | **No Support** | [DIGITAL] | — | — |
| LF_PROTOCOL_TYPE | **No Support** | [DIGITAL] | — | — |
| LF_T3T_MAX | **No Support** | [NCI] | — | — |
| LF_T3T_FLAGS | **No Support** | [NCI] | — | — |
| LF_T3T_RD_ALLOWED | **No Support** | [NCI] | — | — |
| LF_PROTOCOL_TYPE | **No Support** | [NCI] | — | — |
| LI_A_RATS_TB1 | **Full support** | [DIGITAL] | 0x04 | — |
| LI_A_HIST_BY | **Full support** | [DIGITAL] | empty | — |
| LB_H_INFO_RESP | **No Support** | [DIGITAL] | — | — |
| LI_A_BIT_RATE | **Full support** | [DIGITAL] | 0x00 (106 kB/s) | 106 kB/s by default but can be up to 848 kB/s |
| LI_A_RATS_TC1 | **Full support** | [NCI] | 0x00 | — |

**Table 15. Compliance to [NCI] configuration parameters***...continued*

| Configuration parameter | Status on PN722X | Coming from | Default value | Behavior if partial/no support |
|---|---|---|---|---|
| LN_WT | **No Support** | [DIGITAL] | 0x08 | — |
| LN_ATR_RES_GEN_BYTES | **No Support** | [DIGITAL] | Empty | — |
| LN_ATR_RES_CONFIG | **No Support** | [DIGITAL] | 0x30 | — |
| PACM_BIT_RATE | **No Support** | [NCI] | 0x01 | — |
| RF_FIELD_INFO | **Full support** | [NCI] | 0x00 | — |
| RF_NFCEE_ACTION | **No Support** | [NCI] | 0x01 | — |
| NFCDEP_OP | **No Support** | [NCI] | 0x0E | — |
| LLCP_VERSION | **No Support** | [LLCP] | 0x11 | — |
| NFCC_CONFIG_CONTROL | **No support** | [NCI] | — | — |

[1] Using Extended TLV with 0xA2XX, you can configure this parameter which is stored in EEPROM.

### 7.2.6 Compliance to [NCI] data messages

PN722X is fully compliant to the [NCI] data messages.

## 7.3 Extensions added to [NCI] to allow full control of the PN722X NFCC

The [PN722X NFCC-NCI] extensions section gives a quick overview of the numerous extensions required to [NCI] to give full access to all the features available in the PN722X NFCC.

### 7.3.1 [PN722X NFCC-NCI] extensions to [NCI] RF protocols

PN722X NFCC supports an additional protocol than handled today by [NCI].

It is required to extend the RF Interfaces defined in NCI Spec [Table : RF Protocols] such that this protocol can be configured in various commands/notifications:

**Table 16. Proprietary RF protocols**

| Chapter | Value | Description |
|---|---|---|
| →Section 10.1.1 | 0x80 | PROTOCOL_**MIFARE_CLASSIC** |
| | Others | Reserved for Proprietary protocols |

### 7.3.2 [PN722X NFCC-NCI] extensions to [NCI] bit rates in NFC-F

PN722X NFCC offers the possibility to poll for NFC-F @ 212 kB/s and NFC-F @ 424 kB/s. Unfortunately, [NCI] only allows configuring one of these 2-bit rates, but not both in the same discovery sequence. The [NCI] parameter used to configure the bit rate in NFC-F is PF_BIT_RATE; the values which can be applied to this parameter are defined in the NCI Spec [Table : Discovery Configuration Parameters for Poll F].

It is therefore required to extend this table such that the PN722X NFCC is configured to poll during one discovery sequence for NFC-F @ 212 kB/s and NFC-F @ 424 kB/s. The proprietary value 0x80 is used for that purpose, and has to be restricted to technology NFC-F in NFC Forum Mode.

**Table 17. Proprietary bit rates**

| Chapter | Value | Description |
|---|---|---|
| →Section 10.1.2 | 0x80 | NFC_BIT_RATE_212 and NFC_BIT_RATE_424 |

### 7.3.3 [PN722X NFCC-NCI] extensions to [NCI] RF interfaces

PN722X NFCC offers some features which are not accessible using the currently defined RF interfaces in NCI Spec. So, the NCI Spec [Table : RF Interfaces] must be extended with some proprietary RF interfaces, as described in the table below:

**Table 18. RF interfaces extension**

| Chapter | New RF interface | Value | Brief description |
|---|---|---|---|
| Section 10.1.1.2 | TAG-CMD | 0x80 | This new interface adds a header to the data payload, in order to encode commands such as:<br>- T2T/MFUL sector select command<br>- MIFARE Classic Authenticate command |
| | | Others | Reserved for proprietary RF interfaces |

### 7.3.4 [PN722X NFCC-NCI] extensions to [NCI] control messages and [NCI] notifications

This section contains all the additional commands/notifications in [PN722X NFCC-NCI].

**Table 19. PN722X NFCC-NCI additional commands/notifications**

| Chapter | PN722X NFCC-NCI control message | Brief description | Support on PN722X |
|---|---|---|---|
| Section 9.2.1 | NCI_PROPRIETARY_ ACT_ CMD/RSP | Command used by the DH to activate proprietary extension of NFCC | **Full Support** |
| Section 12.4.1 | CORE_SET_POWER_ MODE_CMD/RSP | Command allowing the DH to configure the power mode (standby or no standby state). | **Full Support** |
| Section 10.1.3.3 | WTX_NTF | Notification indicating that RF communication is in phase of WTX(RTOX) REQ/RESP exchange for longer period | **Full Support** |
| Section 7.3.10 | PLL_UNLOCKED_NTF | Notification indicating that RF PLL fails to lock after it is started | **Full Support** |
| Section 7.3.11 | TXLDO_ERROR_NTF | Notification sent to indicate that TxLDO cannot be started successfully | **Full Support** |
| Section 7.3.12 | FLUSH_SRAM_AO_TO_ FLASH | Command sent by the DH to save the content of the RAM always on area to FLASH. | **Full Support** |

[NCI] defines some rules which constraint the use of the control messages. That means that depending on the state the NCI RF State Machine is in, depending on the RF Interface used, depending on some parameters, the control messages are valid or incorrect, and sometimes they trigger state transitions.

NXP has extended these rules for the [PN722X NFCC-NCI] extensions.

Figure 29 and Figure 30 illustrate the rules for most of the notification and commands:

| | | | CURRENT STATE | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | RFST_IDLE | | RFST_DISCOVERY | | RFST_W4_ALL_DISC. | | RFST_W4_HOST_SELECT | | RFST_POLL_ACTIVE | | RFST_LISTEN_ACTIVE | | RFST_LISTEN_SLEEP | |
| Source | Control Message | parameter / RF Interface | Next state | Response STATUS | Next state | Response STATUS | Next state | Response STATUS | Next state | Response STATUS | Next state | Response STATUS | Next state | Response STATUS | Next state | Response STATUS |
| NCI1.0 | CORE_RESET_CMD/RSP | | | Status_OK | IDLE | Status_OK | IDLE | Status_OK | IDLE | Status_OK | IDLE | Status_OK | IDLE | Status_OK | IDLE | Status_OK |
| NCI1.0 | CORE_INIT_CMD/RSP | | | Status_OK | | SEMANTIC | | SEMANTIC | | SEMANTIC | | SEMANTIC | | SEMANTIC | | SEMANTIC |
| NCI2.0 | CORE_SET_CONFIG_CMD/RSP | generic and proprietary param | | Status_OK | | SEMANTIC | | SEMANTIC | | SEMANTIC | | SEMANTIC | | SEMANTIC | | SEMANTIC |
| | | LF_T3T_FLAGS_CON_DISCOVERY | | Status_OK | | Status_OK | | Status_OK | | Status_OK | | Status_OK | | SEMANTIC | | Status_OK |
| NCI1.0 | CORE_GET_CONFIG_CMD/RSP | | | Status_OK | | Status_OK | | Status_OK | | Status_OK | | Status_OK | | SEMANTIC | | Status_OK |
| | | | | Status_OK | | Status_OK | | Status_OK | | Status_OK | | Status_OK | | SEMANTIC | | Status_OK |
| NCI1.0 | CORE_CONN_CREATE_CMD/RSP | | | Status_OK | | SEMANTIC | | SEMANTIC | | SEMANTIC | | SEMANTIC | | SEMANTIC | | SEMANTIC |
| NCI1.0 | CORE_CONN_CLOSE_CMD/RSP | | | Status_OK | | SEMANTIC | | SEMANTIC | | SEMANTIC | | SEMANTIC | | SEMANTIC | | SEMANTIC |
| NCI2.0 | CORE_SET_POWER_SUB_STATE | | | Status_OK | | Status_OK | | Status_OK | | Status_OK | | Status_OK | | SEMANTIC | | Status_OK |
| NCI1.0 | RF_DISCOVER_MAP_CMD/RSP | | | Status_OK | | SEMANTIC | | SEMANTIC | | SEMANTIC | | SEMANTIC | | SEMANTIC | | SEMANTIC |
| NCI1.0 | RF_SET_LISTEN_MODE_ROUTING_CMD/RSP | | | Status_OK | | SEMANTIC | | SEMANTIC | | SEMANTIC | | SEMANTIC | | SEMANTIC | | SEMANTIC |
| NCI1.0 | RF_GET_LISTEN_MODE_ROUTING_CMD/RSP | | | Status_OK | | SEMANTIC | | SEMANTIC | | SEMANTIC | | SEMANTIC | | SEMANTIC | | SEMANTIC |
| NCI1.0 | RF_DISCOVER_CMD/RSP | | DISCOVERY | Status_OK | | READY_STARTE | | READY_STARTE | | READY_STARTE | | READY_STARTE | | READY_STARTE | | READY_STARTE |
| NCI1.0 | RF_DISCOVER_SELECT_CMD/RSP | | | SEMANTIC | | SEMANTIC | | SEMANTIC | | Status_OK | | SEMANTIC | | SEMANTIC | | SEMANTIC |
| NCI1.0 | RF_DEACTIVATE_CMD/RSP | Idle_Mode | | SEMANTIC | IDLE | Status_OK | IDLE | Status_OK | IDLE | Status_OK | IDLE | Status_OK | IDLE | Status_OK | IDLE | Status_OK |
| | | Sleep_Mode / Frame RF Interface | | SEMANTIC | IDLE | Status_OK | IDLE | Status_OK | IDLE | Status_OK | W4_HOST_SELEC | Status_OK | | REJECTED | | REJECTED |
| | | Sleep_Mode / ISO-DEP RF Interface | | SEMANTIC | IDLE | Status_OK | IDLE | Status_OK | IDLE | Status_OK | W4_HOST_SELEC | Status_OK | | REJECTED | | REJECTED |
| | | SleepAF_Mode / Frame RF Interface | | SEMANTIC | IDLE | Status_OK | IDLE | Status_OK | IDLE | Status_OK | | REJECTED | | REJECTED | | SEMANTIC |
| | | SleepAF_Mode / NFC-DEP RF Interface | | SEMANTIC | IDLE | Status_OK | IDLE | Status_OK | IDLE | Status_OK | W4_HOST_SELEC | Status_OK | | REJECTED | | SEMANTIC |
| | | Discovery | | SEMANTIC | IDLE | Status_OK | IDLE | Status_OK | IDLE | Status_OK | DISCOVERY | Status_OK | | REJECTED | | REJECTED |
| NCI1.0 | RF_T3T_POLLING_CMD/RSP | Frame RF Interface | | SEMANTIC | | SEMANTIC | | SEMANTIC | | SEMANTIC | | Status_OK | | SEMANTIC | | SEMANTIC |
| NCI1.0 | RF_PARAMETER_UPDATE_CMD/RSP | Frame RF Interface | | SEMANTIC | | SEMANTIC | | SEMANTIC | | SEMANTIC | | Status_OK | | SEMANTIC | | SEMANTIC |
| NCI2.0 | RF_INTF_EXT_START | Frame RF Interface | | SEMANTIC | | SEMANTIC | | SEMANTIC | | SEMANTIC | | REJECTED | | REJECTED | | SEMANTIC |
| NCI2.0 | RF_INTF_EXT_STOP | Frame RF Interface | | SEMANTIC | | SEMANTIC | | SEMANTIC | | SEMANTIC | | REJECTED | | REJECTED | | SEMANTIC |
| NCI2.0 | RF_INTF_AGG_ABORT | Frame RF Interface | | SEMANTIC | | SEMANTIC | | SEMANTIC | | SEMANTIC | | REJECTED | | REJECTED | | SEMANTIC |
| NCI1.0 | NFCEE_DISCOVER_CMD/RSP | | | Status_OK | | Status_OK | | Status_OK | | Status_OK | | Status_OK | | Status_OK | | Status_OK |
| NCI1.0 | NFCEE_MODE_SET_CMD/RSP | | | Status_OK | | Status_OK | | Status_OK | | Status_OK | | Status_OK | | Status_OK | | Status_OK |
| NCI2.0 | NFCEE_POWER_AND_LINK_CONTROL | | | Status_OK | | Status_OK | | Status_OK | | Status_OK | | Status_OK | | Status_OK | | Status_OK |
| NXP | CORE_SET_POWER_MODE_CMD/RSP | | | Status_OK | | SEMANTIC | | SEMANTIC | | SEMANTIC | | SEMANTIC | | SEMANTIC | | SEMANTIC |
| NXP | NCI_PROPRIETARY_ACT_CMD/RSP | | | Status_OK | | SEMANTIC | | SEMANTIC | | SEMANTIC | | SEMANTIC | | SEMANTIC | | SEMANTIC |
| NXP | TEST_SWP_CMD/RSP | | | Status_OK | | SEMANTIC | | SEMANTIC | | SEMANTIC | | SEMANTIC | | SEMANTIC | | SEMANTIC |
| NXP | TEST_PRBS_CMD/RSP | | | Status_OK | | SEMANTIC | | SEMANTIC | | SEMANTIC | | SEMANTIC | | SEMANTIC | | SEMANTIC |

**Figure 29. Command/response versus the current state of the NCI RF state machine**

| | | | CURRENT STATE | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | RFST_IDLE | | RFST_DISCOVERY | | RFST_W4_ALL_DISC. | | RFST_W4_HOST_SELECT | | RFST_POLL_ACTIVE | | RFST_LISTEN_ACTIVE | | RFST_LISTEN_SLEEP | |
| Source | Control Message | parameter | Next state | NTF expected in this state ? | Next state | NTF expected in this state ? | Next state | NTF expected in this state ? | Next state | NTF expected in this state ? | Next state | NTF expected in this state ? | Next state | NTF expected in this state ? | Next state | NTF expected in this state ? |
| NCI2.0 | CORE_RESET_NTF | | | Yes | IDLE | Yes | IDLE | Yes | IDLE | Yes | IDLE | Yes | IDLE | Yes | IDLE | Yes |
| NCI1.0 | RF_FIELD_INFO_NTF | | | Yes | | Yes | | No | | No | | No | | Yes | | Yes |
| NCI1.0 | CORE_CONN_CREDITS_NTF | Dynamic logical connection | | Yes | | No | | No | | No | | No | | No | | No |
| NCI1.0 | | RF static logical connection | | No | | No | | No | | No | | Yes | | Yes | | Yes |
| NCI2.0 | | NFCEE logical connection | | Yes | | Yes | | Yes | | Yes | | Yes | | Yes | | Yes |
| NCI1.0 | CORE_GENERIC_ERROR_NTF | NCI1.0 Status | | No | | Yes | | No | | Yes | | No | | No | | No |
| NXP | | NXP Status | | Yes | | Yes | | Yes | | Yes | | Yes | | Yes | | Yes |
| NCI1.0 | CORE_INTERFACE_ERROR_NTF | Loop-Back logical connection | | Yes | | No | | No | | No | | No | | No | | No |
| | | RF static logical connection | | No | | No | | No | | No | | Yes | | Yes | | No |
| | | NFCEE logical connection | | Yes | | Yes | | Yes | | Yes | | Yes | | Yes | | Yes |
| NCI1.0 | RF_DISCOVER_NTF | Notif. Type = 2 | | No | W4_ALL_DISC | Yes | | Yes | | No | | No | | No | | No |
| | | Notif. Type = 0/1 | | No | | No | W4_HOST_SELEC | Yes | | No | | No | | No | | No |
| NCI1.0 | RF_DEACTIVATE_NTF | Idle_Mode | | No | | No | | No | | No | IDLE | Yes | IDLE | Yes | | No |
| | | Sleep_Mode | | No | | No | | No | | No | W4_HOST_SELEC | Yes | LISTEN_SLEEP | Yes | | No |
| | | SleepAF_Mode | | No | | No | | No | | No | W4_HOST_SELEC | Yes | LISTEN_SLEEP | Yes | | No |
| | | Discovery | | No | | No | | No | | No | DISCOVERY | Yes | DISCOVERY | Yes | DISCOVERY | Yes |
| NCI1.0 | RF_INTF_ACTIVATED_NTF | Poll Mode | | No | POLL_ACTIVE | Yes | | No | POLL_ACTIVE | Yes | | No | | No | | No |
| | | Listen Mode | | No | LISTEN_ACTIVE | Yes | | No | POLL_ACTIVE | Yes | | No | | No | LISTEN_ACTIVE | Yes |
| NCI1.0 | RF_NFCEE_ACTION_NTF | | | No | | No | | No | | No | | No | | Yes | | No |
| NCI1.0 | RF_T3T_POLLING_NTF | | | No | | No | | No | | No | | Yes | | No | | No |
| NCI1.0 | NFCEE_DISCOVER_NTF | | | Yes | | Yes | | Yes | | Yes | | Yes | | Yes | | Yes |
| NCI1.0 | NFCEE_DISCOVERY_REQ_NTF | | | Yes | | Yes | | Yes | | Yes | | Yes | | Yes | | Yes |
| NCI2.0 | NFCEE_MODE_SET_NTF | | | Yes | | Yes | | Yes | | Yes | | Yes | | Yes | | Yes |
| NCI2.0 | NFCEE_STATUS_NTF | | | Yes | | Yes | | Yes | | Yes | | Yes | | Yes | | Yes |
| NXP | PLL_UNLOCKED_NTF | | | No | | No | | Yes | | Yes | | Yes | | No | | No |
| NXP | SYSTEM_WTX_NTF | | | No | | No | | No | | No | | Yes | | No | | No |

**Figure 30. NTFs versus the current state of the NCI RF state machine**

To secure the implementation of the “atomic behavior” of `CORE_RESET_CMD` and `CORE_INIT_CMD`, and to handle the mapping of RF protocol to RF interface with `RF_DISCOVER_MAP_CMD`, PN722X NFCC defines additional states to the RF state machine defined in [NCI]. Figure 31 illustrates the additional states linked to the [NCI]-defined RFST_IDLE.

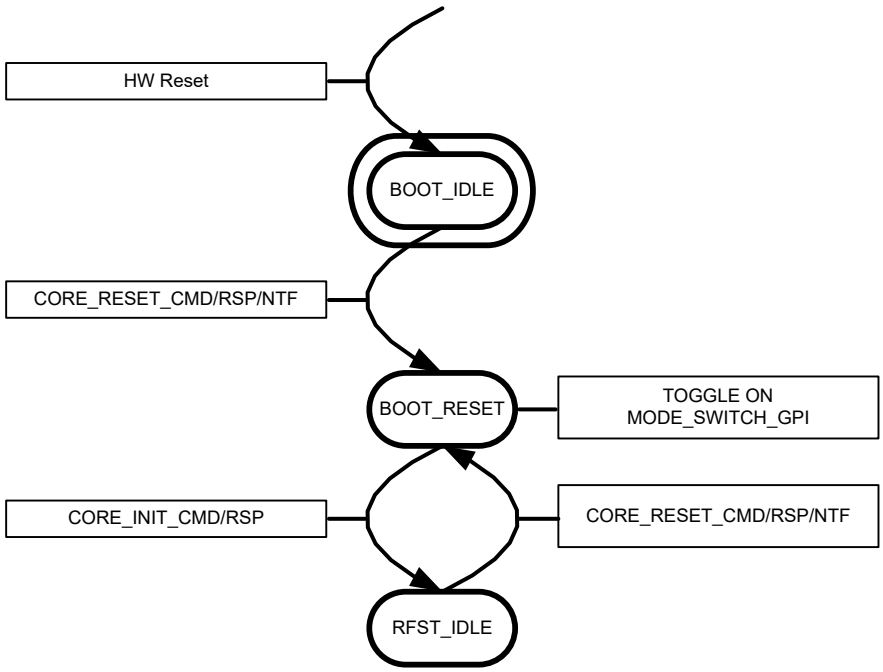



**Figure 31. States added to the [NCI] state machine**

### 7.3.5 [PN722X NFCC-NCI] extensions to [NCI] configuration parameters

[NCI] lists the parameters required to set up the RF discovery. PN722X NFCC requires additional parameters, for instance to configure some RF protocols which are not supported by [NCI], or to configure the power and clock management.

Table 20 lists the sets of additional parameters.

**Table 20. Overview of additional configuration parameters**

| Section | Feature to configure | Comment |
|---|---|---|
| Section 13.1 | System | Parameters allowing the DH to configure the System: Clock management, MIFARE Classic Keys handling… |
| Section 13.2 | RF Discovery | Parameters allowing the DH to configure the Discovery activity (Discovery profile between NFC Forum and EMVCo,….) |
| Section 13.3 | Contactless Interface | Parameters allowing the DH to configure all internal HW settings in the contactless interface (CLIF) as well as specific RF platform settings (LPCD, DPC) |

### 7.3.6 [PN722X NFCC-NCI] extensions to [NCI] proprietary parameters space

[NCI] defines a parameter space with a size of 255 parameters, in which around 96 tags are allocated for proprietary parameters:

**Table 21. Parameter space**

| Parameters space subsections | Tag |
|---|---|
| Assigned and reserved in [NCI] | 0x00-0x9F |
| Reserved for Proprietary Use | 0xA0-0xFE |
| RFU (Reserved for Extension) | 0xFF |

Regarding the PN722X NFCC needs, this reserved area is not sufficient. To extend this space, the solution chosen is to define a space of tags coded on 16 bits, instead of 8 bits. These extended tags always start with the value 0xA0, 0xA1, or 0xA2, which is the first value available in the Proprietary range. This allows adding 256 new parameters.

***Note:*** *If this is not sufficient, 16-bit tag values starting by 0xA3 may be used.*

**Table 22. Extended TLV for proprietary parameters**

| Payload | Field | Length | Description |
|---|---|---|---|
| m+3 Octets | Tag = 0xA0XX, 0xA1XX or 0xA2XX | 2 Octets | Extended tag identifier. |
| | Len | 1 Octet | The length of Val (m). |
| | Val | m Octets | Value of the configuration parameter. |

Figure 32 illustrates the comparison between regular and extended TLVs.

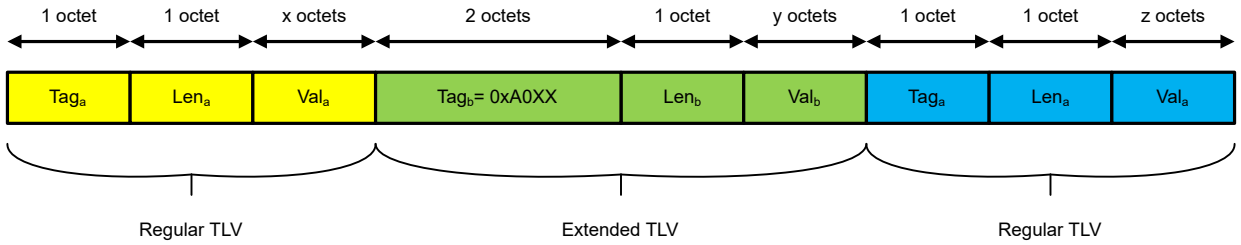



**Figure 32. Comparison between regular and extended TLVs**

### 7.3.7 [PN722X NFCC-NCI] extensions to [NCI] status codes

[NCI] defines a set of standard Status Codes in [NCI_Table1] (see Section 3).

NXP has extended this set of status codes with the following values:

**Table 23. Proprietary Status Codes**

| Status code | Description | Used in | Error recovery |
|---|---|---|---|
| 0xE3 | STATUS_PMU_TXLDO_ OVERCURRENT | CORE_GENERIC_ ERROR_NTF | Check VDDPA connection.<br>Restart the NFC service. If issue still present, Contact NXP support |
| 0xE4 | STATUS_EMVCO_PCD_ COLLISION | CORE_GENERIC_ ERROR_NTF | Polling sequence will be restarted in EMVCo Mode |
| 0xE7 | STATUS_GPADC_ERROR | CORE_GENERIC_ ERROR_NTF | Restart the NFC service |
| 0xEE | STATUS_NCI_USER_AREA_ CRC_ERROR | CORE_GENERIC_ ERROR_NTF | No action as automatically recovered by NFCC. |
| 0xF0 | STATUS_NCI_USER_AREA_ RECOVERY_ERROR | CORE_GENERIC_ ERROR_NTF | Restart the load of customer-specific parameters. As default config values will be applied. |
| 0xF1 | EXTERNAL RF ERROR | CORE_GENERIC_ ERROR_NTF | Returned during Field ON when external RF is present |
| 0xF2 | TX LDO ERROR | CORE_GENERIC_ ERROR_NTF | Field ON is not successful due to LX LDO Error |
| 0xF3 | INVALID LOAD RF CONFIGURA TION | SWITCH_RF_FIELD_RSP | Load RF Configuration CMD is not executed with a valid TX and RX Configuration |
| 0xF4 | SECURE_EE_CRC_ERROR | CORE_GENERIC_ ERROR_NTF | Customer to perform Secure FW update procedure with FW containing User EE area. Update the customer settings in User EE area once again using Prop Setconfig extension using 0xA2. |

Except error code related to EMVCo PCD collision that may happen in EMVCo mode. The other error codes can only occur in rare error cases and should be reported to customer support.

### 7.3.8 [PN722X NFCC-NCI] extensions to [NCI] reason code in CORE_RESET_NTF

[NCI] defines a set of standard reason codes in the CORE_RESET_NTF. Refer to [NCI_Table9] (see Section 3).

NXP has extended the set of reason codes with the following values shown in Table 24.

**Table 24. Proprietary reason codes in CORE_RESET_NTF**

| Reason code | Description |
|---|---|
| 0xA0 | An assert has triggered PN722X NFCC reset/reboot |
| 0xA1 | An over temperature has triggered the reset of PN722X NFCC |
| 0xA2 | RFU |
| 0xA3 | The watchdog has triggered PN722X NFCC reset/reboot |
| 0xA4 | RFU |
| 0xA5 | RFU |
| 0xA6 | RFU |
| 0xA7 | HIF1_I2C, HIF_SPI - After POR, any NCI packet other than CORE_RESET_CMD, OR any invalid HDLL packet (i.e CRC NOK) is received by PN722X.<br>HIF2_I2C - Any Packet other than CORE_RESET_CMD is received by PN722X.<br>***Note:*** *PN722X can receive HDLL CMDs before 5sec after POR in HIF1_I2C and HIF_SPI Interface.* |
| 0xA8 | Boot due to Switch Mode reset to switch to NFC Forum Mode |
| 0xA9 | Boot due to Switch Mode reset to switch to EMVCo Mode |

***Note:*** *When the 0xA0 reason code is used, the CORE_RESET_NTF is out of [NCI] compliance.*

Indeed, PN722X appends one parameter at the end of the CORE_RESET_NTF, to provide some information for debug purposes. The CORE_RESET_NTF format is then:

**Table 25. CORE_RESET_NTF when reason code = 0xA0 is used**

| Payload field(s) | Length | Description | Default |
|---|---|---|---|
| Reason Code | 1 Octet | 0xA0: NXP proprietary | 0xA0 |
| Config. Status | 1 Octet | See [NCI] | |

PN722X follows the sequence below when an over temperature is detected (Reason Code = 0xA1):

- PN722X turns off the RF Field and waits until any RSP/NTF pending to be read from PN722X by DH and next enters into Standby.
- PN722X waits until the chip temperature comes down to an internal threshold.
- When the internal temperature is low enough, PN722X reboots and sends a CORE_RESET_NTF (0xA1) to inform the DH that an over temperature event occurred.
- DH may continue with usual NFC Service initialization.

### 7.3.9 [PN722X NFCC-NCI] extensions to [NCI] reason code in NFCEE_STATUS_NTF

[NCI] defines a set of standard reason codes in the NFCEE_STATUS_NTF supported by PN722X. Refer to [NCI].

NXP has extended this set of reason codes with the values listed in Table 26.

**Table 26. Proprietary Reason Codes in NFCEE_STATUS_NTF**

| Reason code | Description |
|---|---|
| 0x92<br>0x93 | NFCC got a timeout during CT activation request |
| 0x98 | NFCC has detected an issue during CT activation |
| 0x99 | NFCC has detected an issue during CT deactivation |

***Note:*** *When any NFCEE_STATUS_NTF with a proprietary reason code is used, DH considers CT as unresponsive and applies the same recovery procedure as UNRECOVERABLE_ERROR: NFCEE_DISCOVER_CMD followed by the corresponding NFCEE_MODE_SET_CMD.*

### 7.3.10 [PN722X NFCC-NCI] extensions to [NCI] PLL unlocked notification

NXP has created a new notification which is sent in case the PLL cannot lock on the incoming system clock; here is the description of this notification:

**Table 27. PLL_UNLOCKED_NTF**

| GID | OID | Parameter number | Description |
|---|---|---|---|
| 0001b | 0x21 | 0 | Notification sent in case PLL does not lock to system clock |

### 7.3.11 [PN722X NFCC-NCI] extensions to [NCI] TxLDO ERROR Notification

NXP has created a new notification which is sent to notify the DH when TxLDO driver cannot be started successfully.

**Table 28. TxLDO_ERROR_NTF**

| GID | OID | Parameter number | Description | |
|---|---|---|---|---|
| 0001b | 0x23 | 0 | TxLDO driver cannot be started | |

### 7.3.12 [PN722X NFCC-NCI] extension to [NCI] FLUSH_SRAM_AO_TO_FLASH

The PN722X NFCC contains a RAM area which remains powered on even in standby state.

This area is used to store temporarily some parameters which can be dynamically updated by the DH.

At boot time, the RAM always is initialized with the content of its mirrored FLASH area.

This concerns the following parameters: NCI ‘official’ registers, routing table.

Synchronization is done thanks to NCI proprietary command that can be used by the DH to force a copy of the RAM always on area to it's mirrored FLASH area. DH shall only use this command when in RFST_IDLE.

NFCC expects that this command is sent by the DH

- at the end of NFCC initialization before sending NCI_DISCOVER_CMD
- before entering into a low-power state

**Table 29. FLUSH_SRAM_AO_TO_FLASH_CMD**

| GID | OID | Parameter number | Description |
|---|---|---|---|
| 1111b | 0x21 | 0 | Command to save the content of the SRAM Always ON to FLASH. |

**Table 30. FLUSH_SRAM_AO_TO_FLASH_RSP**

| GID | OID | Parameter number | Description |
|---|---|---|---|
| 1111b | 0x21 | 1 | The NFCC writes the SRAM Always On content into flash if content of both areas is different. Then it returns STATUS_OK. |

**Table 31. FLUSH_SRAM_AO_TO_FLASH_RSP parameters**

| Payload Field(s) | Length | Value/Description | |
|---|---|---|---|
| STATUS | 1 Octet | 0x00 | STATUS_OK |
| | | 0x09 | PH_NCI_STATUS_INVALID_PARAM |
| | | 0x06 | STATUS_SEMANTIC_ERROR |
| | | Others | RFU |

# 8 NFCEEs management

## 8.1 Overview

In addition to the NFCEE hosted by the DH (DH-NFCEE), the PN722X NFCC can be connected to 3 different Contact devices via TDAs:

1. Two SAMs can be connected over 2 TDAs.
2. One CT Payment card using one TDA.

Here are the features offered by the PN722X NFCC over the different NFCEE:

**Table 32. Features overview**

| Mode | Features | DH-NFCEE | NFCEE over TDA |
|---|---|---|---|
| Card Emulation | ISO-DEP NFC-A | ☑ | ☒ |
| | ISO-DEP NFC-B | ☒ | ☒ |
| | FeliCa NFC-F | ☒ | ☒ |
| | MIFARE Classic NFC-A | ☒ | ☒ |
| | MIFARE DESFire NFC-A | ☒ | ☒ |
| Reader/Writer | T2T, MIFARE Classic NFC-A | ☑ | ☒ |
| | ISO-DEP NFC-A | ☑ | ☒ |
| | MIFARE DESFire NFC-A | ☒ | ☒ |
| | ISO-DEP NFC-B | ☑ | ☒ |
| | FeliCa, T3T | ☑ | ☒ |

☑ ... covered

☒ … not covered

The enabling/disabling of the NFCEEs is controlled by the DH thanks to the standard [NCI] command NFCEE_MODE_SET_CMD. CT support using TDA is not used for any RF Interface use case, but instead will only be used to communicate with SAM or CT Payment card.

## 8.2 NFCEE power supply concept

PN722X will not control power supply to any TDA device.

## 8.3 NFCEE reset concept

As described in Section 8.3, PN722X does not control the power supply to any TDAs. Therefore, it is not possible to reset any TDA device.

## 8.4 NFCEE discovery

PN722X supports the connection to three TDA through IOAUX and CLKAUX interfaces. PN722X can also receive ISO7816 APDUs from the DH.

PN722X sends one NFCEE_DISCOVER_NTF notifications for each CT device connected using TDA which is outside the HCI network. The notifications inform the DH about the NFCEE ID assigned by PN722X. The NFCEE ID is used later on to communicate with ISO7816 APDUs and to Enable/Disable the NFCEE with the NFCEE_MODE_SET_CMD.

[NCI] defines a new way of managing NFCEE that impacts both the NCI API and how it is used.

The NFCEE management is composed of five functions:

- NFCEE discovery:
  - How the DH detects some NFCEEs and gets information about their state, power mode and capabilities.
- NFCEE activation:
  - How the DH can enable or disable a specific NFCEE.
- NFCEE identity mapping and communication:
  - [NCI] supports NFCEE from various networks and is able to map an NFCEE network-specific number with an NCI number.
- NFCEE state management:
  - [NCI] defines an NFCEE state transition list to manage the state of an NFCEE.
  - Some NFCEE require a non-predictable time for boot-up operation and initialization. With this NCI interface, the DH knows the status of all NFCEE in real time.
- NFCEE power management:
  - PN722X does not support NFCEE power management.

The DH oversees the NFCEE discovery process. The NFCC provides in the response the number of NFCEE physically or theoretically present whether the NFCEEs belong to the HCI network or not. For each NFCEE present, an NFCEE discover notification is sent with the current state of the NFCEE, the supported protocol, and the type of NFCEE. PN722X does not support any HCI NFCEE.

For NFCEE CT, the logical channel connection must be created dynamically.

The NFCEE mode set procedure is used to activate the link interface of a particular NFCEE. This triggers the activation procedure of the NFCEE. If the NFCEE was already enabled, the status message will go as success and a notification will also be sent as success. NFCEE CT Payment card insertion is highly dynamic in nature, once inserted a discovery notification is sent to the DH with the new NFCEE presence. Then DH should send the Mode Set Enable command to activate the NFCEE. After the activation a logical connection can be established to exchange the APDUs.

### 8.4.1 NFCEE discovery command

NFCEE Discovery module which is no longer supported by the NFCC but by the DH. As such, the DH oversees discovering the presence of the NFCEEs attached to the NFCC. After the NFCEE discovery command, the NFCC reacts by providing the list of visible NFCEEs already enabled or not. Apart from NFCEE Payment card, if some NFCEE (for example SAM) are attached after the NFCEE discovery request, the NFCC no longer sends notifications. Apart from NFCEE Payment card, if an NFCEE (for example SAM) is removed after an NFCEE discovery request, NFCC sends a removal notification. Then the DH can initiate a new NFCEE discovery or wait for the discovery notification about a card insertion.

Upon receiving an NFCEE discover request, the NFCC sends an NFCEE discovery response indicating the number of NFCEE among the following: CT1, CT2 and CT3.

After the NFCEE discover response has been sent, the NFCC sends as many NFCEE discovery notification as the number of NFCEE interfaces available. For each NFCEE, the NFCC indicates if the NFCEE is enabled, disabled, or unresponsive (if the NFCEE is not present in the slot), the number of protocol information supported, NFCEE specific information, and NFCEE power supply mode.

### 8.4.2 NFCEE identity mapping and communication

The DH cannot use the static logical channel to access the NFCEEs outside HCI network. Instead, a dynamic logical channel must be created. Table 33 shows how NFCEE CT identifiers are mapped with an NCI number.

**Table 33. NFCEE Identifiers mapping between NCI**

| NFCEE | Interface | Conn ID | NCI ID | Chip Select | | |
|---|---|---|---|---|---|---|
| | | | | GPIO0 | GPIO1 | DWL_REQ |
| DH_ NFCEE | Host IF | NA | 0x00 | NA | NA | NA |
| CT1 | IOAUX | 0x0A | 0x20 | 1 | 0 | 0 |
| CT2 | IOAUX | 0x0B | 0x21 | 0 | 1 | 0 |
| CT3 | IOAUX | 0x0C | 0x22 | 0 | 0 | 1 |

The PN7220 GPIO used to connect the TDA chip-select defines the CT slot numbering. PN7220 configures the number of active CT slots connected to PN7220. Based on this configuration, PN7220 uses TDA to check the CT card presence.

NFCEE CT is out of the HCI network. It supports [7816-4] protocol and requires a dynamic logical connection.

### 8.4.3 NFCEE state management

After having received all NFCEE discover notifications, the DH can initiate NFCEE_MODE_SET_CMD *enable* command in order to enable a specific NFCEE. At the start of the discovery, the NFCEE is in disabled state or unresponsive state (if no card is present in the slots). If the NFCEE is in unresponsive state, it goes to disabled state only when a card is inserted and a discover notification is sent to the DH.

If the NFCEE was declared as disabled, the successful completion of the NFCEE mode set procedure switches the NFCEE state into enable state. Otherwise, the DH can disable an NFCEE by sending the NFCEE_MODE_SET_CMD *disable* command.

When NFCEE mode is set to *enable*, the NFCEE starts activating the card. The NFCEE_MODE_SET_NTF is sent after the NFCEE has been enabled or disabled. Once the card activation is complete, a discover notification is also sent with the ATR information and enabled state.

When NFCEE is being used, some transmission errors might occur, or the NFCEE might be removed. Then the NFCEE status *unrecoverable error* notification is sent to the DH. In such scenario, the NFCEE is set in *unresponsive* state and can only be retrieved and used after an NFCEE discover request from the DH.

Deactivation of the NFCEE is carried out when the DH sends a MODE SET command with disable option. The CT payment card removal deactivates the card automatically and the NFCEE state becomes unresponsive until the next insertion of the card. A discover notification is also sent to the DH with the state as unresponsive.

### 8.4.4 NFCEE power management

The NFCEE_DISCOVER_NTF has been extended in [NCI] to indicate for each NFCEE whether the NFCC is able to control the power supply of this NFCEE or not.

The DH uses this information to exchange data with the NFCEE. If NFCC controls the NFCEE power supply, the DH can force NFCC to always maintain the power supply or the communication link with this NFCEE, using NCI command `NFCEE_POWER AND_LINK_CTRL`. This API has no persistent effect.

***Note:*** *PN722X does not control the power supply to TDAs.*

In case NFCC does not control the NFCEE power supply (this is the case for CT), the DH must use other means to ensure that the NFCEE is always power supplied.

### 8.4.5 NFCEE discovery for one NFCEE CT using TDA

Figure 33 describes the NFCEE CT discovery sequence when the NFCEE has not yet performed its CT initialization. The assumption is that the NFCEE CT card is inserted dynamically in payment card insertion use case.

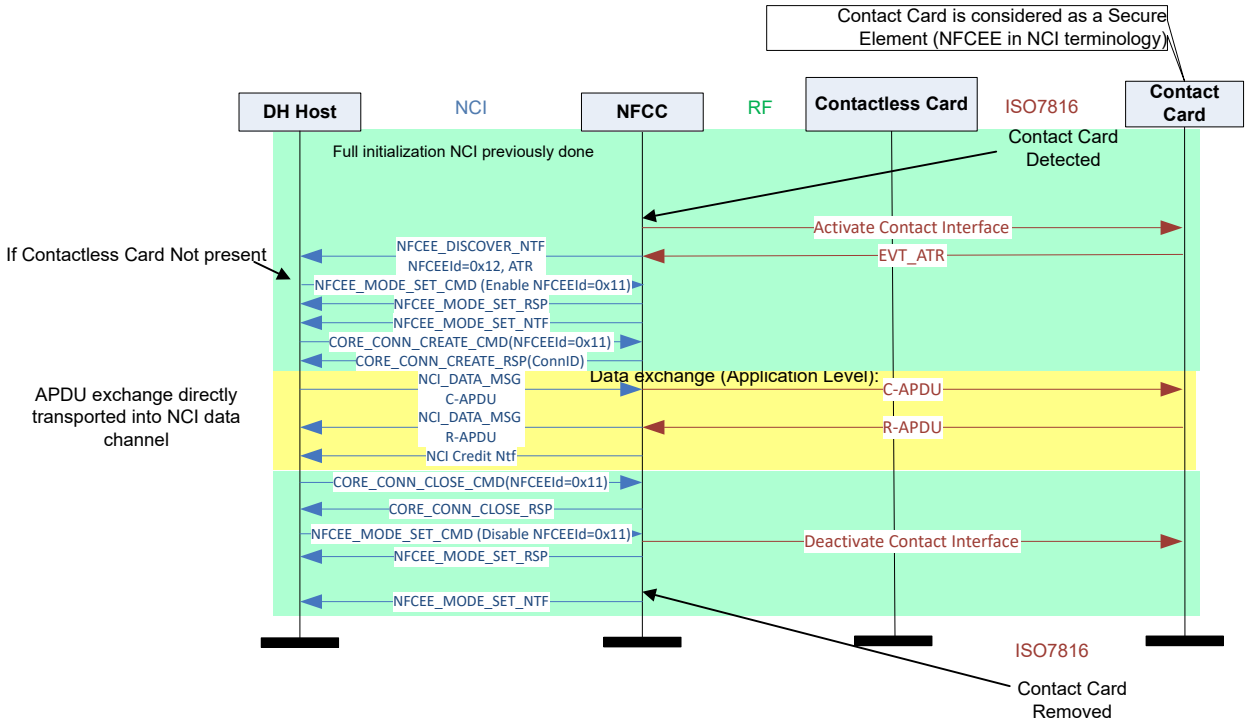



**Figure 33. NFCEE discovery sequence when the NFCEE CT card is inserted dynamically**

### 8.4.6 Format of the NFCEE_DISCOVER_NTF for the CT device

Table 34 shows the parameters of NFCEE discovery sequences when NFCEE_DISCOVER_NTF is sent to inform the DH about the NFCEEs connected outside the HCI network.

**Table 34. NFCEE_DISCOVER_NTF for each CT NFCEE outside the HCI network**

| Payload Field(s) | Length | Value/Description |
|---|---|---|
| NFCEE ID | 1 Octet | NFCEE ID<br>• 0x20 for CT1<br>• 0x21 for CT2<br>• 0x22 for CT3 |
| NFCEE Status | 1 Octet | • 0x00 if NFCEE connected and enabled<br>• 0x01 if NFCEE connected and disabled<br>• 0x02 if NFCEE is unresponsive or card not inserted |
| Number of Protocol Information Entries | 1 Octet | 1 => for CT NFCEE |
| Supported NFCEE Protocols [0..n] | n Octet | 0x00 => APDU mode for CT NFCEE |
| Number of NFCEE Information TLVs | 1 Octet | 1 |
| NFCEE Information | Length of ATR Bytes + 2 Octets | Type 0x01 (ATR Bytes)<br>Length: n Bytes<br>Value: ATR Bytes of length n bytes. |
| NFCEE Power Supply | 1 Octet | NFCC has no control of power supply for CT devices<br>• 0x00 for any CT device |

### 8.4.7 RF_NFCEE_ACTION_NTF and RF_NFCEE_DISCOVERY_REQ_NTF

PN722X does not support RF use case with NFCEE CT interface.

# 9 Initialization and operation configuration

## 9.1 Reset/initialization

[NCI] defines Reset/Init sequence based on two commands:

- CORE_RESET_CMD
- CORE_INIT_CMD

The DH calls the two commands in an “atomic” way: there cannot be any other command in-between and the PN722X NFCC operation cannot start any operation (Reader/Writer, Card Emulation, Combined modes etc.) if it does not first receive these two commands.

[NCI] defines two modes for the reset command: Keep Configuration and Reset Configuration. shows the differences between the two reset modes.

**Table 35. Comparison of the two reset modes**

| Features | Reset configuration | Keep configuration |
|---|---|---|
| CPU reboot | Yes | Yes |
| Listen Mode Routing table[1] | Lost | Kept |
| NCI Configuration parameters | Back to default | Kept |
| Proprietary Configuration parameters | Lost[2] | Kept |
| Interface Mapping Table | Lost | Kept |
| Discovery activity | Lost | Lost |
| Dynamic connections | Lost | Lost |

[1] Listen Mode Routing table is updated in FLASH DATA using RF_SET_LISTEN_MODE_ROUTING_CMD. DO not use this configuration frequently.
[2] RF Discovery configuration is not updated to either FLASH DATA or RAM-AO.

If the DH sends a CORE_RESET_CMD while PN722X NFCC has already indicated that it has some data available to be read by the DH (IRQ pin activated), the DH has to first read the data available from PN722X NFCC before it can get the CORE_RESET_RSP. The reason is that the NCI output buffer in PN722X NFCC needs to be flushed before PN722X NFCC can apply a Reset and then send the CORE_RESET_RSP.

## 9.2 Manufacturer-specific information in [NCI2.2] CORE_RESET_NTF

The NCI notification `CORE_RESET_NTF` for [NCI2.2] includes a 5-byte field for Manufacturer Specific Information. Table 36 shows the meaning of the 5 Bytes and the conditions increment or update the value.

**Table 36. Manufacturer-specific information in CORE_RESET_NTF**

| Byte | Meaning | Condition to increment | Value for PN722X |
|---|---|---|---|
| 0 | Model ID | Specific Model ID (variant of the hardware platform) | 0x2X |
| 1 | Hardware Version number | Silicon identifier | 0x53 |
| 2 | ROM Code Version number | ROM Code identifier | 0x03 |
| 3 | FLASH Major version | Firmware major branch | 0xYY |
| 4 | FLASH Minor version | New firmware, solving bugs on existing features. | 0xZZ |

### 9.2.1 Proprietary command to enable proprietary extensions

It is visible on the previous flowchart that NXP has introduced a proprietary command sent by the DH to enable the proprietary extensions to [NCI], which are available in the PN722X NFCC. So, when the PN722X NFCC receives this command NCI_PROPRIETARY_ACT_CMD, it knows that the DH is aware of the proprietary extensions and may therefore send proprietary notifications (see the list in Table 17).

Here is the description of this command:

**Table 37. NCI_PROPRIETARY_ACT_CMD**

| GID | OID | Numbers of parameter(s) | Description |
|---|---|---|---|
| 1111b | 0x02 | 0 | DH informs the PN722X NFCC that it knows the proprietary extensions. |

**Table 38. NCI_PROPRIETARY_ACT_RSP**

| GID | OID | Numbers of parameter(s) | Description |
|---|---|---|---|
| 1111b | 0x02 | 2 | See table below for details |

**Table 39. NCI_PROPRIETARY_ACT_RSP parameters**

| Payload Field(s) | Length | Value/description |
|---|---|---|
| STATUS | 1 Octet | One of the following Status codes, as defined in [NCI_Table1]<br>0x00 – STATUS_OK<br>0x03 – STATUS_FAILED<br>Others – Forbidden |
| FW_Build_Number | 4 Octets | NXP internal firmware build number |

## 9.3 PLL input clock management

The PN722X NFCC clock source can be:

- One 27.12 MHz quartz, or
- One clock signal available on the platform to which PN722X NFCC is connected. A PLL inside PN722X NFCC converts the input clock signal into an internal 27.12 MHz signal that generates the RF carrier. The input clock frequency has to be one of the predefined set of external input frequencies: 19.2 MHz, 24 MHz, 32 MHz and 48 MHz.

The DH configures the EEPROM parameter `RF_CLOCK_CFG` to set the clock source used in the current application. For `EEPROM_CFG CMD` used to update the PLL configuration defined in the data sheet, refer to Section 13.1.

**Table 40. Clock sources supported**

| Name | Description |
|---|---|
| Xtal | To be selected when a 27.12 MHz quartz is used as a clock source |
| PLL | To be selected when an input clock is provided to PN722X NFCC, with a frequency equal to either 24 MHz, 32 MHz and 48 MHz. |

To optimize the system power consumption, it may be required to switch OFF the PLL input clock when the PN722X NFCC does not have to generate the 13.56 MHz RF carrier, or to answer to a reader using ALM.

PN722X NFCC assumes that the PLL input clock is on and stable after a programmable time-out. A specific parameter of PN722X is used to configure the time-out. For `EEPROM_CFG CMD` used to update the PLL configuration defined in the data sheet, refer to Section 13.1.

## 9.4 Protection mechanism against corrupted memory area (anti-tearing)

The PN722X NFCC tearing proof memory areas are:

- The User Area: standard Flash data memory area with CRC and with a secondary area to duplicate its content.
- The Mirrored User Area: Subset of the mirrored Flash Data area with CRC and with a secondary area to duplicate its content.

The mirrored Flash Data area is loaded in RAM always ON memory after power-on reset and only the RAM always ON content is changed dynamically by the NFCC during runtime.

The RAM always ON content is written back to Flash Data area only when the host sends a specific NCI command `FLUSH_SRAM_AO_TO_FLASH_CMD` (see Table 29).

Each time the NFCC processes NCI `CORE_INIT` command, it checks if the CRC of each protect area is valid. If the CRC is invalid, it gets overwritten by the content of the secondary area. In this case, a generic error notification is sent to inform the host: `STATUS_NCI_USER_AREA_CRC_ERROR` or `STATUS_NCI_USER_AREA_RECOVERY_ERROR` (see Table 23).

# 10 Poll side: reader/writer mode

## 10.1 Reader/Writer hosted by the DH

### 10.1.1 T2T, MIFARE Ultralight, MIFARE Classic and MIFARE Plus tags

***Note:*** *All the Tags/Cards in this category are based on NFC-A technology, but they do not support the ISO-DEP Protocol.*

MIFARE Plus cards support the ISO-DEP protocol, but only when they are configured in Security Level3.

#### 10.1.1.1 Access through the [NCI] Frame RF interface

[NCI] allows the data exchange with tags T2T using the Frame RF Interface.

Most of the commands of the MIFARE Classic and MIFARE Plus can also be mapped on the Frame RF Interface. But NXP decided to use a separate RF interface (TAG-CMD, see Section 10.1.1.2) because the MIFARE Classic Authenticate command is split in two steps and has a tight response timeout (1 ms) which can hardly be monitored by the DH through the NFCC.

Table 41 lists the tags/card based on technology NFC-A which can be accessed through the Frame RF interface.

**Table 41. Tag/cards accessible over the [NCI] Frame RF Interface**

| Tag/card | Access through the Frame RF Interface |
|---|---|
| T2T | ☑ |
| MIFARE Ultralight, Ultralight C | ☑ |
| MIFARE Classic | ☒ |
| MIFARE Plus for Security levels 1 and 2 | ☒ |

☑ ... covered

☒ … not covered

Table 42 lists the commands and configuration parameters to prepare the Reader/Writer Mode for T2T through the Frame RF Interface.

**Table 42. Configuration sequence for Reader/Writer of T2T through the Frame RF Interface**

| Command | Main Parameters | Values |
|---|---|---|
| RF_DISCOVER_MAP_CMD[1] | RF Protocol | PROTOCOL_T2T |
| | Mode | Poll |
| | RF Interface | Frame RF Interface |
| CORE_SET_CONFIG_CMD | PA_BAIL_OUT | |
| RF_DISCOVER_CMD | RF Technology and Mode | NFC_A_PASSIVE_POLL_MODE |

[1] RF_DISCOVER_MAP_CMD is optional since the mapping to Frame RF interface is done by default.

### 10.1.1.2 [PN722X NFCC-NCI] extension: TAG-CMD interface

When the Sector Select command is required, the TAG-CMD implemented in PN722X NFCC is limited to:

- MIFARE Classic operation in Reader/Writer
- T2T operation in Reader/Writer

Figure 34 represents the location of the TAG-CMD RF interface.

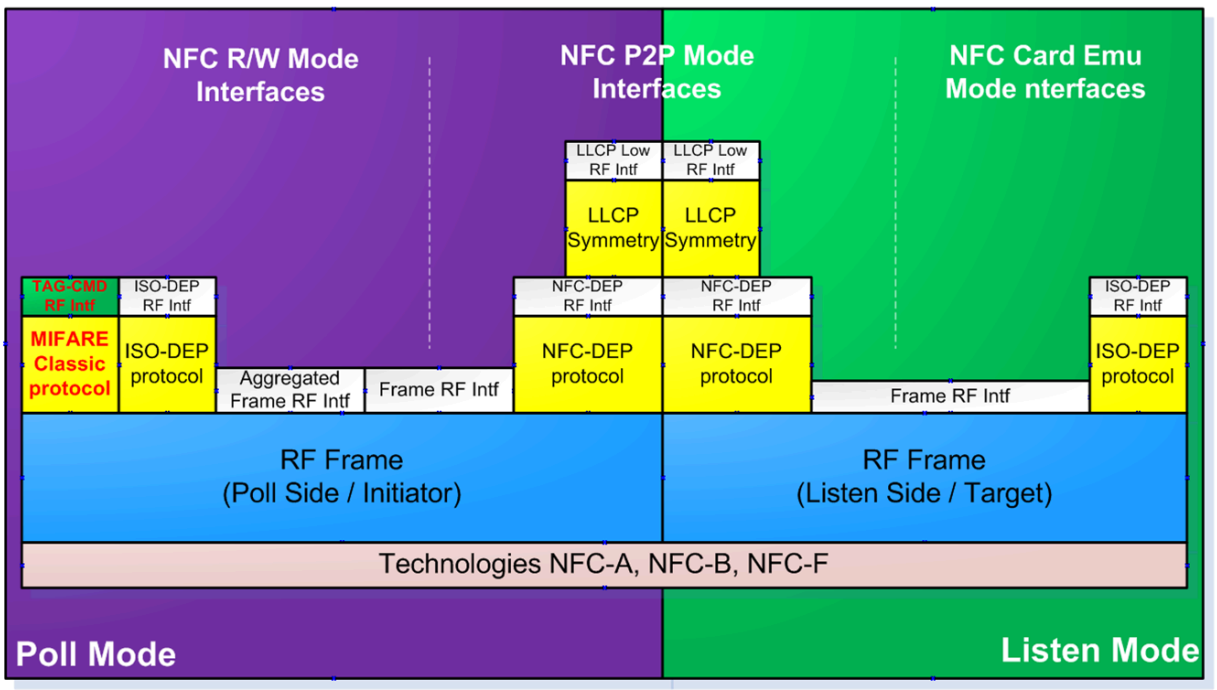



**Figure 34. TAG-CMD RF interface**

### 10.1.1.3 [PN722X NFCC-NCI] extension: Payload structure of the TAG-CMD RF interface

The TAG-CMD RF Interface uses the same data mapping as the one defined for the [NCI] Frame RF Interface (see section 8.2.1 in [NCI]). For the TAG-CMD RF Interface, the Payload is defined differently.

Two different structures are defined:

1. REQ (requests): commands from the DH to the NFCC.
2. RSP (responses): responses from the NFCC to the DH.

Figure 35 details how the Payload is modified to insert the header with the REQ ID or the RSP ID and some parameters, if required.

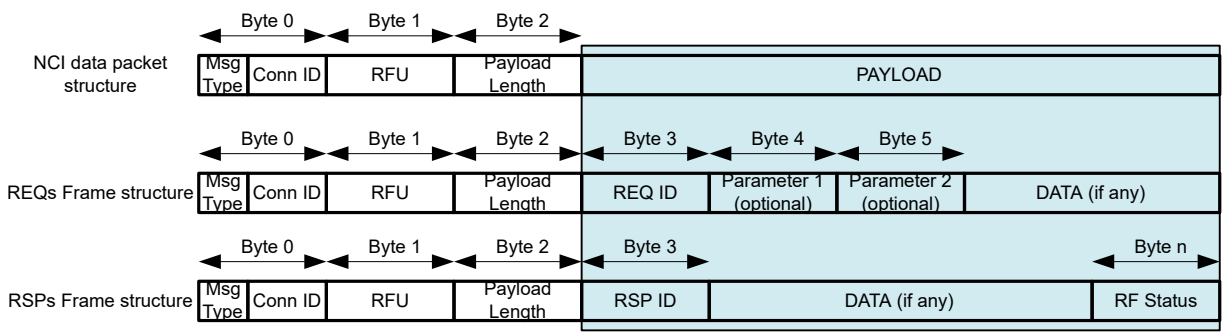



**Figure 35. Data message payload for the TAG-CMD interface**

***Note:*** *REQs and RSPs do not share exactly the same structure:*

REQs: Figure 35 shows two parameters, bur REQs may have no parameters or only one. Some REQuests might also need parameters longer than 1 Byte. Parsing The REQ ID is the way to know how many parameters follow and their length.

RSPs: there are no parameters in ReSPonses. A Byte is added at the end of the payload (after the DATA field) to inform the DH on the RF status (to report RF errors if they were some). TABLE lists the status codes.

**Table 43. TAG-CMD RF status codes**

| Value | Description |
|---|---|
| 0x00 | STATUS_OK |
| 0x03 | STATUS_FAILED |
| 0xB0 | RF_TRANSMISSION_ERROR |
| 0xB1 | RF_PROTOCOL_ERROR |
| 0xB2 | RF_TIMEOUT_ERROR |
| Others | Forbidden |

#### 10.1.1.4 [PN722X NFCC-NCI] extension: REQs and RSPs rules

A REQ command always goes from DH to RF, through the NFCC.

An RSP response always goes from the RF to the DH, through the NFCC.

The DH SHALL wait until it has received an RSP associated to a REQ before it can send a new REQ.

#### 10.1.1.5 [PN722X NFCC-NCI] extension: List of REQs and RSPs

In this section, the following acronyms are used:

**Table 44. Acronyms definition**

| Acronym | Description |
|---|---|
| MF | MIFARE family, not ISO-DEP compliant, including T2T, MIFARE Ultralight (std or C), MIFARE Classic, and MIFARE Plus for Security Level 1 and 2. |
| MFC | MIFARE Classic and MIFARE Plus for Security Level 1 and 2. |

The added REQuests/ReSPonses pairs are listed in the following table:

**Table 45. List of REQuests and RESPonses**

| REQ/RSP Name | ID | Param 1 | Param 2 | Param 3 | Data | Description |
|---|---|---|---|---|---|---|
| XCHG_DATA_REQ | 0x10 | None | None | None | Yes | MFC: DH sends Raw data to the NFCC, which encrypts them before sending them to MFC.<br>T2T: DH sends Raw data to the NFCC, which forwards them in plain to the Tag. |
| XCHG_DATA_RSP | 0x10 | N/A | N/A | N/A | Yes | MFC: DH gets Raw data once RF data from MFC are decrypted by the NFCC, if successful.<br>T2T: DH gets Raw plain data once the NFCC receives RF data from the Tag, if successful. |
| MF_SectorSel_REQ | 0x32 | Sector Address | None | None | No | T2T and MFU only: DH sends the address of the Block to select. |
| MF_SectorSel_RSP | 0x32 | N/A | N/A | N/A | No | T2T and MFU only: DH gets the “Sector Select” response status |
| MFC_Authenticate_REQ | 0x40 | Sector Address | Key Selector | Key (optional) | No | DH asks NFCC to perform MFC Authenticate command. |
| MFC_Authenticate_RSP | 0x40 | N/A | N/A | | No | DH gets the MFC Authenticate command status |

All these REQs and RSPs are detailed in the next sections.

### 10.1.1.6 [PN722X NFCC-NCI] extension: raw data exchange REQs and RSPs

**Table 46. XCHG_DATA_REQ**

| REQ_ID | REQ Name | Number of parameter(s) | Presence of data | Description |
|---|---|---|---|---|
| 0x10 | XCHG_DATA_REQ | 0 | Yes | MFC: DH sends Raw data to the NFCC, which encrypts them before sending them to MFC.<br>T2T: DH sends Raw data to the NFCC, which forwards them in plain to the Tag. |

**Table 47. XCHG_DATA_RSP**

| RSP_ID | RSP Name | Presence of Data | Description |
|---|---|---|---|
| 0x10 | XCHG_DATA_RSP | Yes | MFC: DH gets Raw data once RF data from MFC are decrypted by the NFCC, if successful.<br>T2T: DH gets Raw plain data once the NFCC receives RF data from the Tag, if successful. If the response from the MF tag in the field is an ACK or a NACK, the ACK/NACK is also sent back to the DH inside the Data field. Since ACK and NACK are 4-bit commands, they are transported on the 4 LSBs of the data byte; the 4MSBs of that Byte are forced to the logical '0' value. |

### 10.1.1.7 [PN722X NFCC-NCI] extension: T2T and MFU REQs and RSPs

All the REQs and RSPs described in this section can be used whatever the tag between:

- T2T
- MIFARE Ultralight (Standard or C)

**Table 48. MF_SectorSel_REQ**

| REQ_ID | REQ Name | Number of parameter(s) | Presence of data | Description |
|---|---|---|---|---|
| 0x32 | MF_SectorSel_REQ | 1 | No | DH Sends the address of the Sector to select. |

**Table 49. MF_SectorSel_REQ parameter**

| Parameter | | Length (Byte) | Value | Description |
|---|---|---|---|---|
| 1 | Sector Address | 1 | ? | Defines the address of the sector which has to be selected.<br>The address can be any block address in this sector. |

**Table 50. MF_SectorSel_RSP**

| RSP_ID | RSP Name | Presence of Data | Description |
|---|---|---|---|
| 0x32 | MF_SectorSel_RSP | No | DH gets sector select status |

### 10.1.1.8 [PN722X NFCC-NCI] extension: MIFARE Classic REQs and RSPs

**Table 51. MFC_Authenticate_REQ**

| REQ_ID | REQ Name | Number of parameter(s) | Presence of data | Description |
|---|---|---|---|---|
| 0x40 | MFC_Authenticate_REQ | 3 | No | DH asks NFCC to perform MFC authenticate. |

**Table 52. MFC_Authenticate_REQ parameters**

<table>
<tr><th colspan="2">Parameter</th><th>Length (Byte)</th><th>Value</th><th>Description</th></tr>
<tr><td>1</td><td>Sector Address</td><td>1</td><td></td><td>Address of the sector to authenticate</td></tr>
<tr><td>2</td><td>Key Selector</td><td>1</td><td>N/A</td><td>
<table>
<tr><th colspan="8">Bit Mask</th><th>Description</th></tr>
<tr><td>b7</td><td>b6</td><td>b5</td><td>b4</td><td>b3</td><td>b2</td><td>b1</td><td>b0</td><td></td></tr>
<tr><td>X</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td>Key A (‘0’) or Key B (‘1’)</td></tr>
<tr><td></td><td></td><td></td><td>X</td><td></td><td></td><td></td><td></td><td>0 => use pre-loaded key<br>1 => use Key embedded in the REQ (param Nbr 3)</td></tr>
<tr><td></td><td></td><td></td><td></td><td>X</td><td>X</td><td>X</td><td>X</td><td>Pre-loaded key number (0 to 15)</td></tr>
<tr><td></td><td>0</td><td>0</td><td></td><td></td><td></td><td></td><td></td><td>RFU</td></tr>
</table>
</td></tr>
<tr><td>3</td><td>Embedded Key (optional)</td><td>6</td><td>N/A</td><td>This parameter is present in the MFC_Authenticate_CMD only if bit b4 is set to logical ‘1’ in Key Selector parameter. If present, this parameter defines the value of the Key used for the authentication.</td></tr>
</table>

**Table 53. MFC_Authenticate_RSP**

| RSP_ID | RSP Name | Presence of Data | Description |
|---|---|---|---|
| 0x40 | MFC_Authenticate_RSP | No | DH gets the “authenticate” cmd status |

**Table 54. TAG-CMD RF Status code, in the special case of MFC_Authenticate_CMD**

| Value | Description | Reason |
|---|---|---|
| 0x00 | STATUS_OK | Authentication was successful |
| 0x03 | STATUS_FAILED | Authentication failed (wrong key, time-out triggered during authentication etc.) |
| 0xB0 | RF_TRANSMISSION_ERROR | Not used |
| 0xB1 | RF_PROTOCOL_ERROR | Not used |
| 0xB2 | RF_TIMEOUT_ERROR | Not used |
| Others | Forbidden | |

Once a sector is authenticated, PN722X NFCC automatically encrypts any data sent by the DH to be transferred over RF, using to the XCHG_DATA_REQ command. The key used is the one used for the sector currently authenticated.

In a symmetrical way, PN722X NFCC automatically decrypts the data received from RF before it forwards to the DH thanks to the XCHG_DATA_RSP response, again using the key of the sector currently authenticated.

Figure 36 illustrates the use of the MFC_Authenticate_REQ and XCHG_DATA_REQ in MIFARE Classic reader sequence.

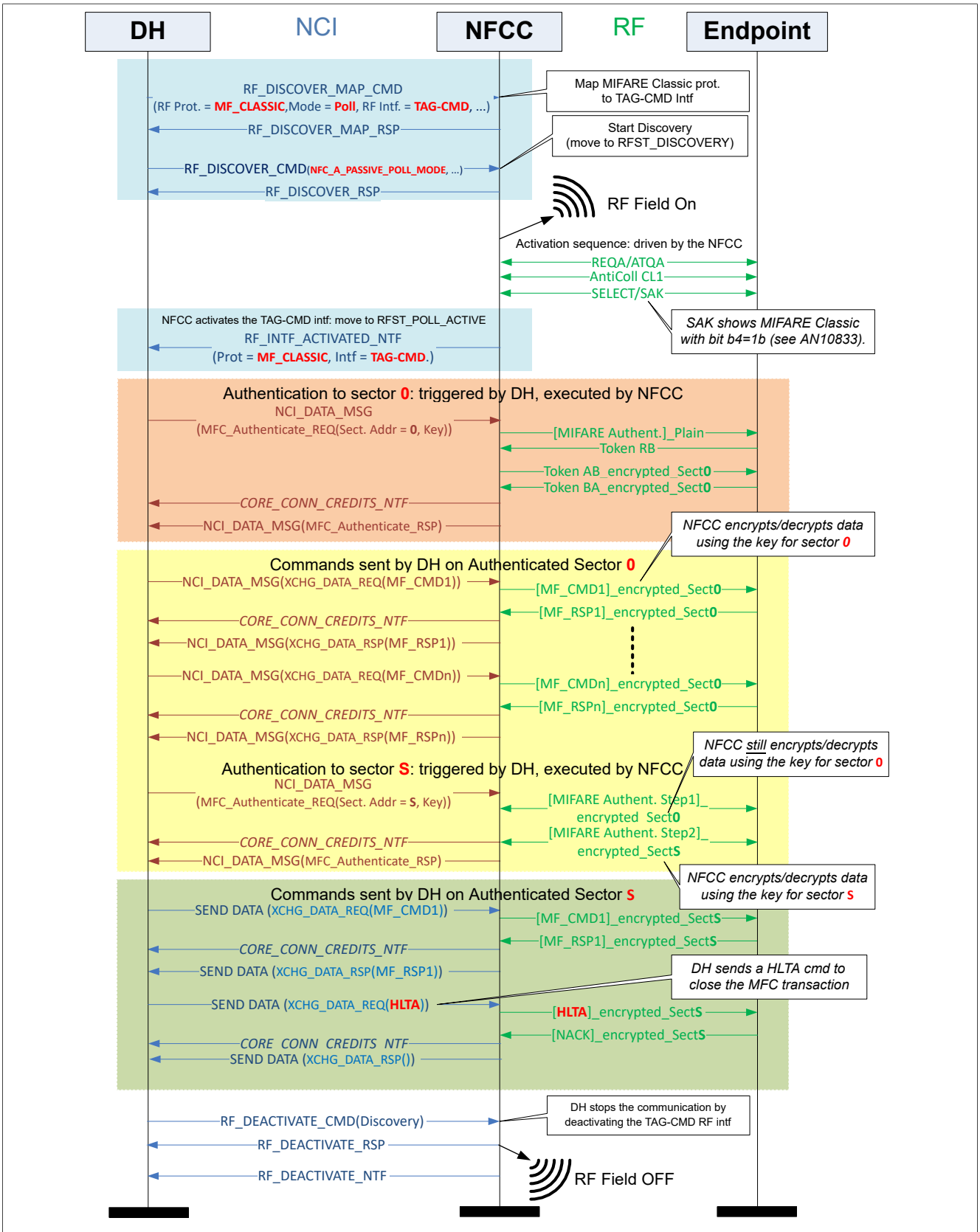



**Figure 36. MIFARE Classic Reader Sequence**

### 10.1.1.9 Access through the TAG-CMD RF interface

The TAG-CMD RF interface allows full access to all the Tags based on NFC-A technology and not supporting the ISO-DEP protocol, leaving up to the PN722X NFCC to manage the low-level TAG-CMD.

**Table 55. Tag/Cards accessible over the TAG-CMD interface**

| Tag/Card | Access through the TAG-CMD Interface |
|---|---|
| T2T | ☑ |
| MIFARE Ultralight, Ultralight C | ☑ |
| MIFARE Classic | ☑ |
| MIFARE Plus for Security levels 1 and 2 | ☑ |

☑ ... covered

☒ … not covered

Table 56 lists the commands and configuration parameters to prepare the Reader/Writer Mode for T2T, and MIFARE Classic through the TAG-CMD interface.

**Table 56. Configuration sequence for R/W of T2T and MFC through the TAG-CMD Interface**

| Command | Main Parameters | Values |
|---|---|---|
| RF_DISCOVER_MAP_CMD | RF Protocol (choose between the 2 possible protocols) | PROTOCOL_T2T |
| | | PROTOCOL_MIFARE_CLASSIC |
| | Mode | Poll |
| | RF Interface | TAG-CMD |
| CORE_SET_CONFIG_CMD | PA_BAIL_OUT [1] | — |
| RF_DISCOVER_CMD | RF Technology and Mode | NFC_A_PASSIVE_POLL_MODE |

[1] This parameter is not active in PN722X: it can be read/written, but PN722X always behaves with Bail Out in NFC-A, whatever the value written by the DH to that parameter.

### 10.1.2 T3T tag

[NCI] allows the data exchange with a tag T3T by using the Frame RF Interface. A proprietary RF interface is not needed.

#### 10.1.2.1 Access through the Frame RF interface

Table 57 lists the commands and configuration parameters to prepare the Reader/Writer Mode for T3T Tags/ Cards through the Frame RF Interface.

**Table 57. Configuration sequence for Reader/Writer of T3T through the Frame RF Interface**

| Command | Main Parameters | Values |
|---|---|---|
| RF_DISCOVER_MAP_CMD | RF Protocol | PROTOCOL_T3T |
| | Mode | Poll |
| | RF Interface | Frame |
| CORE_SET_CONFIG_CMD | PF_BIT_RATE | — |
| | PF_RC_CODE | — |
| RF_DISCOVER_CMD | RF Technology and Mode | NFC_F_PASSIVE_POLL_MODE |

### 10.1.3 T4T and ISO-DEP tags/cards

[NCI] allows the data exchange with a T4T tag or an ISO-DEP tag by using the Frame RF interface or the ISO-DEP RF interface. A proprietary RF interface is not needed.

#### 10.1.3.1 Access through the Frame RF interface

The Frame RF interface allows full access to all the tags based on NFC-A and NFC-B technology and supporting the ISO-DEP protocol, assuming that the ISO-DEP protocol is fully handled by the DH (Table 58).

**Table 58. Tag/Cards accessible over the Frame RF Interface**

| Tag/Card | Access through the Frame RF Interface |
|---|---|
| T4T | ☑ |
| MIFARE DESFire | ☑ |
| MIFARE Plus for Security levels 3 | ☑ |
| JCOP-based smart cards | ☑ |

Table 59 lists the commands and configuration parameters to prepare the Reader/Writer Mode for ISO-DEP Tags/Cards through the Frame RF Interface for technology NFC-A.

**Table 59. Configuration sequence for R/W of NFC-A / ISO-DEP through the Frame RF interface**

| Command | Main Parameters | Values |
|---|---|---|
| RF_DISCOVER_MAP_CMD[1] | RF Protocol | PROTOCOL_ISO-DEP |
| | Mode | Poll |
| | RF Interface | Frame |
| CORE_SET_CONFIG_CMD | PA_BAIL_OUT | — |
| RF_DISCOVER_CMD | RF Technology and Mode | NFC_A_PASSIVE_POLL_MODE |

[1] RF_DISCOVER_MAP_CMD is optional as the mapping to Frame RF Interface is done by default.

Table 60 lists the commands and configuration parameters to prepare the Reader/Writer Mode for ISO-DEP Tags/Cards through the Frame RF Interface for technology NFC-B.

**Table 60. Configuration sequence for R/W of NFC-B / ISO-DEP through the Frame RF interface**

| Command | Main Parameters | Values |
|---|---|---|
| RF_DISCOVER_MAP_CMD[1] | RF Protocol | PROTOCOL_ISO-DEP |
| | Mode | Poll |
| | RF Interface | Frame |
| CORE_SET_CONFIG_CMD | PB_AFI | — |
| | PB_BAIL_OUT | — |
| | PB_SENSB_REQ_PARAM | — |
| RF_DISCOVER_CMD | RF Technology and Mode | NFC_B_PASSIVE_POLL_MODE |

[1] RF_DISCOVER_MAP_CMD is optional as the mapping to Frame RF Interface is done by default.

### 10.1.3.2 Access through the ISO-DEP RF interface

The ISO-DEP RF interface allows full access to all the Tags based on NFC-A and NFC-B technology and supporting the ISO-DEP protocol, leaving up to the PN722X NFCC to manage the ISO-DEP protocol (Table 61).

**Table 61. Tag/Cards accessible over the ISO-DEP RF interface**

| Tag/Card | Access through the ISO-DEP RF Interface |
|---|---|
| T4T | ☑ |
| MIFARE DESFire | ☑ |
| MIFARE Plus for Security levels 3 | ☑ |
| JCOP-based smart cards | ☑ |

Table 62 lists the commands and configuration parameters to prepare the Reader/Writer Mode for ISO-DEP through the ISO-DEP Interface for NFC-A technology.

**Table 62. Configuration sequence for R/W of NFC-A / ISO-DEP through the ISO-DEP interface**

| Command | Main Parameters | Values |
|---|---|---|
| RF_DISCOVER_MAP_CMD | RF Protocol | PROTOCOL_ISO-DEP |
| | Mode | Poll |
| | RF Interface | ISO-DEP |
| CORE_SET_CONFIG_CMD | PA_BAIL_OUT | — |
| | PI_BIT_RATE | — |
| RF_DISCOVER_CMD | RF Technology and Mode | NFC_A_PASSIVE_POLL_MODE |

Table 63 lists the commands and configuration parameters to prepare the Reader/Writer Mode for ISO-DEP through the ISO-DEP Interface for NFC-B technology.

**Table 63. Configuration sequence for R/W of NFC-B/ISO-DEP through the ISO-DEP interface**

| Command | Main Parameters | Values |
|---|---|---|
| RF_DISCOVER_MAP_CMD | RF Protocol | PROTOCOL_ISO-DEP |
| | Mode | Poll |
| | RF Interface | ISO-DEP |
| CORE_SET_CONFIG_CMD | PB_AFI | — |
| | PB_BAIL_OUT | — |
| | PB_H_INFO | — |
| | PI_BIT_RATE | — |
| | PB_SENSB_REQ_PARAM | — |
| RF_DISCOVER_CMD | RF Technology and Mode | NFC_B_PASSIVE_POLL_MODE |

#### 10.1.3.3 [PN722X NFCC-NCI] extension: WTX notification

After data is sent to the card/tag, an additional processing time may be required before the data response is sent. The wait time extension (WTX) request is used for the additional processing time. If WTX REQ/RESP exchange phase follows an NCI system notification, WTX is sent with a period of ~1.3 s.

**Table 64. WTX_NTF**

| GID | OID | Numbers of parameter(s) | Description |
|---|---|---|---|
| 1111b | 0x17 | 0 | Notification indicating that RF communication is in phase of WTX(RTOX) REQ/RESP exchange for longer period |

# 11 Listen side: Card emulation mode

## 11.1 Card Emulation hosted by the DH

### 11.1.1 ISO-DEP based on NFC-A

For Card Emulation hosted by the DH based on technology NFC-A, the PN722X NFCC only supports the ISO-DEP protocol.

[NCI] defines all the mechanisms necessary to implement this feature. Two options are possible:

1. The DH wants to manage by itself the ISO-DEP protocol, it SHALL then map the ISO-DEP protocol on the Frame RF Interface. This is not supported by PN722X device.
2. The DH leaves the ISO-DEP protocol management to the NFCC: it SHALL then map the ISO-DEP protocol on the ISO-DEP interface.

#### 11.1.1.1 Access through the ISO-DEP RF interface

Table 65 lists the commands and configuration parameters to prepare the ISO-DEP Card Emulation for technology NFC-A, in the DH, through the ISO-DEP RF Interface.

**Table 65. Configuration sequence for ISO-DEP/NFC-A Card Emulation in the DH over ISO-DEP RF Interface**

| Command | Main Parameters | Values |
|---|---|---|
| RF_DISCOVER_MAP_CMD | RF Protocol | PROTOCOL_ISO-DEP |
| | Mode | Listen |
| | RF Interface | ISO-DEP |
| CORE_SET_CONFIG_CMD | LI_A_RATS_TC1 | — |
| | LA_HIST_BY | — |
| | LI_BIT_RATE | — |
| RF_SET_LISTEN_MODE_ ROUTING_CMD | Technology-based routing | NFC_RF_TECHNOLOGY_A routed to DH-NFCEE |
| | Protocol-based routing | PROTOCOL_ISO-DEP routed to DH-NFCEE |
| RF_DISCOVER_CMD | RF Technology and Mode | NFC_A_PASSIVE_LISTEN_MODE |

# 12 RF Discovery (Polling Loop) management

***Note:*** *the RF Discovery is the name given by the NFC Forum to what was called the Polling Loop on previous products.*

## 12.1 RF Discovery functions

This section covers RF Discovery concepts applied for PN722X NFCC. [NCI] defines the general RF state machine used for the NFC controller to discover either cards or readers. The RF state machine includes RFST_DISCOVERY state where the RF Discovery profile is applied.

In compliance with the standard, the PN722X NFCC supports the NFC Forum polling activity, limited to the current technologies defined in the standardization body (NFC-A, NFC-B, NFC-F, NFC-V).

In addition to these RF profiles, the PN722X NFCC offers a way to limit the power consumption by applying an LPCD concept. The LPCD can be seen as a precondition to enable a dedicated profile. It means that if the LPCD is triggered, the default profile is automatically started.

***Note:*** *[NCI] defines the TOTAL_DURATION of the discovery period independently of the reader phases applied. To simplify the implementation for PN722X NFCC, a timer is applied only during the Listen/pause phase. Therefore, depending on the polling phase configuration (one technology or more), the total duration can vary.*

### 12.1.1 RF Discovery state machine

Figure 37 shows the [NCI] RF state machine fully supported by PN722X.

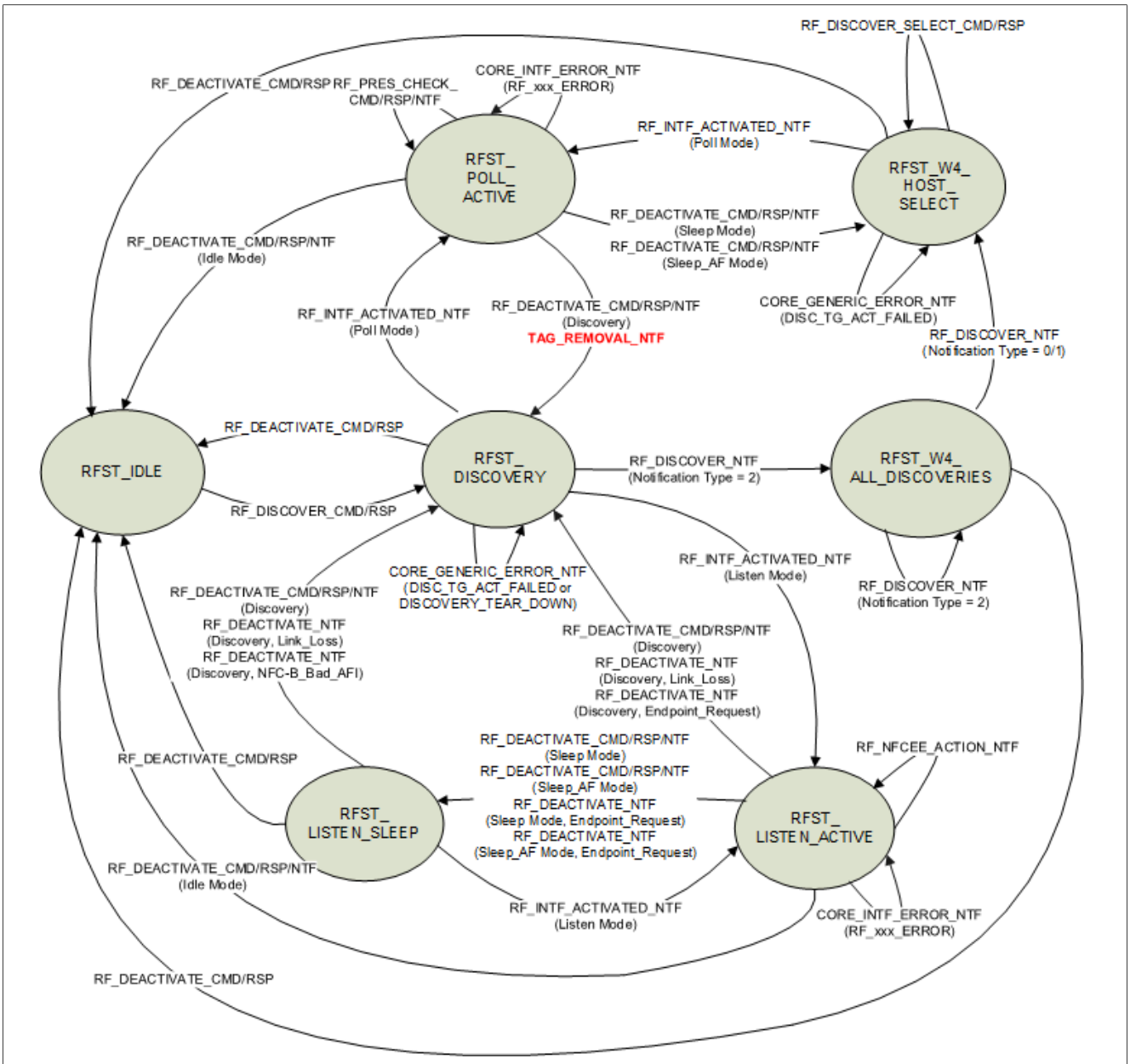



**Figure 37. NXP RF state machine**

Since the [NCI] RF State Machine is quite complex, it is presented slightly differently in Annex A of the present document: The State Machine is drawn depending on the RF interface to be used. See Section "Annex A: Details on RF state machine" for further details.

***Note:*** *As PN722X does not support Listen Mode using the Frame RF Interface, it does not accept the RF_DEACTIVATE_CMD (Sleep mode) or RF_DEACTIVATE_CMD(Discovery) in RFST_LISTEN_ACTIVE or RFST_LISTEN_SLEEP.*

## 12.2 NFC Forum profile as defined in [NCI]

The NFC Forum profile is the implementation of the RF discovery activity as defined in the NFC Forum (see [ACTIVITY] specification).

[NCI] implementation of PN722X only covers technologies NFC-A, NFC-B, NFC-F, and NFC-V. So, the basic NFC Forum profile polls for these technologies only. Furthermore, for NFC-F, only one bit rate is used during the polling phase. This is configured thanks to the “Poll F parameter” PF_BIT_RATE as defined in [NCI], section 6.1.3. So, the DH configures if NFC-F is polled at 212 kB/s or at 424 kB/s, before it activates the discovery by sending the RF_DISCOVER_CMD command.

Figure 38 represents the profile defined by the NFC Forum. The assumption is that the DH has enabled the four technologies currently supported by the product (NFC-A, NFC-B, NFC-F, and NFC-V) in Poll mode and Listen mode.

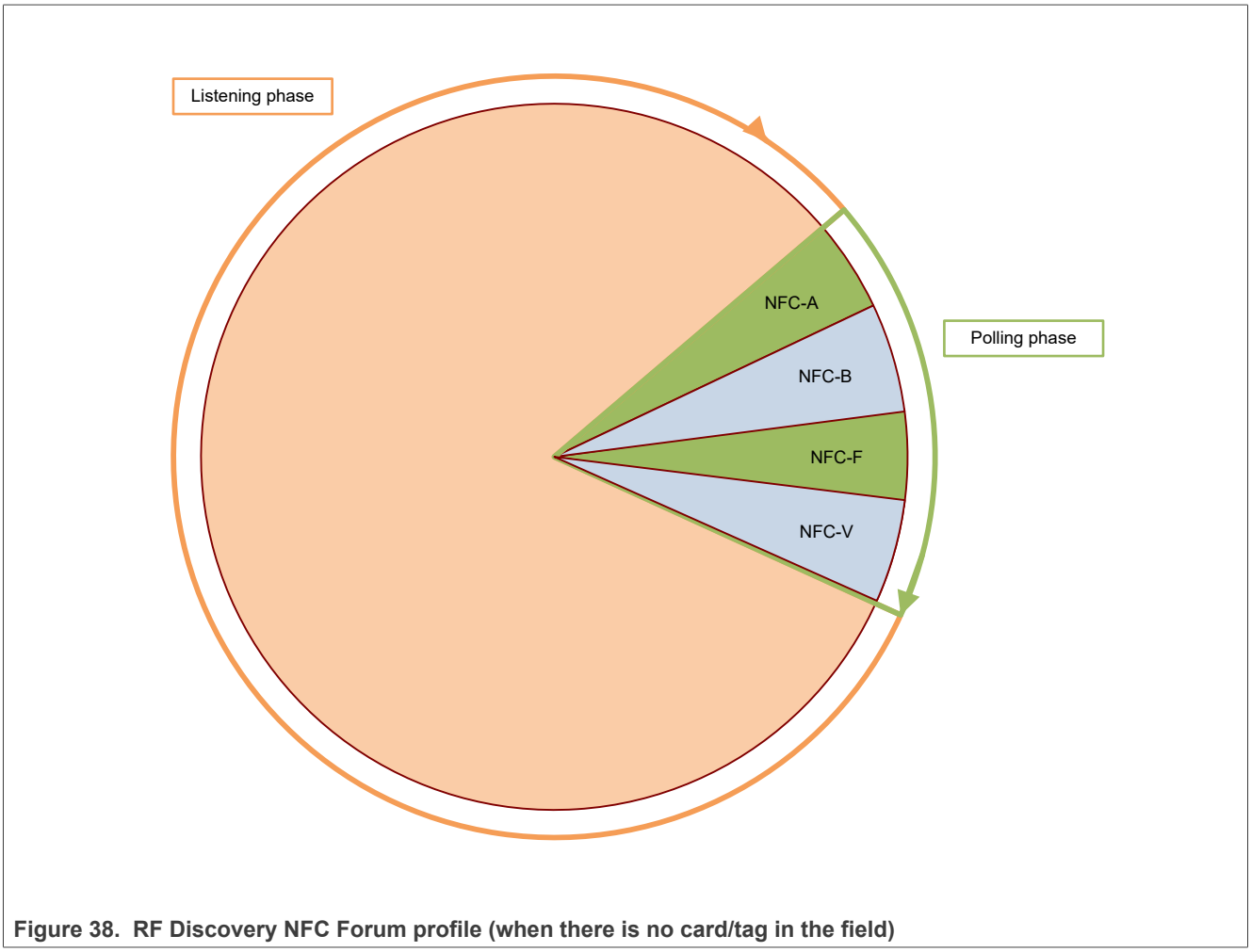



**Figure 38. RF Discovery NFC Forum profile (when there is no card/tag in the field)**

## 12.3 [PN722X NFCC-NCI] extension: Low-power card detector (LPCD) Mode

### 12.3.1 Description

The Low-Power Card Detector is an NXP proprietary extension, which is applied in case the DH wants to reduce the power consumption.

The concept is to avoid using the Technology Detection Activity as defined in [ACTIVITY], which implies to generate an RF Field for several tens of milliseconds and to send technology-specific request commands to see if there is a Card/Tag in the field to respond. The more technologies the PN722X NFCC is configured to detect, the longer the RF Field is generated and the higher the current consumption.

The LPCD is based on another concept, which only relies on the antenna characteristics, not on valid responses from a Card/Tag. The antenna impedance is influenced by the Card/tag which may enter into its proximity, due to the magnetic coupling between the two antennas. The LPCD monitors the antenna impedance to see if there is a significant variation. The variation is interpreted as being caused by a Card/Tag being in proximity.

To achieve that, the LPCD periodically generates very short pulses of RF Field, without any modulation, and measures some antenna characteristics during this pulse. The time between these RF pulses is defined by the Total_duration parameter, as specified in the NCI Configuration table.

When a Card/Tag enters the field, there is an antenna impedance variation. If this variation is higher than a predefined threshold, the default Technology Detection profile (NFC Forum) is automatically started. Next, the PN722X NFCC sends technology-specific request commands, expecting a response since the LPCD detected a change on the antenna impedance.

***Note:*** *The LPCD may also be triggered by a metal object, which can influence the Antenna impedance in a similar way as a Card/Tag. The PN722X NFCC anyhow detects that this object is not a contactless device since it immediately starts sending contactless commands to check if a Card/Tag can respond.*

The Low-Power Card Detector is configured and enabled/disabled with a specific configuration parameter in EEPROM and can be configured by DH. Refer to EEPROM_CFG CMD in Section 13.1. The command is used to update LPCD configuration defined in the data sheet.

Figure 39 illustrates the RF Discovery when the LPCD is enabled.

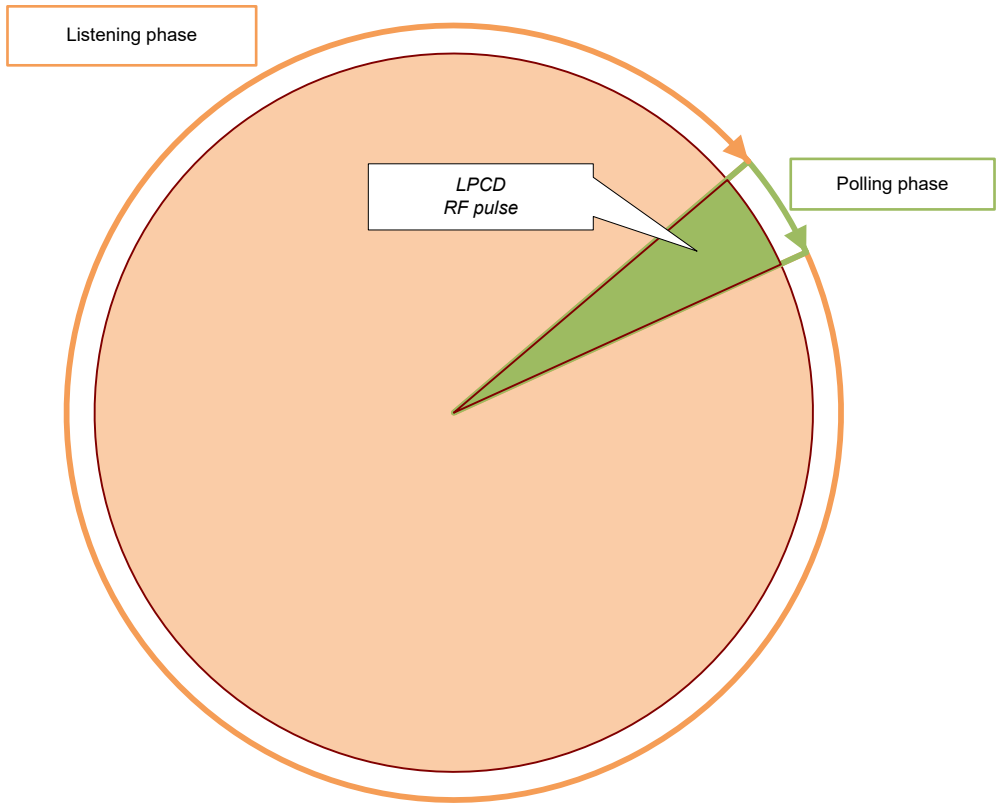



**Figure 39. Low-Power Card Detector mode (when there is no card/tag in the field)**

Figure 40 compares the RF Discovery with the LPCD disabled to the RF Discovery with the LPCD enabled. The figure highlights the impact on the average current consumption. The assumption is that TOTAL_DURATION ~ 350 ms.

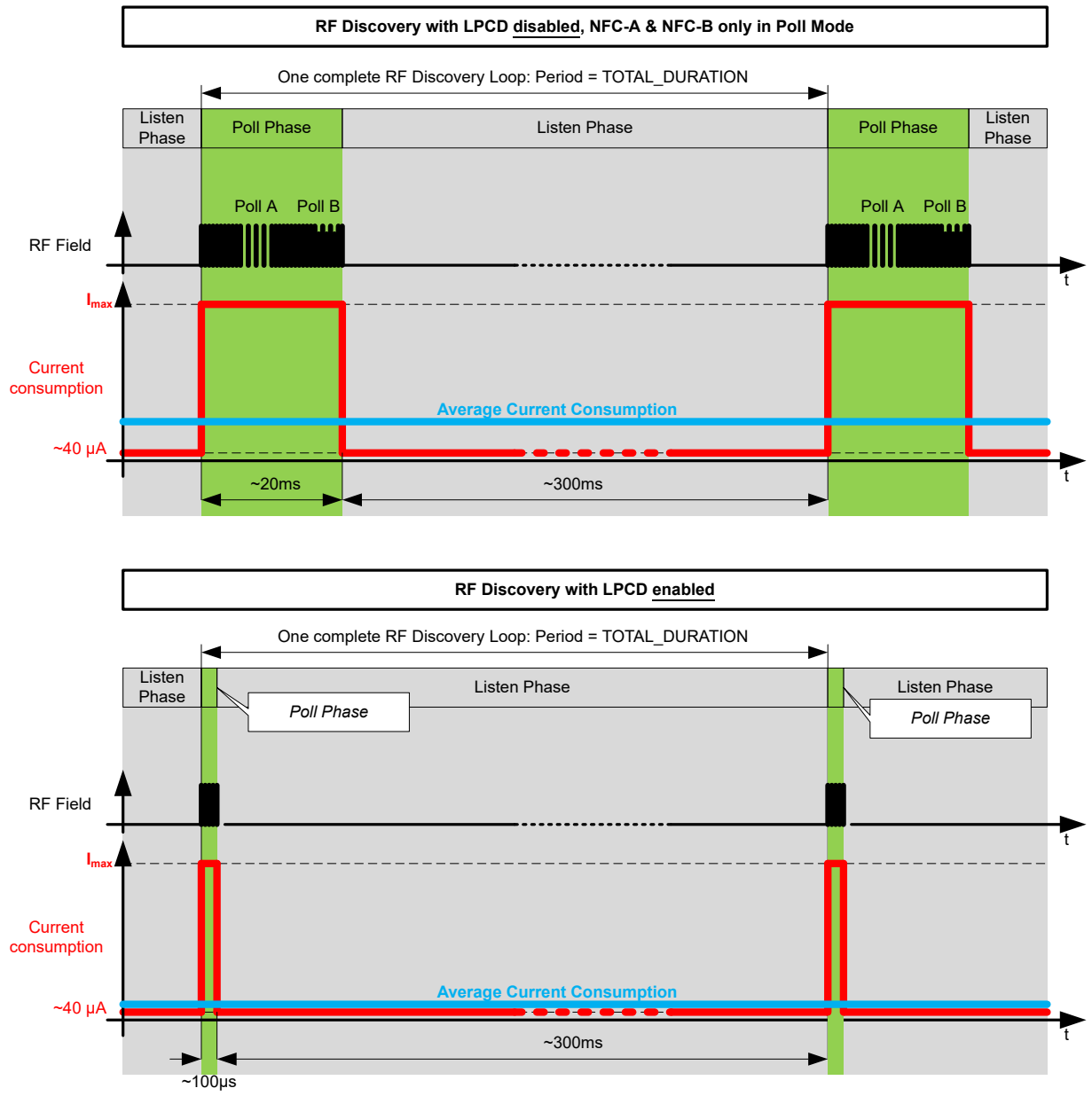



**Figure 40. Comparison of the RF Discovery with the LPCD disabled or enabled**

### 12.3.2 Technology Detection Activity when the LPCD has detected an "object"

As described in Section 12.3.1, once the PN722X detects a change in the antenna impedance, it performs a Technology Detection as defined in [ACTIVITY]. Technology Detection activates the “object” by sending Request Commands from the different technologies configured for the RF Discovery.

Figure 41 describes the Technology Detection activity when the LPCD is enabled.

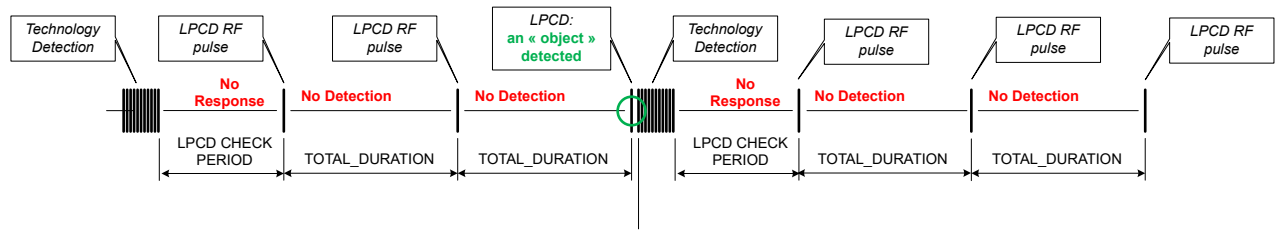



**Figure 41. Illustration of the Low Power Card detector and the subsequent Technology Detection activity**

## 12.4 [PN722X NFCC-NCI] extension: Power optimization

PN722X NFCC offers a standby state, which can be activated together with the RF Discovery, such that the overall power consumption is significantly reduced.

One dedicated proprietary function is added to enable/disable these different power modes: CORE_SET_POWER_MODE. This command can be used to overwrite the DEFAULT_POWER register and will have effect until the next hardware reboot.

### 12.4.1 CORE_SET_POWER_MODE command/response

***Note:*** *The Standby State is enabled by default in PN722X. Given the very strong impact on the power consumption, disabling the Standby State should be restricted to debug sessions.*

*This command can be called in RFST_IDLE and RFST_DISCOVERY NCI state*

**Table 66. CORE_SET_POWER_MODE_CMD**

| GID | OID | Numbers of parameter(s) | Description |
|---|---|---|---|
| 1111b | 0x00 | 1 | Command to request the PN722X NFCC to enable/disable the Standby State |

**Table 67. CORE_SET_POWER_MODE_CMD parameter**

| Payload Field(s) | Length | Value/Description | |
|---|---|---|---|
| Mode | 1 Octet | 0x00 | Standby State disabled |
| | | 0x01 | Standby State enabled |
| | | 0x02-0xFF | RFU |

***Note: The PN722X is prevented from going into standby when:***

- *It detects any activity on HIF (either SPI or HIF1-$I^2C$) interface.*
- *It detects any activity on GPIO3 (generally used along with HIF2-$I^2C$ interface) used to wake-up due to activity on HIF2-$I^2C$ interface.*
- *Once NFCEE_DISCOVERY_CMD is received and CT card is present in EMVCo CT slot.*
- *Once NFCEE_MODE_SET_CMD (Enable) on any CT slot is received from DH, PN7220 shall not enter Standby till NFCEE_MODE_SET_CMD(Disable) is received from DH.*
- *It detects an external RF Field and it is configured for host card emulation.*
- *Change in the state on MODE_SWITCH GPI exits PN722X from standby and PN722X enters either NFC forum or EMVCo mode based on the MODE_SWITCH GPI.*

**Table 68. CORE_SET_POWER_MODE_RSP**

| GID | OID | Numbers of parameter(s) | Description |
|---|---|---|---|
| 1111b | 0x00 | 1 | Response to inform the DH of the status of the command |

**Table 69. CORE_SET_POWER_MODE_RSP parameter**

| Payload Field(s) | Length | Value/Description | |
|---|---|---|---|
| Status | 1 Octet | 0x00 | STATUS_OK |
| | | 0x06 | STATUS_SEMANTIC_ERROR |
| | | 0x09 | STATUS_INVALID_PARAM |
| | | Others | RFU |

# 13 Configurations

***Note:*** *When the DH needs to update the value of the parameters described hereafter, it shall send a CORE_RESET_CMD/CORE_INIT_CMD sequence after the CORE_SET_CONFIG_CMD, to ensure that the new value is used for the parameters.*

*If several parameters are updated thanks to multiple CORE_SET_CONFIG_CMDs, a single CORE_RESET_CMD/ CORE_INIT_CMD sequence is enough after the last CORE_SET_CONFIG_CMD.*

***Note:*** *Any CORE_SET_CONFIG_CMD to one of the parameters that is stored in EEPROM will trigger a Flash write cycle. Since the PN722X NFCC Flash has a limited number of Erase/Write cycles, it is highly recommended to only use the CORE_SET_CONFIG_CMD during the NCI initialization sequence. As most of the EEPROM values are copied to RAM before referring in the Code. A POR is expected to be performed by DH.*

## 13.1 [PN722X NFCC-NCI] extension: System configurations

PN722X NFCC offers several parameters used to configure the system aspects.

**Table 70. Core configuration parameters**

<table>
<tr><th>Name and Rights</th><th>Description</th><th>Ext. Tag</th><th>Len</th><th>Default Value</th></tr>
<tr><td>NFCC_DIE_ID<br>Read only in FLASH DATA<br>Not overwritten by FW download</td><td>16 bytes unique NFCC die identifier</td><td>0xA001</td><td>16</td><td>N/A<br>Unique per chip</td></tr>
<tr><td>DEFAULT_POWER<br>RW in FLASH DATA</td><td>Default power state after hardware reset (can be overridden by SET_POWER_MODE API)<br>0x00: no standby state<br>0x01: standby state</td><td>0xA015</td><td>1</td><td>0x01</td></tr>
<tr><td>EEPROM_CFG<br>RW in FLASH DATA</td><td>EEPROM Configuration of PN722X device configuration for Power Management functionality, Clock Configuration, CLIF functionality and CT configuration etc.<br>This configuration will be used to Read and Write the EEPROM.<br>Offset is used to address the EEPROM Configuration as a multiple of 8<br><table><tr><td>Offset</td><td>EEPROM Address (hex)</td></tr><tr><td>0</td><td>0x00</td></tr><tr><td>1</td><td>0x08</td></tr><tr><td>2</td><td>0x10</td></tr><tr><td>3</td><td>0x18</td></tr><tr><td>n</td><td>0x(n*8)</td></tr></table>Annex E: Define the EEPROM mapping and functionality.</td><td>0xA2XX<br>XX – Offset</td><td>1-8</td><td>PN722X_DS</td></tr>
</table>

**Table 70. Core configuration parameters***...continued*

| Name and Rights | Description | Ext. Tag | Len | Default Value |
|---|---|---|---|---|
| TO_BEFORE_STDBY_CFG<br>***RW in Mirrored Flash DATA (RAM-AO)*** | Timeout used to wait after last DH communication before going into standby (from 0 to 65.536 s in steps of 10 milli sec). Minimum value needs to be more than 100 milli sec.<br>Applies only when the discovery is stopped and standby state is activated.<br>NOTE: when this timeout is reached, in case DH has to read pending RSP/NTF, NFCC waits until the read operation by DH clears the pending RSP/NTF. | 0xA009 | 2 | 0x03E8 =1000 (1 s) |
| DYNAMIC_POWER_REDUCTION<br>***RW in RAM*** | Configuration to reduce the Targe current (configured in EEPROM Area for NFC Forum or EMVCo using<br>EEPROM_CFG defined in this table) dynamically in RFST_IDLE.<br>The configured dynamic power level will be reset in case of CORE_RESET_CMD, Toggle on Mode_Switch_GPI or POR. | 0xA1A4 | 1 | NA |
| MW_AREA<br>***RW in FLASH DATA*** | Specific Area reserved for MW purpose in order to save specific data in flash | 0xA00F | 32 | All 0x00 |

## 13.2 [PN722X NFCC-NCI] extension: RF Discovery configuration

### 13.2.1 Poll Mode

Several configuration parameters are required for the Poll Mode in RF discovery:

**Table 71. Poll Mode configuration**

<table>
<tr><th>Name and Rights</th><th>Description</th><th>Ext. Tag</th><th>Len.</th><th>Default Value</th></tr>
<tr><td>EMVCo_PCD_SETTINGS<br>RW in FLASH DATA</td><td>
<table>
<tr><th>Payload Field</th><th>Length (bytes)</th><th>Default Value</th></tr>
<tr><td>RF OFF delay applied during EMVCo removal procedure (between 5.1 ms and 10 ms)</td><td>2</td><td>0x13EC (5100us)</td></tr>
<tr><td>RFU</td><td>1</td><td>0x06</td></tr>
</table>
</td><td>0xA064</td><td>3</td><td>0x13EC<br>0x06</td></tr>
<tr><td>POLL_PROFILE_SEL_ CFG<br>RW in Mirrored FLASH DATA (RAM-AO)</td><td>Discovery profile selection in Poll Mode as follows:
<table>
<tr><th>Bit</th><th>Description</th></tr>
<tr><td>Bit 0</td><td>RFU</td></tr>
<tr><td>Bit 1</td><td>Required for specific analog compliance.<br>0 => Removal on Idle deactivation<br>1 => Power off on Idle deactivation</td></tr>
<tr><td>Bit 2</td><td>Required for specific analog compliancy (TAB114).<br>0 => Removal on discover deactivation<br>1 => Power off on discover deactivation</td></tr>
<tr><td>Bit 3 – Bit 4</td><td>RFU</td></tr>
<tr><td>Bit 5</td><td>Required for specific analog compliance<br>0 => Polling Type B first<br>1 => Polling Type A first</td></tr>
<tr><td>Bit 6</td><td>Required for specific analog compliance<br>0 => EMVCo PCD polling (digital)<br>1 => EMVCo PCD polling (analog) (TRANSAC)</td></tr>
<tr><td>Bit 7</td><td>RFU</td></tr>
</table>
NOTE: When Bit 6 is set to 1, NFCC reports an CORE_ GENERIC_ERROR_NTF(DISCOVERY_TARGET_ ACTIVATION_FAILED) after successfully sending the Commands List defined by the DTE Specification.<br>NFCC will continue to send CORE_GENERIC_ERROR_ NTF(DISCOVERY_TARGET_ACTIVATION_FAILED) and the Commands List until the DH issues RF_DEACTIVATE_CMD requesting NFCC to stop the Command sequence sent on RF Interface..
</td><td>0xA044</td><td>1</td><td>0x00</td></tr>
</table>

**Table 71. Poll Mode configuration***...continued*

| Name and Rights | Description | Ext. Tag | Len. | Default Value |
|---|---|---|---|---|
| MFC_KEY-0_CFG<br>***WO in FLASH DATA***[1] | Key 0, used in MIFARE Classic Authentication command. | 0xA04D | 6 | 0xA0A<br>1A2A<br>3 A4A5 |
| MFC_KEY-1_CFG<br>***WO[1] in FLASH DATA*** | Key 1, used in MIFARE Classic Authentication command. | 0xA04E | 6 | 0xD3F7<br>D3F7<br>D3F7 |
| MFC_KEY-2_CFG<br>***WO[1] in FLASH DATA*** | Key 2, used in MIFARE Classic Authentication command. | 0xA04F | 6 | 0xFFFF<br>FFFF<br>FFFF |
| MFC_KEY-3_CFG<br>***WO[1] in FLASH DATA*** | Key 3, used in MIFARE Classic Authentication command. | 0xA050 | 6 | 0xFFFF<br>FFFF<br>FFFF |
| MFC_KEY-4_CFG<br>***WO[1] in FLASH DATA*** | Key 4, used in MIFARE Classic Authentication command. | 0xA051 | 6 | 0xFFFF<br>FFFF<br>FFFF |
| MFC_KEY-5_CFG<br>***WO[1] in FLASH DATA*** | Key 5, used in MIFARE Classic Authentication command. | 0xA052 | 6 | 0xFFFF<br>FFFF<br>FFFF |
| MFC_KEY-6_CFG<br>***WO[1] in FLASH DATA*** | Key 6, used in MIFARE Classic Authentication command. | 0xA053 | 6 | 0xFFFF<br>FFFF<br>FFFF |
| MFC_KEY-7_CFG<br>***WO[1] in FLASH DATA*** | Key 7, used in MIFARE Classic Authentication command. | 0xA054 | 6 | 0xFFFF<br>FFFF<br>FFFF |
| MFC_KEY-8_CFG<br>***WO[1] in FLASH DATA*** | Key 8, used in MIFARE Classic Authentication command. | 0xA055 | 6 | 0xFFFF<br>FFFF<br>FFFF |
| MFC_KEY-9_CFG<br>***WO[1] in FLASH DATA*** | Key 9, used in MIFARE Classic Authentication command. | 0xA056 | 6 | 0xFFFF<br>FFFF<br>FFFF |
| MFC_KEY-10_CFG<br>***WO[1] in FLASH DATA*** | Key 10, used in MIFARE Classic Authentication command. | 0xA057 | 6 | 0xFFFF<br>FFFF<br>FFFF |
| MFC_KEY-11_CFG<br>***WO[1] in FLASH DATA*** | Key 11, used in MIFARE Classic Authentication command. | 0xA058 | 6 | 0xFFFF<br>FFFF<br>FFFF |
| MFC_KEY-12_CFG<br>***WO[1] in FLASH DATA*** | Key 12, used in MIFARE Classic Authentication command. | 0xA059 | 6 | 0xFFFF<br>FFFF<br>FFFF |
| MFC_KEY-13_CFG<br>***WO[1] in FLASH DATA*** | Key 13, used in MIFARE Classic Authentication command. | 0xA05A | 6 | 0xFFFF<br>FFFF<br>FFFF |
| MFC_KEY-14_CFG<br>***WO[1] in FLASH DATA*** | Key 14, used in MIFARE Classic Authentication command. | 0xA05B | 6 | 0xFFFF<br>FFFF<br>FFFF |

**Table 71. Poll Mode configuration***...continued*

| Name and Rights | Description | Ext. Tag | Len. | Default Value |
|---|---|---|---|---|
| MFC_KEY-15_CFG<br>***WO[1] in FLASH DATA*** | Key 15, used in MIFARE Classic Authentication command. | 0xA05C | 6 | 0xFFFF FFFF FFFF |

[1] Use CORE_SET_CONFIG_CMD for write-only (WO) parameters. PN722X NFCC always returns CORE_GET_CONFIG_RSP (STATUS_INVALID_PARAM) to any attempt to read the value of the WO parameter. The MFC_KEY configurations are overwritten after each FW download procedure.

[1] WO (write-only) parameters can only be written, using CORE_SET_CONFIG_CMD. PN722X NFCC always returns CORE_GET_CONFIG_RSP (STATUS_INVALID_PARAM) to any attempt to read the value of the WO parameter. The MFC_KEY configurations will be overwritten after each FW download procedure.

## 13.3 [PN722X NFCC-NCI] extension: Contactless interface configurations

PN722X NFCC offers multiple configuration options for the contactless interface, optimizing the match between the antenna characteristics and the transmitter and receiver in PN722X NFCC.

A generic TLV mechanism has been defined to write the contactless Interface settings. The mechanism relies on the [NCI] CORE_SET_CONFIG_CMD / CORE_GET_CONFIG_CMD and is described in Table 72.

**Table 72. Mechanism to configure the RF protocol configuration**

<table>
<tr><th>Name and Rights</th><th>Description</th><th>Ext. Tag</th><th>Default value</th></tr>
<tr><td>RF_PROTOCOL_CFG<br>RW in FLASH DATA</td><td>
Used to update and retrieve the RF configuration within E2PROM.<br>
• This configuration allows updating at register granularity value, i.e. not the complete set needs to be updated (though, it is possible to do it).<br>
• This configuration allows to retrieve the RF Configuration of one Tx/Rx Protocol Index at a time.<br>
Parameter to configure the defined RF Protocol Configurations as described in Annex C.<br>
Conditions<br>
The size of the field array ‘Configuration’ must be in the range from 1 – 15, inclusive. The field array ‘Configuration’ must contain a set of ‘RF Configuration’, ‘Register Address’ and ‘Value’. The field ‘RF configuration’ must be in the range from 0x0 – 0x2B for TX Configuration and 0x80 – 0xAB for the RX configuration, inclusive. The address within field ‘Register Address’ must exist within the respective RF configuration. Field ‘Value’ should contain a value which has to be written into the given register and must be 4 bytes long (little-endian format).<br>
GetConfig: to be provided with 1 byte Index and response as multiple of 5 Bytes
<table>
<tr><th>Configuration Index</th><th>Value/Description</th></tr>
<tr><td>1 Byte</td><td>RF Configuration for which the EEPROM Values must be retrieved</td></tr>
</table>
<table>
<tr><th>Configuration Len [1….n]</th><th colspan="3">Value/Description</th></tr>
<tr><td rowspan="2">5 Byte * n</td><td>Register Address</td><td>1 Byte</td><td>Register Address within the given RF technology.</td></tr>
<tr><td>Value</td><td>4 Bytes</td><td>Value which must be written into the register. (Little-endian)</td></tr>
</table>
SetConfig: multiple of 6
<table>
<tr><th>Configuration Len [1….n]</th><th colspan="3">Value/Description</th></tr>
<tr><td rowspan="3">6 Byte * n</td><td>RF Configuration</td><td>1 Byte</td><td>RF Configuration for which the register must be changed.</td></tr>
<tr><td>Register Address</td><td>1 Byte</td><td>Register Address within the given RF technology.</td></tr>
<tr><td>Value</td><td>4 Bytes</td><td>Value which must be written into the register. (Little-endian)</td></tr>
</table>
</td><td>0xA00D</td><td>—</td></tr>
</table>

# 14 Test modes

## 14.1 Test session

Whatever the test command used by the DH, it is necessary to implement a "test session", which isolates the test mode from a regular "NCI session" of PN722X. This test session is defined thanks to the following sequence:

- Reset/Init the PN722X using CORE_RESET_CMD/CORE_INIT_CMD
- Retrieve Standby state
- Disable Standby using CORE_SET_POWER_MODE Command
- Launch the selected test function
- Get the response (optionally the notifications) transporting the status of the test executed
- Restore Standby state
- Reset/Init the PN722X using CORE_RESET_CMD/CORE_INIT_CMD

## 14.2 TEST_PRBS_CMD/RSP

This command is used to start PRBS infinite stream generation using a specific HW block. A 511-bit pseudo-random test sequence (PRBS9) or a 32 767-bit pseudo-random test sequence (PRBS15) can be selected. RF Field needs to be turned On using SWITCH_RF_FIELD_CMD before this CMD is issued by DH.

To exit PRBS state, either POR or CORE_RESET_CMD/CORE_INIT_CMD needs to be issued by DH.

**Table 73. TEST_PRBS_CMD**

| GID | OID | Numbers of parameter(s) | Description |
|---|---|---|---|
| 1111b | 0x30 | 6 | Command to start PRBS generation |

**Table 74. TEST_PRBS_CMD parameters**

| Payload Field(s) | Length | Value/Description |
|---|---|---|
| PRBS Mode | 1 Octet | 0x01: Hardware generated PRBS<br>Others: RFU |
| PRBS type | 1 Octet | 0x00: PRBS9 when hardware generated PRBS selected<br>0x01: PRBS15 when hardware generated PRBS selected<br>Others: RFU |
| Technology to stream | 1 Octet | 0x00: Type A<br>0x01: Type B<br>0x02: Type F<br>0x03: Type V<br>Others: RFU |

**Table 74. TEST_PRBS_CMD parameters***...continued*

| Payload Field(s) | Length | Value/Description |
|---|---|---|
| Bit rate | 1 Octet | 0x00: 106 kbit/s (Type A,B)<br>0x01: 212 kbit/s (Type A,B and F)<br>0x02: 424 kbit/s (Type A,B and F)<br>0x03: 848 kbit/s (Type A,B)<br>0x04: 26 kbit/s (Type V)<br>Others: RFU |
| PRBS series length | 2 Octets | 0x0000: Fixed<br>Others: RFU |

**Table 75. TEST_PRBS_RSP**

| GID | OID | Numbers of parameter(s) | Description |
|---|---|---|---|
| 1111b | 0x30 | 1 | PN722X reports if the TEST_PRBS_CMD is successful. |

In case the NCI package is malformed (wrong length for example), STATUS_SYNTAX_ERROR is returned. If parameters are out of bounds (forbidden values as stated above), STATUS_INVALID_PARAM will be sent. In case the PRBS tests are started in any state other than RFST_IDLE, STATUS_SEMANTIC_ERROR will be returned.

**Table 76. TEST_PRBS_RSP parameters**

| Payload Field(s) | Length | Value/Description |
|---|---|---|
| STATUS | 1 Octet | 0x00: STATUS_OK<br>0x05: STATUS_SYNTAX_ERROR<br>0x06: STATUS_SEMANTIC_ERROR<br>0x09: STATUS_INVALID_PARAM<br>Others: RFU |

***Note:*** *The only way to stop the on-going PRBS pattern generation is to send a NCI_CORE_RESET_CMD.*

## 14.3 REG_TEST_API_CMD/RSP

This command is used to read/write internal hardware register values that can be used for different purpose such as antenna matching, DPC calibration etc.

**Table 77. REG_TEST_API_CMD for Read Operation**

| GID | OID | Numbers of parameter(s) | Description |
|---|---|---|---|
| 1111b | 0x32 | 2 | Command to get specific hardware parameters |

**Table 78. REG_TEST_API_CMD parameter for Read Operation**

| Payload Field(s) | Length | Value/Description | |
|---|---|---|---|
| Operation | 1 Octet | | |
| | | 0x00 | Read PN722X registers. |
| | | 0x01 | Write PN722X registers. |
| | | Others | RFU |
| Hardware Register | 1 Octet | | |
| | | 0x00 – 0x6D | PN722X registers. |
| | | 0x80 – 0x8F | |
| | | Others | RFU |

**Table 79. REG_TEST_API _RSP for Read Operation**

| GID | OID | Numbers of parameter(s) | Description |
|---|---|---|---|
| 1111b | 0x32 | 5 | PN722X reports if the RF_TEST_API_CMD is successful and the corresponding hardware value if successful. |

If parameters are out of bounds (forbidden values as stated above), STATUS_INVALID_PARAM will be sent. In case the command is sent in any state other than RFST_IDLE, STATUS_SEMANTIC_ERROR will be returned.

**Table 80. REG_TEST_API_RSP parameters for Read Operation**

| Payload Field(s) | Length | Value/Description | |
|---|---|---|---|
| STATUS | 1 Octet | | |
| | | 0x00 | STATUS_OK |
| | | 0x05 | STATUS_SYNTAX_ERROR |
| | | 0x06 | STATUS_SEMANTIC_ERROR |
| | | 0x09 | STATUS_INVALID_PARAM |
| | | Others | RFU |
| Hardware Register Value | 4 Octet | Please refer to details on the applicable register values in PN722X_DS | |

Below command/response format is used to write internal hardware register values.

**Table 81. REG_TEST_API_CMD for Write Operation**

| GID | OID | Numbers of parameter(s) | Description |
|---|---|---|---|
| 1111b | 0x32 | 6 | Command to set specific hardware parameters |

**Table 82. REG_TEST_API_CMD parameter for Write Operation**

| Payload Field(s) | Length | Value/Description | |
|---|---|---|---|
| Operation | 1 Octet | 0x00 | Read PN722X registers. |
| | | 0x01 | Write PN722X registers. |
| | | Others | RFU |
| Hardware Register | 1 Octet | 0x00 – 0x6D | PN722X registers. |
| | | 0x80 – 0x8F | |
| | | Others | RFU |
| Hardware Register Value | 4 Octet | Please refer to details on the applicable register values in PN722X_DS | |

**Table 83. REG_TEST_API _RSP for Write Operation**

| GID | OID | Numbers of parameter(s) | Description |
|---|---|---|---|
| 1111b | 0x32 | 1 | PN722X reports if the RF_TEST_API_CMD is successful and the corresponding hardware value if successful. |

If parameters are out of bounds (forbidden values as stated above), STATUS_INVALID_PARAM will be sent. In case the command is sent in any state other than RFST_IDLE, STATUS_SEMANTIC_ERROR will be returned.

**Table 84. REG_TEST_API_RSP parameters for Write Operation**

| Payload Field(s) | Length | Value/Description | |
|---|---|---|---|
| STATUS | 1 Octet | 0x00 | STATUS_OK |
| | | 0x05 | STATUS_SYNTAX_ERROR |
| | | 0x06 | STATUS_SEMANTIC_ERROR |
| | | 0x09 | STATUS_INVALID_PARAM |
| | | Others | RFU |

## 14.4 CTS Command/Response/Notification

### 14.4.1 CTS_CONFIG_CMD/RSP

This command is used by DH to configure all the required CTS registers such as triggers, test bus registers, sampling, configuration, and more. Upon receiving this command the NFCC validates the configuration, calculates the buffer and if valid; NFCC programs the settings in the hardware.

**Table 85. CTS_CONFIG_CMD**

| GID | OID | Numbers of parameter(s) | Description |
|---|---|---|---|
| 1111b | 0x22 | 0x28 | Command to configure the CTS Configuration |

**Table 86. CTS_CONFIG_CMD parameters**

| Payload Field(s) | Length | Value/Description |
|---|---|---|
| COMMAND | 1 Octet | 0x80: CTS_CONFIG_CMD |
| PRE_TRIGGER_SHIFT | 1 Octet | Defines the length of the after-trigger acquisition sequence in 256-bytes units.<br>0 means no shift; n means n*256 bytes block shift.<br>Note: Valid only if TRIGGER_MODE is “PRE” trigger mode |
| TRIGGER_MODE | 1 Octet | Specifies Acquisition mode to be used.<br>0x00 – POST Mode<br>0x01 – RFU<br>0x02 – PRE Mode<br>0x03 – 0xFF – Invalid |
| RAM_PAGE_WIDTH | 1 Octet | Specifies the amount of on-chip memory that is covered by an acquisition. Granularity is chosen by design as 256 Bytes (i.e. 64 32-bits words).<br>Valid values are as below:<br>0x00h - 256 bytes<br>0x01h - 512 bytes<br>0x02h - 768 bytes<br>0x03h - 1024 bytes<br>0x04h - 1280 bytes<br>0x05h - 1536 bytes<br>0x06h - 1792 bytes<br>0x07h - 2048 bytes<br>0x08 - 0xFF - Invalid |

**Table 86. CTS_CONFIG_CMD parameters***...continued*

| Payload Field(s) | Length | Value/Description |
|---|---|---|
| SAMPLE_CLK_DIV | 1 Octet | The decimal value of this field specifies the clock rate division factor to be used during acquisition.<br>CTS clock = 13.56 MHz / $2^{SAMPLE\_CLK\_DIV}$<br>00 - 13560 kHz<br>01 - 6780 kHz<br>02 - 3390 kHz<br>03 - 1695 kHz<br>04 - 847.5 kHz<br>05 - 423.75 kHz<br>06 - 211.875 kHz<br>07 - 105.9375 kHz<br>08 - 52.96875 kHz<br>09 - 26.484375 kHz<br>10 - 13.2421875 kHz<br>11 - 6.62109375 kHz<br>12 - 3.310546875 kHz<br>13 - 1.6552734375 kHz<br>14 - 0.82763671875 kHz<br>15 - 0.413818359375 kHz |
| SAMPLE_BYTE_SEL | 1 Octet | These bits are used to specify which bytes of the two 16-bits input buses contribute to the interleave mechanism that generates data to be transferred to the on-chip memory. The meaning and usage of them is depending from the SAMPLE_MODE_SEL values.<br>Note: Given value is always masked with 0x0F and then effective value is considered. |
| SAMPLE_MODE_SEL | 1 Octet | Selects the sampling interleave mode as described by the CTS design specs. Decimal value 3 is reserved and will be treated as 0.<br>Note: Given value is always masked with 0x03, and then effective value is considered. |
| TB0 | 1 Octet | Selects which test bus to be connected to TB0.[1] |
| TB1 | 1 Octet | Selects which test bus to be connected to TB1.[1] |
| TB2 | 1 Octet | Selects which test bus to be connected to TB2.[1] |
| TB3 | 1 Octet | Selects which test bus to be connected to TB3.[1] |
| TTB_SELECT | 1 Octet | Selects which TB to be connected to the trigger sources.[1] |
| RFU | 4 Octets | Send always 0x00000000 |
| MISC_CONFIG | 24 Octets | Trigger occurrences, polarity etc.[2] |

[1] Refer to TB_Signal_Index value in Section 20.
[2] To know which CTS configuration to use, refer to (CTS configuration part of NFC Cockpit.

**Table 87. CTS_CONFIG_RSP**

| GID | OID | Numbers of parameter(s) | Description |
|---|---|---|---|
| 1111b | 0x22 | 1 | PN722X reports if the CTS_CONFIG_CMD is successful or not. |

In case the NCI package is malformed (wrong length for example) STATUS_SYNTAX_ERROR is returned. If parameters are out of bound (forbidden values as stated above), STATUS_INVALID_PARAM is sent.

**Table 88. CTS_CONFIG_RSP parameters**

| Payload Field(s) | Length | Value/Description | |
|---|---|---|---|
| STATUS | 1 Octet | 0x00 | STATUS_OK |
| | | 0x05 | STATUS_SYNTAX_ERROR |
| | | 0x09 | STATUS_INVALID_PARAM |
| | | Others | RFU |

### 14.4.2 CTS_CONTROL_CMD/RSP

This command is used by DH to Enable/Disable the CTS.

**Table 89. CTS_CONTROL_CMD**

| GID | OID | Numbers of parameter(s) | Description |
|---|---|---|---|
| 1111b | 0x22 | 0x02 | Command to Control the CTS configuration |

**Table 90. CTS_CONTROL_CMD parameters**

| Payload Field(s) | Length | Value/Description |
|---|---|---|
| COMMAND | 1 Octet | 0x81: CTS_CONTROL_CMD |
| CONTROL | 1 Octet | 0x00: Disable<br>0x01: Enable |

**Table 91. CTS_CONTROL_RSP**

| GID | OID | Numbers of parameter(s) | Description |
|---|---|---|---|
| 1111b | 0x22 | 1 | Response to inform the DH of the status of the command |

**Table 92. CTS_CONTROL_RSP parameters**

| Payload Field(s) | Length | Value/Description | |
|---|---|---|---|
| STATUS | 1 Octet | 0x00 | STATUS_OK |
| | | 0x05 | STATUS_SYNTAX_ERROR |
| | | 0x06 | SEMANTIC_ERROR |
| | | 0x09 | STATUS_INVALID_PARAM |
| | | Others | RFU |

### 14.4.3 CTS_LOG_NTF

The notification sends the CTS log data of the captured test bus data samples stored in the PN722X memory buffer to DH.

**Table 93. CTS_LOG_NTF**

| GID | OID | Numbers of parameter(s) | Description |
|---|---|---|---|
| 1111b | 0x22 | n + 1 | CTS notification sent in case CTS Trigger is met using NCI Chaining to DH |

**Table 94. CTS_LOG_NTF Parameters**

| Payload Field(s) | Length | Value/Description |
|---|---|---|
| CTS Log Data [1…n] | n | Captured Samples CTS Log Data chunk |

## 14.5 LOAD_RF_CONFIG_CMD/RSP

This command is used to load the RF configuration from E2PROM into internal CLIF registers. RF configuration refers to a unique combination of RF Technology, mode (target/initiator) and baud rate. RF configuration can be loaded separately for the CLIF receiver (RX configuration) and transmitter (TX configuration) path. The value 0xFF must be used if the corresponding configuration for a path will not be changed.

**Conditions**

Field ‘TX Configuration’ must be in the range from `0x00 – 0x2B`, inclusive. If the value is `0xFF`, TX configuration is not changed.

Field ‘RX Configuration’ must be in the range from `0x80 – 0xAB`, inclusive. If the value is `0xFF`, RX configuration is not changed.

**Table 95. LOAD_RF_CONFIG_CMD**

| GID | OID | Numbers of parameter(s) | Description |
|---|---|---|---|
| 1111b | 0x0E | 2 | Load RF Configuration from E2PROM into internal CLIF Registers. |

**Table 96. LOAD_RF_CONFIG_CMD parameters**

| Payload Field(s) | Length | Value/Description | |
|---|---|---|---|
| TX Configuration | 1 Octet | | |
| | | 0xFF | TX RF Configuration not changed |
| | | 0x00 – 0x2B | Corresponding TX RF Configuration loaded |
| | | Others | RFU |
| RX Configuration | 1 Octet | | |
| | | 0xFF | RX RF Configuration not changed |
| | | 0x80 – 0xAB | Corresponding RX RF Configuration loaded |
| | | Others | RFU |

**Table 97. LOAD_RF_CONFIG_RSP**

| GID | OID | Numbers of parameter(s) | Description | |
|---|---|---|---|---|
| 1111b | 0x0E | 1 | Returns a status byte if the Load RF Config is successful or not. | |
| | | | 0x00 | STATUS_OK |
| | | | 0x05 | STATUS_SYNTAX_ERROR |
| | | | 0x09 | STATUS_INVALID_PARAM |
| | | | Others | RFU |

## 14.6 PASSTHROUGH_RF_EXCHANGE_CMD/RSP

This command performs a transmission of the TX data and waits for the reception of any RX data and work only in Initiator mode. The function returns in case of a reception (either erroneous or correct) or a timeout happened. The timer is started with the END of TRANSMISSION and stopped with the START of RECEPTION. Timeout value needs to be configured by the user before execution of Exchange command.

If transceiver_state is

•IDLE: the TRANSCEIVE mode is entered.

• WAIT_RECEIVE: the transceiver state is reset to TRANSCEIVE MODE.

Before sending the command to PN722X, execute LOAD_RF_CONFIG_CMD and SWITCH_RF_FIELD_CMD for Field ON.

**Table 98. PASSTHROUGH_RF_EXCHANGE_CMD**

| GID | OID | Numbers of parameter(s) | Description |
|---|---|---|---|
| 1111b | 0x0D | n Octet | TX data which must be sent out via CLIF using transceive command. n = 1 – 255 bytes. |

**Table 99. PASSTHROUGH_RF_EXCHANGE_RSP**

| GID | OID | Numbers of parameter(s) | Description |
|---|---|---|---|
| 1111b | 0x0D | 1 + n Octet | PN722X reports if the PASSTHROUGH_RF_EXCHANGE CMD is successful or not and the received RF data if successful. |

In case the command is sent in any state other than RFST_IDLE, STATUS_SEMANTIC_ERROR will be returned.

**Table 100. PASSTHROUGH_RF_EXCHANGE_RSP parameters**

| Payload Field(s) | Length | Value/Description |
|---|---|---|
| STATUS | 1 Octet | 0x00 STATUS_OK<br>0x01 STATUS_REJECTED<br>0x05 STATUS_SYNTAX_ERROR<br>0x06 STATUS_SEMANTIC_ERROR<br>0xF1 STATUS_IO_TIMEOUT_ERROR<br>0xF2 STATUS_INTEGRITY_ERROR<br>0xF3 STATUS_COLLISION_ERROR<br>Others RFU |
| RF Received data | n Octet | RX data which is received via CLIF using excahnge command. n = 1 – 255 bytes. |

## 14.7 SWITCH_RF_FIELD_CMD/RSP

This command is used to control the Field by providing options to turn On, Off and reset the RF Field.

The DH must issue LOAD_RF_CONFIG_CMD before issuing SWITCH_RF_FIELD_CMD.

**Table 101. SWITCH_RF_FIELD_CMD**

| GID | OID | Numbers of parameter(s) | Description |
|---|---|---|---|
| 1111b | 0x3F | 3 | Switch RF Field On/Off |

**Table 102. SWITCH_RF_FIELD_CMD parameters**

| Payload Field(s) | Length | Value/Description | |
|---|---|---|---|
| Test ID | 1 Octet | 0x32 RF Field Control | |
| Field Type | 1 Octet | 0x00 | Field Off |
| | | 0x01 | Field On |
| | | 0x02 | Field Reset |
| | | Others | RFU |
| Field On Config | 1 Octet | Applicable when Field On and Field Reset are used as Field Type. | |
| | | 0 | Use collision avoidance |
| | | 1 | Disable collision avoidance |

**Table 103. SWITCH_RF_FIELD_RSP**

| GID | OID | Numbers of parameter(s) | Description | |
|---|---|---|---|---|
| 1111b | 0x3F | 1 | Returns a status byte if the RF Field could not be successfully controlled. | |
| | | | 0x00 | STATUS_OK |
| | | | 0x05 | STATUS_SYNTAX_ERROR |
| | | | 0xF1 | EXTERNAL RF ERROR |
| | | | 0xF2 | TX LDO ERROR |
| | | | 0xF3 | Invalid Load RF Configuration - LOAD_RF_CONFIG_CMD needs to be successful |
| | | | Others | RFU |

## 14.8 TESTBUS_CONFIG_CMD/RSP

### 14.8.1 TESTBUS_CLEAR_CONFIG_CMD/RSP

The DH uses the command to clear/disable the test bus configuration already enabled by TESTBUS_CONFIG_CMD.

**Table 104. TESTBUS_CLEAR_CONFIG_CMD**

| GID | OID | Numbers of parameter(s) | Description |
|---|---|---|---|
| 1111b | 0x31 | 0x01 | Command to control test bus configuration |

**Table 105. TESTBUS_CLEAR_CONFIG_CMD parameters**

| Payload Field(s) | Length | Value/Description |
|---|---|---|
| COMMAND | 1 Octet | 0x00: TESTBUS_CLEAR_CONFIG_CMD |

**Table 106. TESTBUS_CLEAR_CONFIG_RSP**

| GID | OID | Numbers of parameter(s) | Description |
|---|---|---|---|
| 1111b | 0x31 | 1 | PN722X reports if the TESTBUS_CLEAR_CONFIG_CMD is successful or not. |

In case the NCI package is malformed (wrong length for example) STATUS_SYNTAX_ERROR is returned. If parameters are out of bounds (forbidden values as stated above), STATUS_INVALID_PARAM are sent.

**Table 107. TESTBUS_CLEAR_CONFIG_RSP parameters**

| Payload Field(s) | Length | Value/Description | |
|---|---|---|---|
| STATUS | 1 Octet | 0x00 | STATUS_OK |
| | | 0x05 | STATUS_SYNTAX_ERROR |
| | | 0x09 | STATUS_INVALID_PARAM |
| | | Others | RFU |

### 14.8.2 TESTBUS_DIGITAL_CONFIG_CMD/RSP

This command is used to switch available digital test bus signal on selected pad configurations.

**Table 108. TESTBUS_DIGITAL_CONFIG_CMD**

| GID | OID | Numbers of parameter(s) | Description |
|---|---|---|---|
| 1111b | 0x31 | 0x04 | Command to configure Digital test bus configuration |

**Table 109. TESTBUS_DIGITAL_CONFIG_CMD parameters**

| Payload Field(s) | Length | Value/Description |
|---|---|---|
| COMMAND | 1 Octet | 0xDD: TESTBUS_DIGITAL_CONFIG_CMD |
| SIGNAL INDEX | 1 Octet | As provided in Annex D |
| BIT INDEX | 1 Octet | Bit Index in Signal that is required on Selected PAD. |
| PAD INDEX | 1 Octet | PAD to be used.<br>0x00 – AUX1<br>0x01 – AUX2<br>0x02 – RFU<br>0x03 – GPIO0 (CT feature cannot be used simultaneously)<br>0x04 – GPIO1 (CT feature cannot be used simultaneously)<br>0x05 – 0xFF – Invalid |

**Table 110. TESTBUS_DIGITAL_CONFIG_RSP**

| GID | OID | Numbers of parameter(s) | Description |
|---|---|---|---|
| 1111b | 0x31 | 1 | PN722X reports if the TESTBUS_DIGITAL_CONFIG_CMD is successful or not. |

In case the NCI package is malformed (wrong length for example) STATUS_SYNTAX_ERROR is returned. If parameters are out of bounds (forbidden values as stated above), STATUS_INVALID_PARAM are sent.

**Table 111. TESTBUS_DIGITAL_CONFIG_RSP parameters**

| Payload Field(s) | Length | Value/Description |
|---|---|---|
| STATUS | 1 Octet | 0x00 – STATUS_OK<br>0x05 – STATUS_SYNTAX_ERROR<br>0x09 – STATUS_INVALID_PARAM<br>Others – RFU |

### 14.8.3 TESTBUS_ANALOG_CONFIG_CMD/RSP

This command is used to get available analog test bus signal on selected pad configurations. The signal on analog test bus can be obtained in RAW mode and COMBINED mode.

#### 14.8.3.1 RAW mode

In this mode, the signal chosen by `TB_SignalIndex0` is shifted by `Shift_Index0`, masked with `Mask0` and output on `AUX1`. Similarly, the signal chosen by `TB_SignalIndex1` is shifted by `Shift_Index1`, masked with `Mask1` and output on `AUX2`.

This mode offers flexibility for the customer to output any signal that is 8 bits wide or lesser and not requiring sign conversion to be output onto the analog pads.

#### 14.8.3.2 COMBINED mode

In this mode, analog signal is the 10 bit signed ADCI/ADCQ/pcrm_if_rssi value converted to an unsigned value, scaled back to 8 bits and then output on either `AUX1` or `AUX2` pads.

Only one of either `ADCI/ADCQ` (10-bit) converted values can be output to `AUX1/AUX2` at any time.

#### 14.8.3.3 Note

The host must provide all the fields, regardless of field applicability in “raw” or “combined” mode. The PN722X IC only considers the applicable field values.

**Table 112. TESTBUS_ANALOG_CONFIG_CMD**

| GID | OID | Numbers of parameter(s) | Description |
|---|---|---|---|
| 1111b | 0x31 | 0x09 | Command to configure analog test bus configuration |

**Table 113. TESTBUS_ANALOG_CONFIG_CMD parameters**

| Payload Field(s) | Length | Value/Description |
|---|---|---|
| COMMAND | 1 Octet | 0xDE: TESTBUS_ANALOG_CONFIG_CMD |
| CONFIG | 1 Octet | Configurable bits, refer to Table below |
| COMBINED MODE SIGNAL | 1 Octet | 0x00 – ADCI/ADCQ<br>0x01 – pcrm_if_rssi<br>0x02 – Analog and Digital test bus combined<br>0x03 - 0xFF – Reserved |
| TB_SIGNAL_INDEX0 | 1 Octet | As provided in Annex D |
| TB_SIGNAL_INDEX1 | 1 Octet | As provided in Annex D |
| SHIFT_INDEX0 | 1 Octet | DAC0 input shift positions. Direction will be decided by bit1 in CONFIG. |
| SHIFT_INDEX1 | 1 Octet | DAC1 input shift positions. Direction will be decided by bit2 in CONFIG. |
| MASK0 | 1 Octet | DAC0 Mask. |
| MASK1 | 1 Octet | DAC1 Mask. |

**Table 114. CONFIG BITMASK**

| b7 | b6 | b5 | b4 | b3 | b2 | b1 | b0 | Description | Applicable to Mode |
|---|---|---|---|---|---|---|---|---|---|
| x | x | | | | | | | DAC1 output shift<br>Range – 0, 1, 2 | Raw |
| | | x | x | | | | | DAC0 output shift<br>Range – 0, 1, 2 | Raw |
| | | | | x | | | | In combined mode, signal on AUX1/AUX2 pin<br>0 ➔ Signal on AUX1<br>1 ➔ Signal on AUX2 | Combined |
| | | | | | x | | | DAC1 input shift direction<br>0 ➔ Shift right<br>1 ➔ Shift left | Raw |
| | | | | | | x | | DAC0 input shift direction<br>0 ➔ Shift right<br>1 ➔ Shift left | Raw |
| | | | | | | | x | Mode.<br>0 ➔ Raw mode<br>1 ➔ Combined mode | Raw/Combined |

**Table 115. TESTBUS_ANALOG_CONFIG_RSP**

| GID | OID | Numbers of parameter(s) | Description |
|---|---|---|---|
| 1111b | 0x31 | 1 | PN722X reports if the TESTBUS_ANALOG_CONFIG_CMD is successful or not. |

In case the NCI package is malformed (wrong length for example) STATUS_SYNTAX_ERROR is returned. If parameters are out of bounds (forbidden values as stated above), STATUS_INVALID_PARAM will be sent.

**Table 116. TESTBUS_ANALOG_CONFIG_RSP parameters**

| Payload Field(s) | Length | Value/Description | |
|---|---|---|---|
| STATUS | 1 Octet | 0x00 | STATUS_OK |
| | | 0x05 | STATUS_SYNTAX_ERROR |
| | | 0x09 | STATUS_INVALID_PARAM |
| | | Others | RFU |

# 15 PN722X NFCC encrypted secured firmware upload mode

## 15.1 Introduction

Part of the PN722X NFCC firmware code is permanently stored in the ROM, while the rest of the code and the data are stored in the embedded flash. User data is stored in flash and is protected by anti-tearing mechanisms that ensure the integrity and availability of the data. In order to provide NXP's customers with features that are compliant with the latest standards (for instance NCI etc.), both the code and user data in FLASH can be updated.

NXP is in charge of delivering new firmware updates, together with new User data. The update procedure must be equipped with a mechanism to protect the authenticity, integrity, and confidentiality of NXP code and data.

The authenticity and integrity are protected by asymmetric/symmetric key signature and reverse chained hash mechanism. The first SecuredWrite command contains the hash of the second command and is protected by an RSA signature. PN722X firmware uses the RSA public key to authenticate the first command. The chained hash in each command is used to authenticate the subsequent command. To ensure the firmware code and data is not accessed by third parties.

The payloads of the SecuredWrite commands are encrypted. After authentication of each command, the payload content is decrypted and written to flash by PN722X firmware.

PN722X supports hardware crypto assisted download protocol, that uses the on chip hardware crypto blocks.

The aim of this section is to detail the PN722X NFCC firmware update concepts.

## 15.2 DH interface

As already mentioned in Section 6, the PN722X NFCC has two main modes of operation to communicate with the DH:

1. NCI-based communications
2. HDLL-Based communications, only used when the PN722X NFCC is triggered to enter the "secured firmware upload" mode

Most of the information given in Section 6 still applies here; the only differences are that standby is not used and the frames exchanged do not carry NCI packets anymore, they carry HDLL packets.

The next chapters describes:

- how to trigger the "secured firmware upload" mode
- the HDLL (Host Data Link Layer) packets
- the mapping on Host Interface

### 15.2.1 How to trigger the “secured firmware upload” mode

To force the PN722X NFCC to enter the “secured firmware upload” mode, the DH must proceed as described below. The PN722X NFCC does not need a dedicated NFC_DWL_REQ pin.

1. Either generate a hardware Reset using an RSTN pulse. The NFCC waits for any command during 5 seconds.
   a. If NCI command is sent, the NFCC enters NCI mode (only if download was successful).
   b. If HDLL command for download is sent, the NFCC enters download mode.
2. Or send the NCI reset command `CORE_RESET_CMD` with the proprietary parameter 0x80. Once the Host receives a successful response, the NFCC automatically enters download mode.

PN722X NFCC is now ready to receive commands in “secured firmware upload” mode, using the HDLL framing.

To get out of the “secured firmware upload” mode and reverse to the NCI mode, the DH must generate a `DL_RESET` command and wait for 5 ms for the NFCC to reboot. If no response is received during that time, the DH can consider that reset was successful.

PN722X NFCC is now ready to receive NCI commands, assuming that the previous “secured firmware upload” was successful.

***Note:*** *In case of battery removal during download, NFCC remains in download mode after the next reboot. Only a successful download can recover the NFCC in NCI mode to allow NFC use cases.*

### 15.2.2 Description of HDLL

HDLL is the link layer developed by NXP to ensure a reliable upload mode.

An HDLL message is made of a 2-Byte header, followed by a frame, comprising the opcode and the Payload of the command. Each message ends with a 16-bit CRC, as shown in Figure 42.

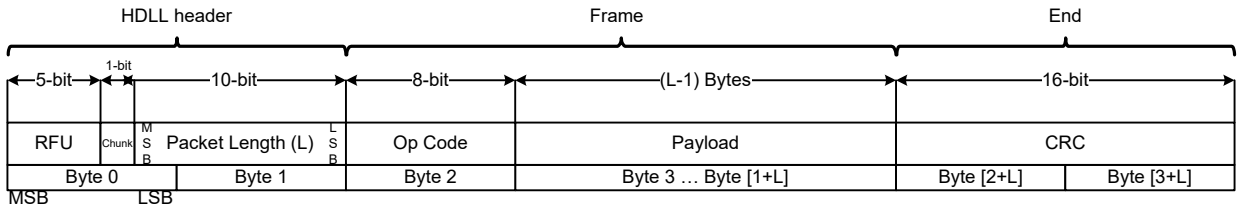



**Figure 42. HDLL frame**

The HDLL header contains:

- A chunk bit: indicates if this is the only or last chunk of a message (chunk = 0) or if at least one other chunk follows (chunk = 1).
- The length of the Payload coded on 10 bits. So, the HDLL Frame Payload can go up to 1023 Bytes.

The byte order has been defined as big-endian, meaning MSByte first.

The CRC16 is compliant to X.25 (CRC-CCITT, ISO/IEC13239) standard with polynome x^16 + x^12 + x^5 +1 and preload value 0xFFFF. It is calculated over the whole HDLL frame (Header + Frame).

Sample C-Code implementation:

```
static uint16_t phHal_Host_CalcCrc16(uint8_t* p, uint32_t dwLength)
{
   uint32_t i ;
   uint16_t crc_new ;
   uint16_t crc = 0xffffU;
   for (I = 0; i < dwLength; i++)
   {
      crc_new = (uint8_t)(crc >> 8) | (crc << 8 );
      crc_new ^= p[i];
      crc_new ^= (uint8_t)(crc_new & 0xff) >> 4;
      crc_new ^= crc_new << 12;
      crc_new ^= (crc_new & 0xff) << 5;
      crc = crc_new;
   }
   return crc;
}
```

### 15.2.3 Transport mapping over $I^2C$

#### 15.2.3.1 Write sequence from the DH (direction DH => PN722X NFCC)

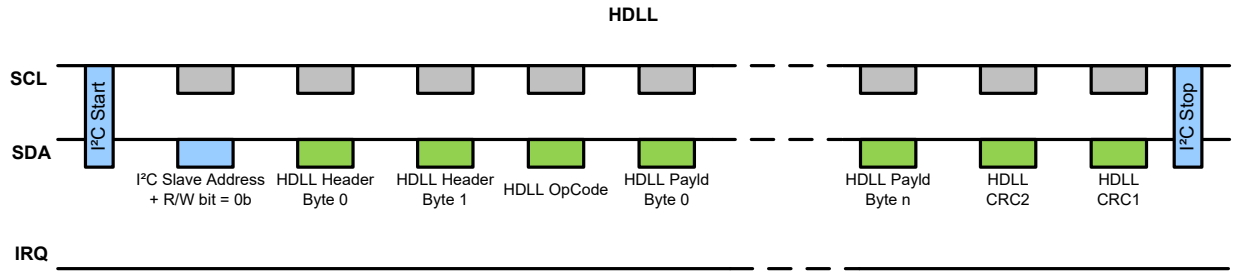



**Figure 43. $I^2C$ write sequence**

#### 15.2.3.2 Read sequence from the DH (Direction PN722X NFCC => DH)

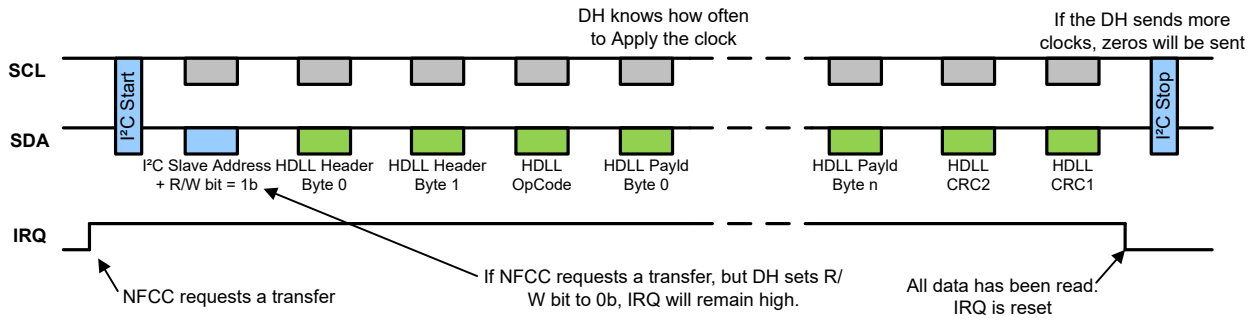



**Figure 44. Figure Caption**

### 15.2.4 Transport mapping over SPI

#### 15.2.4.1 Write sequence from the host (direction DH => PN722X)

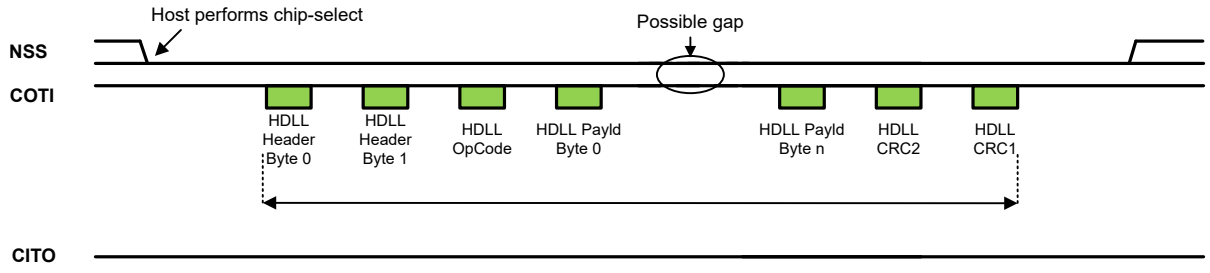



**Figure 45. SPI Write sequence**

#### 15.2.4.2 Read sequence from the host (direction PN722X => DH)

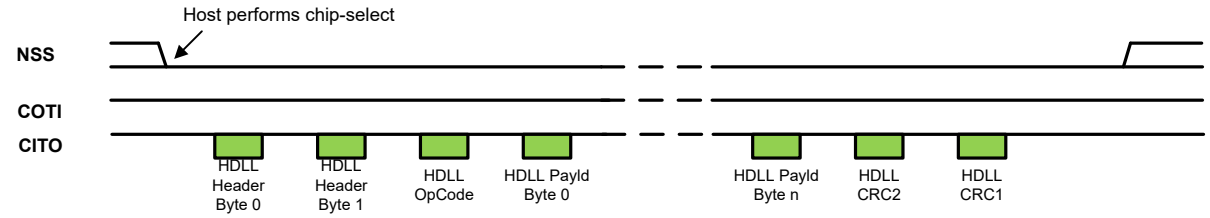



**Figure 46. SPI Read sequence**

## 15.3 Firmware signature and version control

In the PN722X NFCC upload mode, only a firmware signed and delivered by NXP is accepted.

During the upload, a new 16-bit firmware version is sent. The version is composed of a major and a minor number:

1. Major number: 8 bits (MSB)
2. Minor number: 8 bits (LSB)

The PN722X NFCC checks whether the new major version number is bigger or equal to the current one. If not, the secured firmware upload is rejected and the session is closed.

The firmware version number is anti-tearing tampered.

## 15.4 Upload procedure

Only NFC_DWL_REQ pin-less upload procedure exists for PN722X. PN722X does not supports NFC_DWL_REQ pin for download. Instead, download is supported using commands sent during cold boot or during normal operation.

Pre-condition: PN722X is in Operation state.

Scenario1:

1. DH performs a hard-reset to boot the PN722X with HOSTIF_SEL0 pulled down to ‘0’.
2. DH shall use the HIF1 in I2C configuration to send HDLL commands.
3. DH reads the current hardware and firmware version from the Device
   a. DH checks session status if last download was completed
   b. DH applies the version checking rules to decide whether to start the download or exit download.
4. DH loads from a file the firmware binary code to be downloaded
5. DH provides a first secured write command that contains:
   a. The version of the new firmware,
   b. A 16-byte nonce of arbitrary values used for encryption key obfuscation
   c. A digest value of the next frame,
   d. The digital signature of the frame itself
6. The DH loads the secure download protocol sequence to the PN722X with secured write commands
7. When the last secured write command has been sent, the DH executes the CheckIntegrity command to check if the memories have been successfully written.
8. DH reads the new firmware version and checks the session status if closed for reporting to the upper layer .
9. DH performs an NFC controller hard reset to reboot the NFC controller in NCI mode.

Post-Condition: The firmware is updated; new firmware version number is reported.

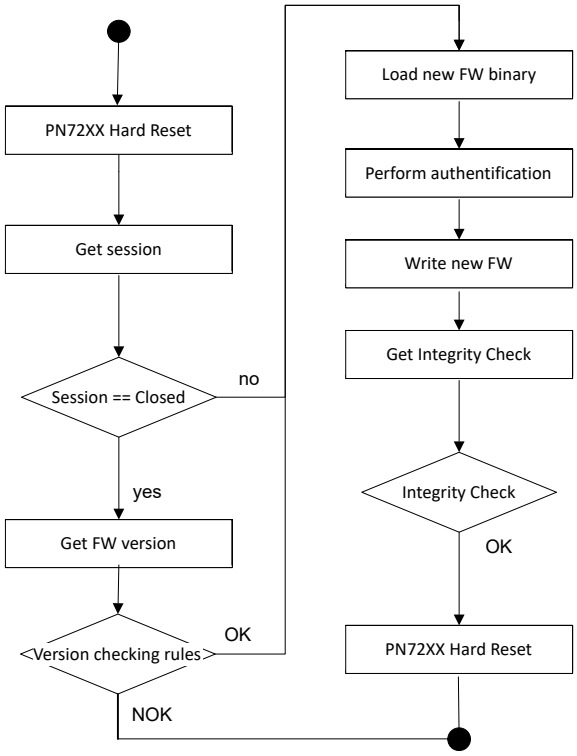



**Figure 47. DL procedure using VEN toggle with HOSTIF_SEL0 pulled down to '0'**

Scenario2:

1. DH performs a hard-reset to boot the PN722X with both HOSTIF_SEL0 pulled up to ‘1’ and Mode_Switch_GPI pulled up to ‘1’.
2. DH shall use the HIF1-SPI configuration to send HDLL commands.
3. DH reads the current hardware and firmware version from the Device
   a. DH checks session status if last download was completed
   b. DH applies the version checking rules to decide whether to start the download or exit download.
4. DH loads from a file the firmware binary code to be downloaded
5. DH provides a first secured write command that contains:
   a. The version of the new firmware,
   b. A 16-byte nonce of arbitrary values used for encryption key obfuscation
   c. A digest value of the next frame,
   d. The digital signature of the frame itself
6. The DH loads the secure download protocol sequence to the PN722X with secured write commands
7. When the last secured write command has been sent, the DH executes the CheckIntegrity command to check if the memories have been successfully written.
8. DH reads the new firmware version and checks the session status if closed for reporting to the upper layer
9. DH performs an NFC controller hard reset to reboot the NFC controller in NCI mode

Post-Condition: The firmware is updated; new firmware version number is reported.

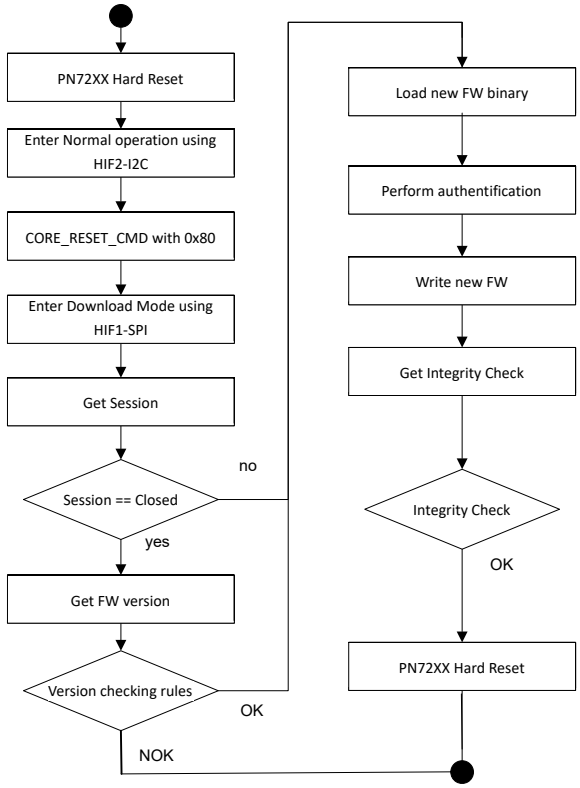



**Figure 48. DL procedure using VEN toggle with HOSTIF_SEL0 pulled up to ‘1’ and Mode_Switch_GPI pulled up to '1'.**

## 15.5 HDLL commands glossary

### 15.5.1 HDLL protocol

#### 15.5.1.1 Introduction

HDLL is a command response protocol. All the operations mentioned above are triggered through a specific command and validated based on the response.

Commands and responses follow HDLL message syntax shown in Figure 42. The DH sends the command, and NFCC sends the response. The opcode indicates the command and response type.

HDLL-based communications are only used when the PN722X is triggered to enter the **"secured firmware upload"** mode.

#### 15.5.1.2 HDLL command opcodes

***Note:*** *HDLL command frames are 4 bytes aligned. Unused payload bytes are left nil.*

**Table 117. List of HDLL command opcodes for HWCEDL**

| Opcode | Command Alias | Description |
|---|---|---|
| 0xE5 | DL_RESET | Performs a soft reset |
| 0xE1 | DL_GET_VERSION | Returns the version numbers |
| 0xDB | DL_GET_SESSION_STATE | Returns the current session state |
| 0xDF | DL_GET_DIE_ID | Returns the die ID |
| 0xE7 | DL_CHECK_INTEGRITY | Checks and return the CRCs over the different areas as well as pass/fail status flags for each |
| 0x8C | DL_SEC_WRITE | Writes x bytes to memory starting at absolute address y |

#### 15.5.1.3 HDLL response opcodes

***Note:*** *HDLL response frames are 4 bytes aligned. Unused payload bytes are left nil. Only DL_OK responses can contain payload values.*

**Table 118. List of HDLL response opcodes**

| Opcode | Response Alias | Description |
|---|---|---|
| 0x00 | DL_OK | Command passed |
| 0x01 | DL_INVALID_ADDR | Address not allowed |
| 0x0B | DL_UNKNOW_CMD | Unknown command |
| 0x0C | DL_ABORTED_CMD | Chunk sequence is too large |
| 0x1E | DL_ADDR_RANGE_OFL_ERROR | Address out of range |
| 0x1F | DL_BUFFER_OFL_ERROR | Buffer is too small |
| 0x20 | DL_MEM_BSY | Memory busy |
| 0x21 | DL_SIGNATURE_ERROR | Signature mismatch |
| 0x24 | DL_FIRMWARE_VERSION_ERROR | Current version equal or higher |
| 0x28 | DL_PROTOCOL_ERROR | Protocol error |
| 0x2A | DL_SFWU_DEGRADED | Flash data corruption |
| 0x2D | PH_STATUS_DL_FIRST_CHUNK | First chunk received |
| 0x2E | PH_STATUS_DL_NEXT_CHUNK | Wait for the next chunk |
| 0xC5 | PH_STATUS_INTERNAL_ERROR_5 | Length mismatch |

### 15.5.2 Reset command

Frame exchange:

```
[HDLL] -> [0x00 0x04 0xE5 0x00 0x00 0x00 0xBF 0xB9]
[HDLL] <- [0x00 0x04 STAT 0x00 CRC16]
```

The reset prevents the PN722X from sending the `DL_STATUS_OK` answer. Therefore, only erroneous status can be received.

`STAT` is the return status.

### 15.5.3 Get version command

Frame exchange:

```
[HDLL] -> [0x00 0x04 0xE1 0x00 0x00 0x00 0x75 0x48]
[HDLL] <- [0x00 0x08 STAT HW_V RO_V MODEL_ID FM1V FM2V RFU1 RFU2 CRC16]
```

Table 119 shows the payload frame of the `GetVersion` response.

**Table 119. Response to the GetVersion command**

| Field | Byte | Description |
|---|---|---|
| STAT | 1 | Status |
| HW_V | 2 | Hardware version |
| RO_V | 3 | ROM code |
| MODEL_ID | 4 | Model ID |
| FMxV | 5-6 | Firmware version (used for download) |
| RFU1-RFU2 | 7-8 | - |

### 15.5.4 Get session state command

Frame exchange:

```
[HDLL] -> [0x00 0x04 0xDB 0x00 0x00 0x00 0x31 0x0A]
[HDLL] <- [0x00 0x04 STAT SSTA RFU CRC16]
```

Table 120 shows the payload frame of the `GetSession` response.

**Table 120. Response to the GetSession command**

| Field | Byte | Description |
|---|---|---|
| STAT | 1 | Status |
| SSTA | 2 | Session state<br>• 0: closed<br>• 1: open<br>• 2: locked (download no more allowed) |
| RFU | 3-4 | |

### 15.5.5 Get die ID command

Frame exchange:

```
[HDLL] -> [0x00 0x04 0xDF 0x00 0x00 0x00 0xFB 0xFB]
[HDLL] <- [0x00 0x14 STAT 0x00 0x00 0x00 ID0 ID1 ID2 ID3 ID4 ID5 ID6 ID7 ID8 ID9 ID10
 ID11 ID12 ID13 ID14 ID15 CRC16]
```

Table 121 shows the payload frame of the `GetDieId` response.

**Table 121. Response to the GetDield command**

| Field | Byte | Description |
|---|---|---|
| STAT | 1 | Status |
| RFU | 2-4 | |
| DIEID | 5-20 | ID of the die (16 bytes) |

### 15.5.6 Check integrity command

Frame exchange:

```
[HDLL] -> [0x00 0x04 0xE7 0x00 0x00 0x00 0x52 0xD1]
[HDLL] <- [0x00 0x88 STAT LEN_DATA LEN_CODE 0x00 [CRC_INFO] [CRC32] CRC16]
```

Table 122 shows the payload frame of the `CheckIntegrity` response.

**Table 122. Response to the CheckIntegrity command**

| Field | Byte | Description | | |
|---|---|---|---|---|
| STAT | 1 | Status | | |
| LEN_DATA | 2 | Total number of data sections | | |
| LEN_CODE | 3 | Total number of code sections | | |
| RFU | 4 | Reserved | | |
| [CRC_INFO] | 5-8 | 32 bits (little-endian). If a bit is set, the CRC of the corresponding section is checked OK, otherwise KO | | |
| | | **Bit** | **Area integrity status** | **Note** |
| | | [31:24] | Reserved | 1 |
| | | [23] | Reserved | 3 |
| | | [22] | NXP Area | 4 |
| | | [21:20] | Reserved | 1 |
| | | [19] | APP_RF_CFG_IC (Reserved) | 1 |
| | | [18] | APP_RF_CFG_LIB (accessible or updated using 0xA2 Prop Config) | 1 |
| | | [17] | APP_USER_CFG (Reserved) | 1 |
| | | [16] | APP_RF_PROTOCOL_CFG (accessible or updated using 0xA00D Prop Config) | 1 |
| | | [15] | Reserved | 1 |
| | | [14:6] | Reserved | 3 |
| | | [5:0] | Reserved | 1 |
| [CRC32] | 9-136 | CRC32 of the 32 sections. Each CRC is of 4 bytes stored in little-endian format. First 4-bytes of CRC is of bit CRC_INFO[31], next 4-bytes of CRC is of bit CRC_INFO[30] and so on. | | |

Notes:

- [1] This bit must be 1 for the PN76 to function properly (with features and or encrypted FW download).
- [2] This bit is set to 1 by default, but user modified settings will invalidate the CRC. No effect on PN76 functionality.
- [3] This bit value, even if it is 0, is not relevant. This bit value can be ignored.
- [4] This bit value, even if it is 0, is not relevant. This bit value can be ignored. Secure FW update procedure will recover this area.

### 15.5.7 Secure write command

The secure write command is used in a sequence of secure write commands: the encrypted “secured firmware upload” (often referred to as eSFWu).

The secure write command first opens the download session and passes the RSA authentication. The following commands pass the encrypted addresses and bytes to write into the PN722X Flash. All commands except the last one include the next one hash to confirm they are not the last, and cryptographically bonding the sequence frames together.

Other commands (except `DL_RESET and DL_CHECK_INTEGRITY`) can be inserted between the secured write commands of a sequence without breaking the sequence.

#### 15.5.7.1 First secured write command

A secured write command is the first command if and only if:

1. The opcode is 0x8C.
2. The frame length is 312 bytes.
3. No secured write command has been received since the last reset.
4. PN722X verified the embedded signature.

**First** frame format exchange:

```
[HDLL] -> [FLEN 0x8C 0x0A VERS NONCE HASH SIGN CRC16]
[HDLL] <- [0x00 0x04 STAT 0x00 0x00 0x00 CRC16]
```

`FLEN` is the size of frame payload (bit 0 to 9 of the 2 first bytes): 0x0134 = 308 for the first frame.

`REGN` is 0x0A – NXP firmware region.

`VERS` is the 16 bits version of the firmware update sequence.

`NONCE` is 16 bytes arbitrary data used for encryption key obfuscation.

`HASH` is the 32 bytes digest of the second frame.

`SIGN` is the 256 bytes digital signature of the first frame.

`STAT` is the return status.

At least one chunk of data must be written during a eSFWu even though the data written may only be one-byte long. The first command always includes the hash of the next command since there are at least two commands.

### 15.5.7.2 Middle secured write command

A secured write command is a middle command if and only if:

1. The opcode is 0x8C.
2. A first secured write command has already been received and successfully verified prior.
3. No reset has occurred since the first secured write command was received.
4. The frame length is equal to the data size + header size + hash size: `FLEN = SIZE + 6 + 32`.
5. The digest of the whole frame is equal to the hash value received in the previous frame.

**Middle** frame format exchange:

```
[HDLL] -> [FLEN 0x8C SIZE ADDR {DATA} HASH CRC16]
[HDLL] <- [0x00 0x04 STAT 0x00 0x00 0x00 CRC16]
```

`STAT` is the return status.

`FLEN` is the size of frame (bit 0 to 9 of the 2 first bytes)

`SIZE` is the 16 bits size of data.

***Note: Data size max value is 512.***

`ADDR` is the 24 bits address of the data to write flash.

***Note: ADDR is encrypted.***

`{DATA}` is the chunk of data.

***Note: DATA is encrypted.***

`HASH` is the 32 bytes digest of the next frame.

### 15.5.7.3 Last secured write command

A secured write command is the last command if and only if:

1. The opcode is 0x8C.
2. A first secured write command has already been received and successfully verified before.
3. No reset has occurred since the first secured write command was received.
4. The frame length is equal to the data size + header size: `FLEN = SIZE + 6`.
5. The digest of the whole frame is equal to the hash value received in the previous frame.

**Last** frame format exchange:

```
[HDLL] -> [FLEN 0x8C SIZE ADDR {DATA} CRC16]
[HDLL] <- [0x00 0x04 STAT 0x00 0x00 0x00 CRC16]
```

`STAT` is the return status.

`FLEN` is the size of frame (bit 0 to 9 of the 2 first bytes)

`SIZE` is the 16 bits size of data.

***Note: Data size max value is 512.***

`ADDR` is the 24 bits address of the data to write in flash.

***Note: ADDR is encrypted.***

`{DATA}` is the chunk of data.

***Note: DATA is encrypted.***

# 16 PN722X NFCC practical approach

## 16.1 Basic examples for Reader/Writer (R/W) mode

### 16.1.1 R/W mode from DH (1 NFC endpoint)

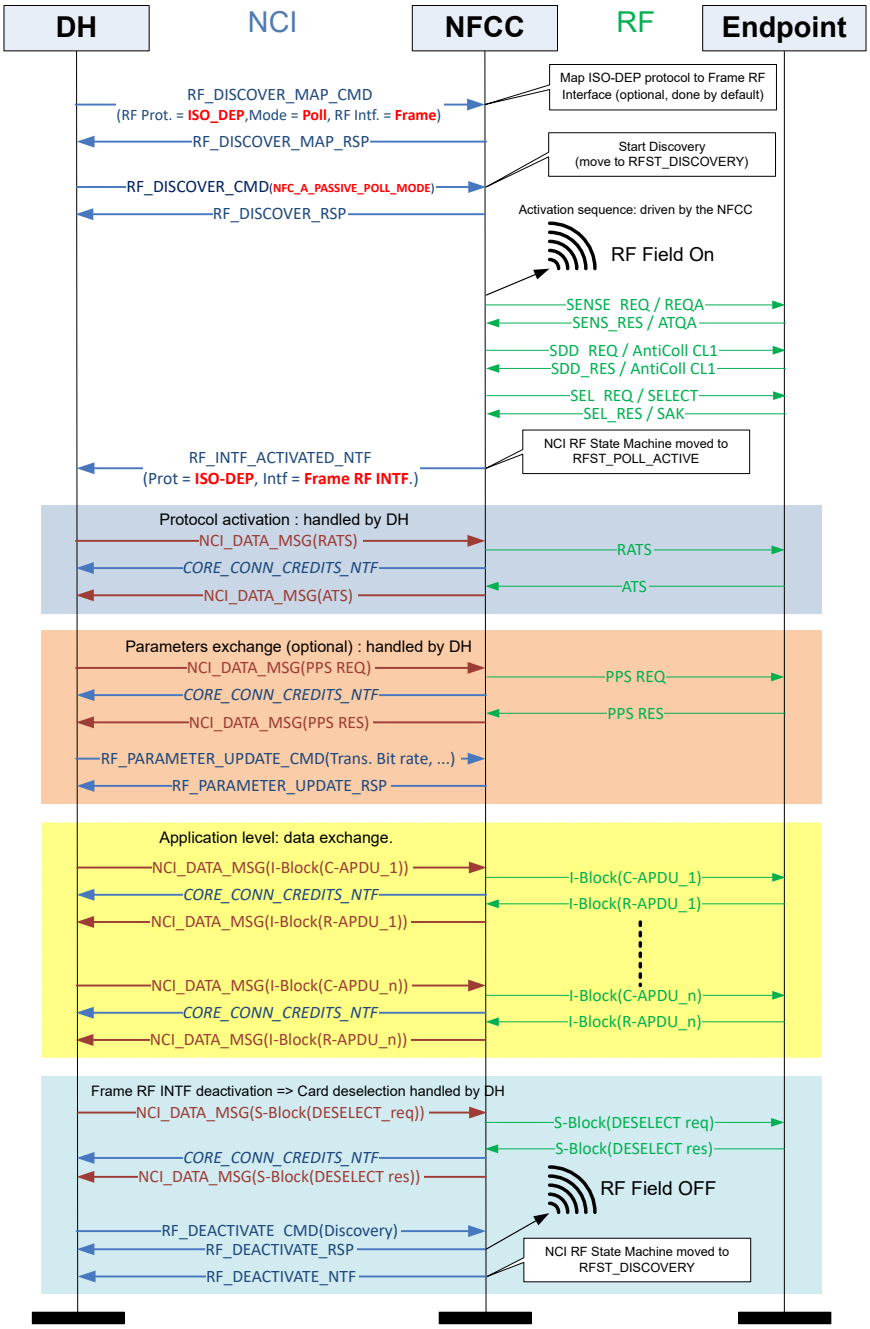



**Figure 49. Poll mode hosted by the DH, NFC-A, ISO-DEP protocol / RF frame interface**

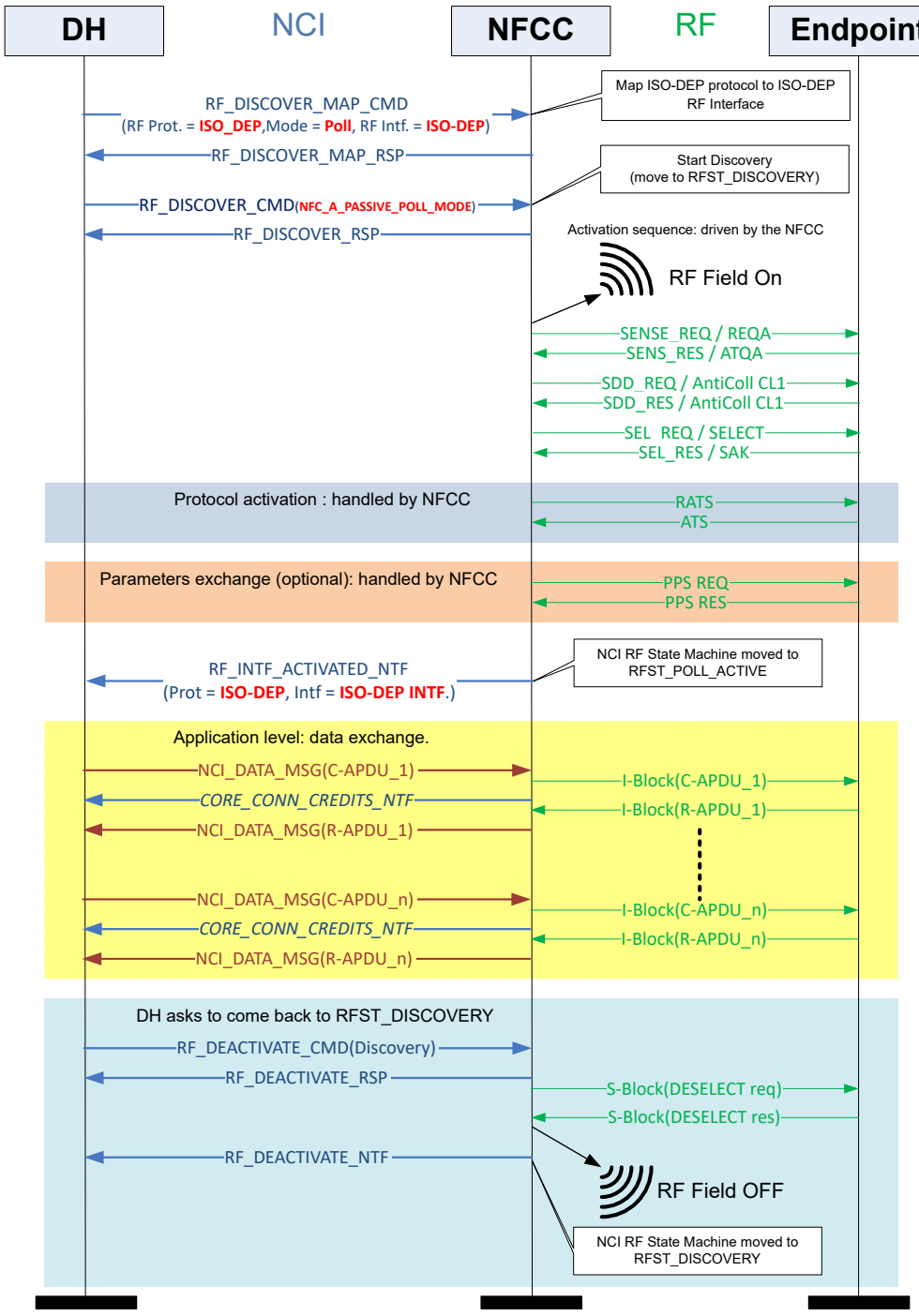



**Figure 50. Poll Mode hosted by the DH, NFC-A, ISO-DEP protocol / ISO-DEP interface**

### 16.1.2 R/W mode from DH (2 NFC endpoints)

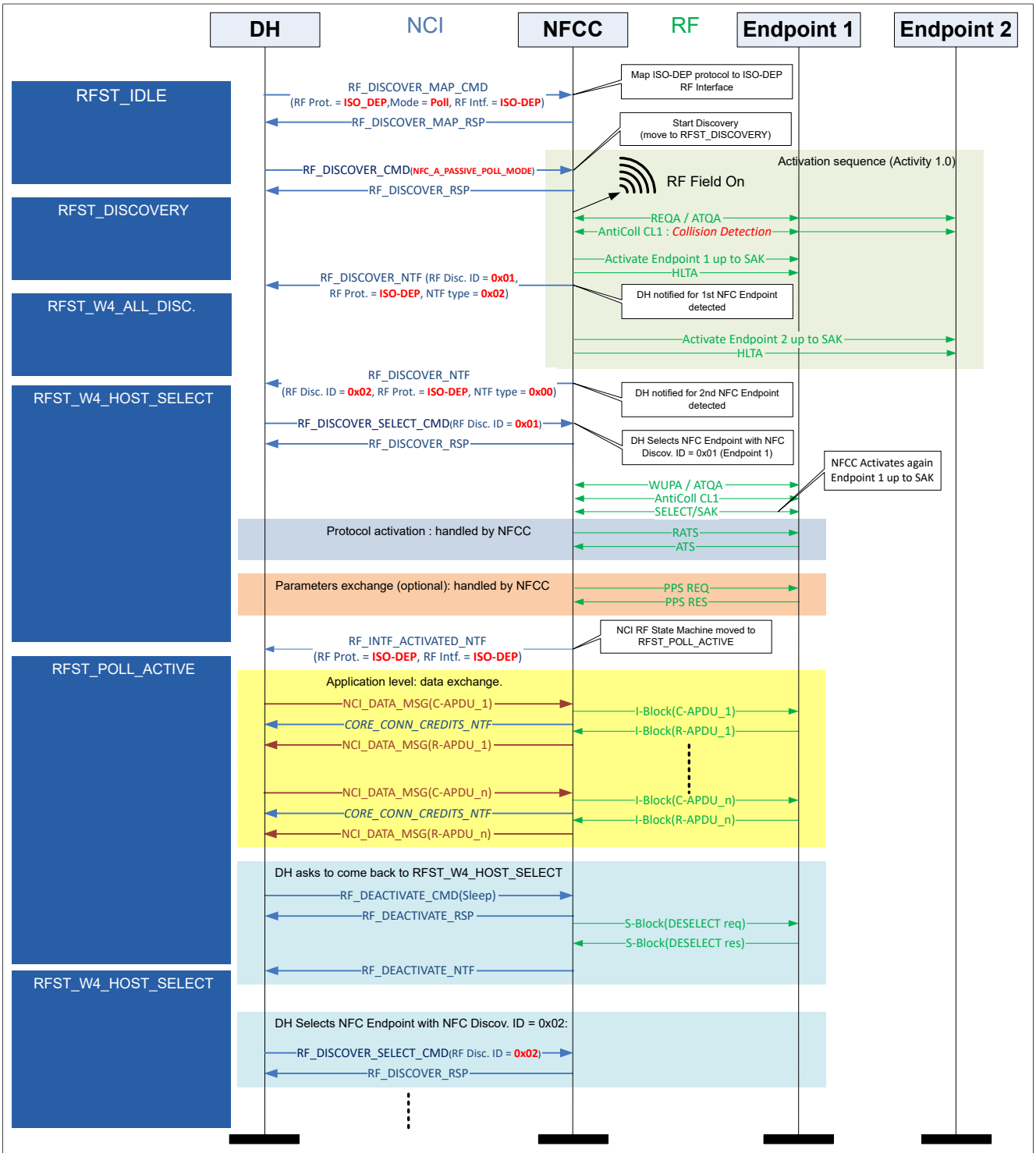



**Figure 51. Poll mode hosted by the DH: 2 NFC-A Cards, ISO-DEP protocol / ISO-DEP interface**

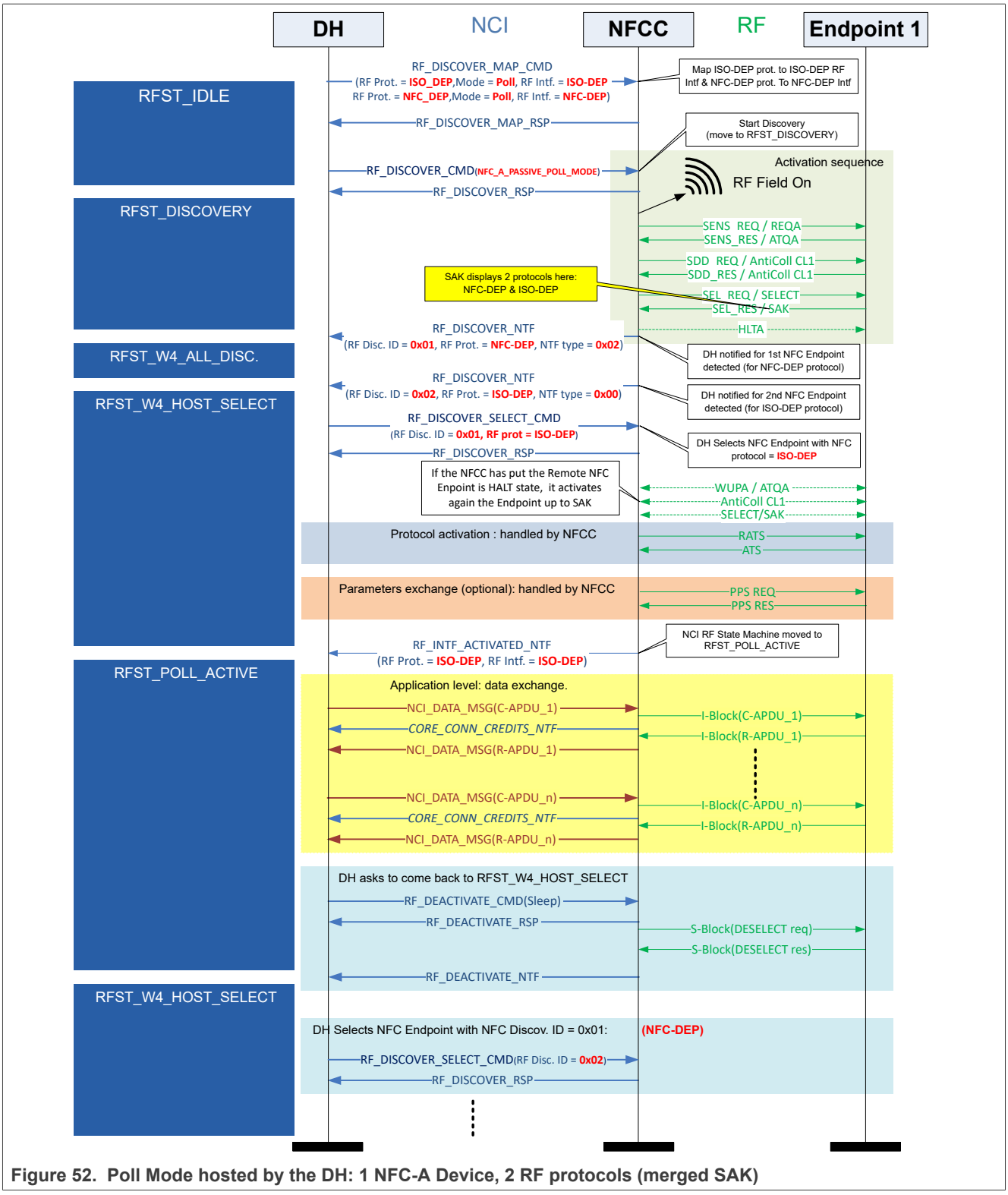



**Figure 52. Poll Mode hosted by the DH: 1 NFC-A Device, 2 RF protocols (merged SAK)**

### 16.1.3 R/W mode from DH (2 NFC endpoints with 2 different technologies)

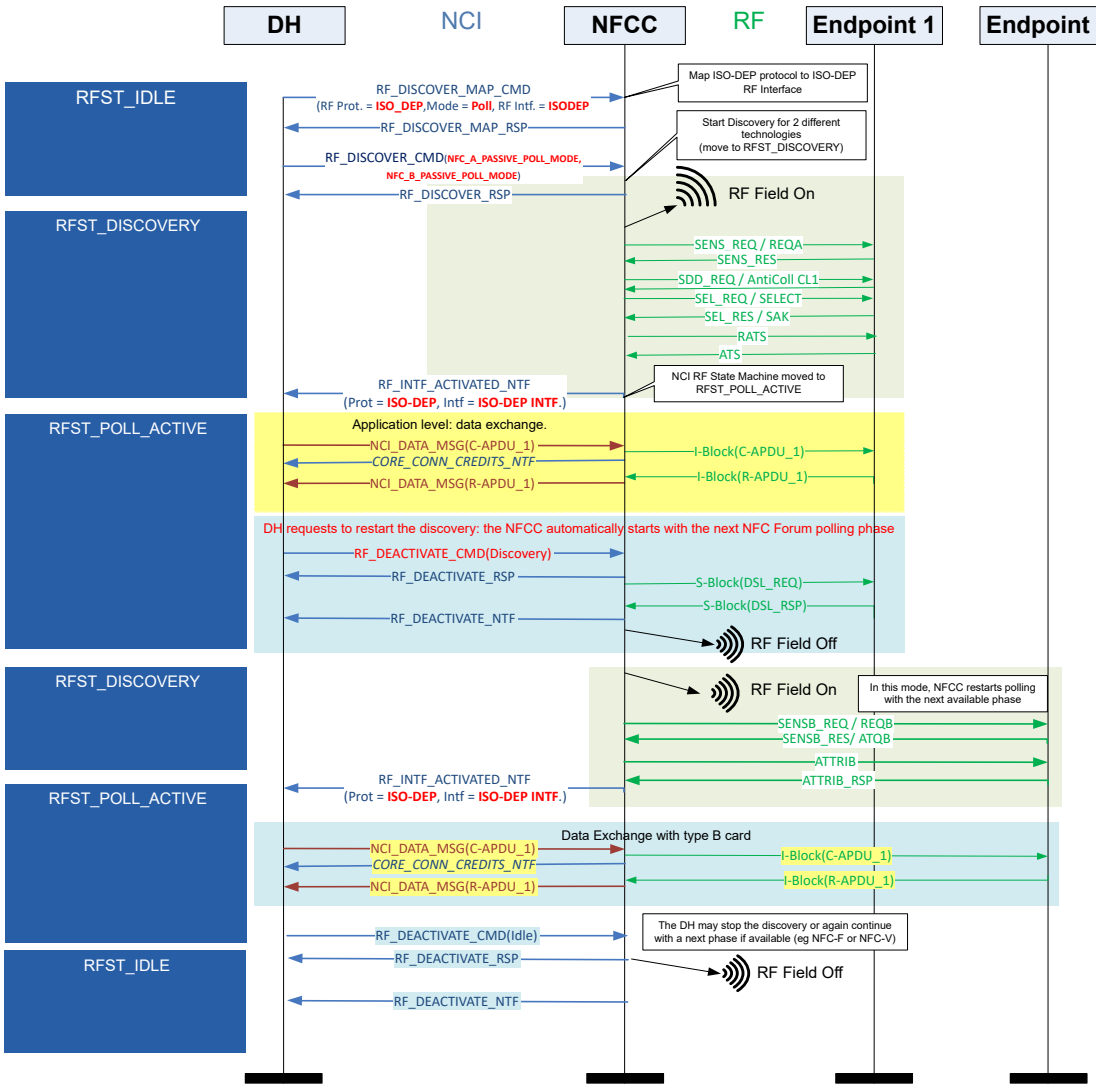



**Figure 53. Poll mode hosted by the DH: 1 NFC-A Card and 1 NFC-B card, ISO-DEP protocol / ISO-DEP interface**

## 16.2 Basic examples for Card Emulation (CE) mode

### 16.2.1 CE mode in DH

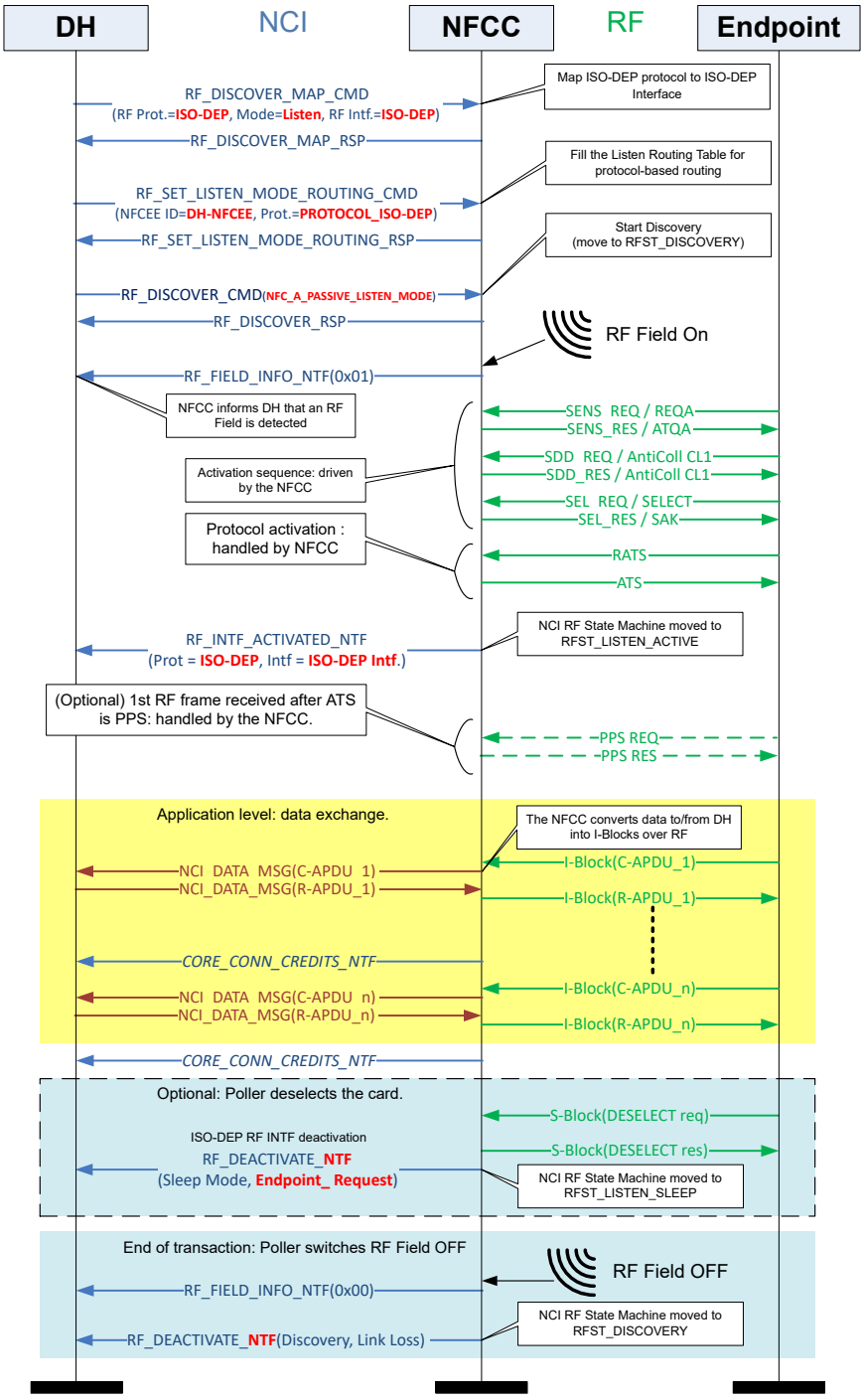



**Figure 54. Card emulation in DH, NFC-A, ISO-DEP protocol / ISO-DEP interface, optional PPS**

***Note:*** *This sequence is not according to NCI on the ISO-DEP RF interface activation versus the PPS reception. This sequence describes the NXP NFCC behavior, not NCI expectation: RF_INTF_ACTIVATED_NTF always indicates 106 kB/s.*

# 17 Annex A: Details on RF state machine

The [NCI] RF state machine is complex and the drawing proposed in the NCI technical specification combines all the modes of operation in a single drawing.

For debug purposes, it is convenient to draw a simplified state machine for each Protocol to RF interface mapping applied by the DH. The transitions not supported by NXP product or by the interface in use are reported in grey in Figure 55 and Figure 56.

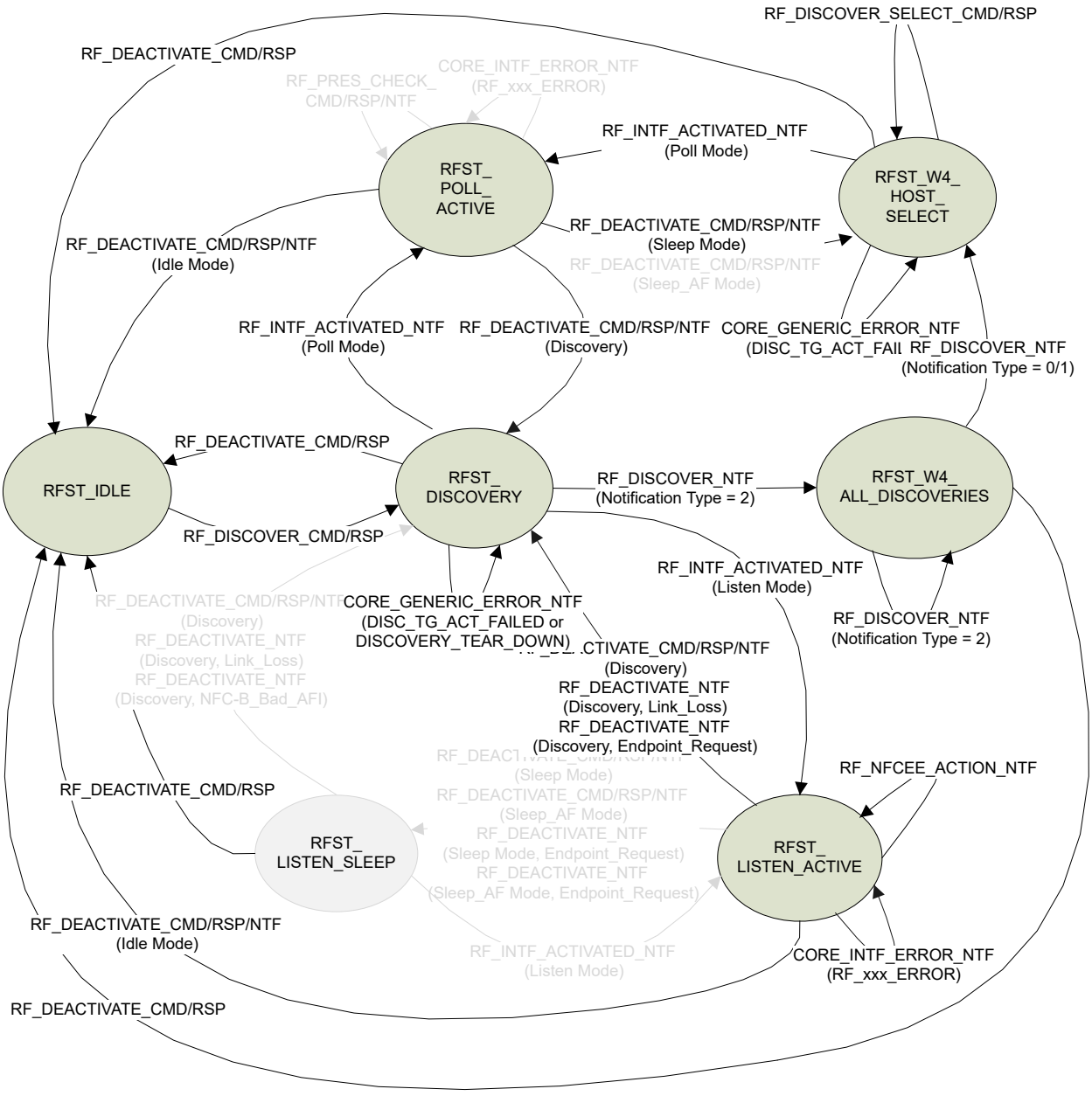
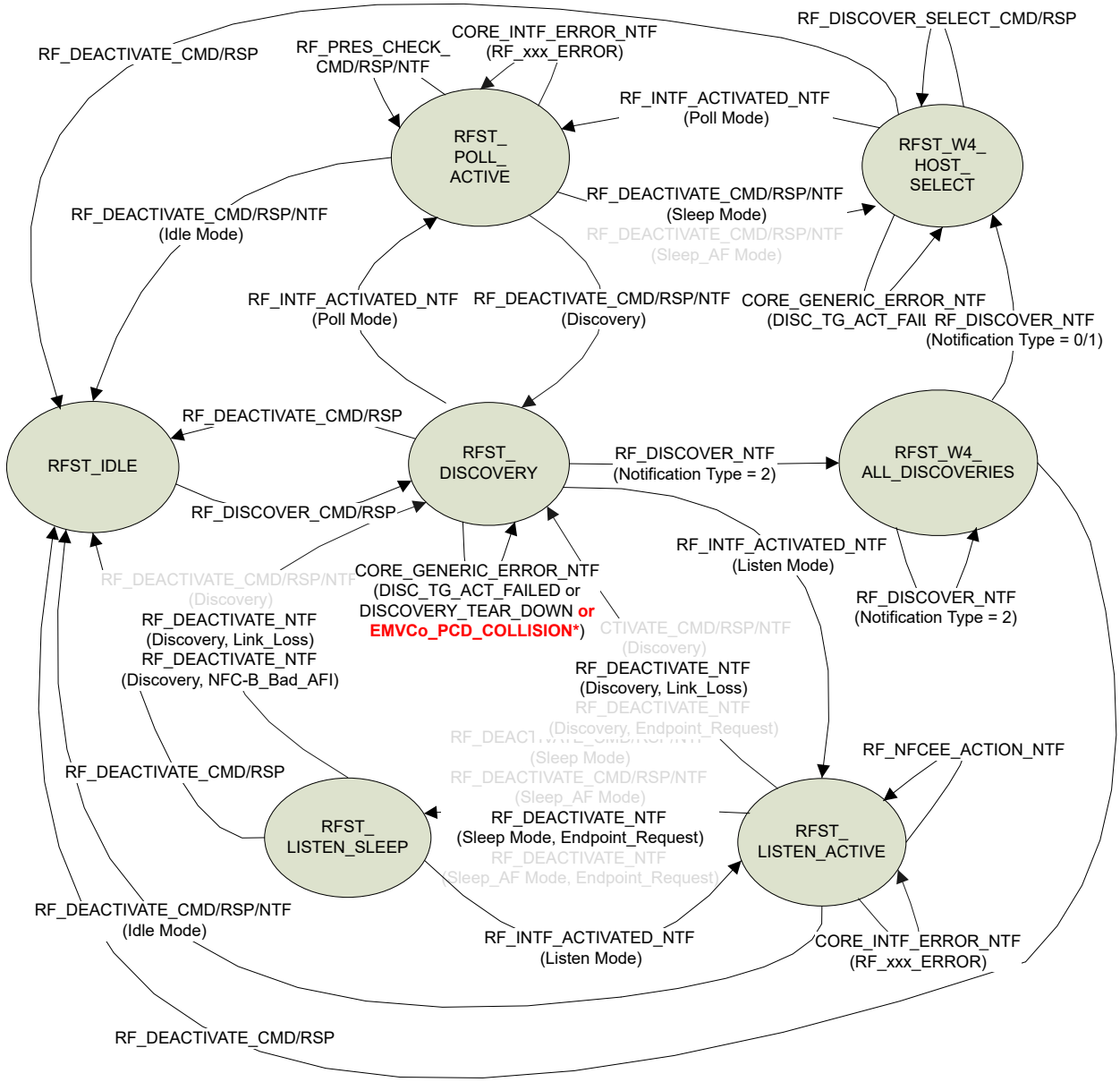



**Figure 55. RF state machine when the Frame RF interface is used**




(*) Proprietary notification

**Figure 56. RF state machine when the ISO-DEP RF interface is used**

# 18 Annex B: List of [PN722X NFCC-NCI] extensions to control messages

**Table 123. [PN722X NFCC-NCI] extensions to control messages**

| Section | GID | OID | [PN722X NFCC-NCI] control message |
|---|---|---|---|
| Section 12.4.1 | 1111b | 0x00 | CORE_SET_POWER_MODE_CMD<br>CORE_SET_POWER_MODE_RSP |
| Section 9.2.1 | 1111b | 0x02 | NCI_PROPRIETARY_ACT_CMD<br>NCI_PROPRIETARY_ACT_RSP |
| Section 14.6 | 1111b | 0x0D | PASSTHROUGH_RF_EXCHANGE_CMD<br>PASSTHROUGH_RF_EXCHANGE_RSP |
| Section 14.5 | 1111b | 0x0E | LOAD_RF_CONFIG_CMD<br>LOAD_RF_CONFIG_RSP |
| Section 10.1.3.3 | 1111b | 0x17 | WTX_NTF |
| — | 1111b | 0x1C | LOW_POWER_TAG_REMOVAL_CMD<br>LOW_POWER_TAG_REMOVAL_RSP<br>LOW_POWER_TAG_REMOVAL_NTF |
| — | 1111b | 0x1D | LPCD_FPC_NTF |
| Section 7.3.12 | 1111b | 0x21 | FLUSH_SRAM_AO_TO_FLASH_CMD<br>FLUSH_SRAM_AO_TO_FLASH_RSP |
| Section 14.4.1 | 1111b | 0x22 | CTS_CONFIG_CMD<br>CTS_CONFIG_RSP<br>CTS_CONTROL_CMD<br>CTS_CONTROL_RSP<br>CTS_LOG_NTF |
| Section 14.2 | 1111b | 0x30 | TEST_PRBS_CMD<br>TEST_PRBS_RSP |
| Section 14.3 | 1111b | 0x32 | REG_TEST_API_CMD<br>REG_TEST_API_RSP |
| — | 1111b | 0x39<br>0x3A<br>0x3B<br>0x3C | PROP_PROTOCOL_CMD<br>PROP_PROTOCOL_RSP |
| Section 14.7 | 1111b | 0x3F | SWITCH_RF_FIELD_CMD<br>SWITCH_RF_FIELD_RSP |

# 19 Annex C: RF protocol configuration indexes

For TX and RX configuration for different RF technologies and data rates supported by PN722X, refer to Table 124. The values are not present in the range mentioned below, should be considered as RFU.

The TX and RX config setting have to be used in LOAD_RF_CONFIG_CMD commands to load the RF configuration from EEPROM into internal CLIF registers. RF configuration refers to a unique combination of RF Technology, mode (target/initiator) and baud rate. RF configuration can be loaded separately for the CLIF receiver (RX configuration) and transmitter (TX configuration) path. The value 0xFF must be used if the corresponding configuration for a path shall not be changed.

**Table 124. RX and TX configuration for different RF technologies and data rates**

| | TX | | | | RX | | | |
|---|---|---|---|---|---|---|---|---|
| | **Setting** | **Protocol** | **Speed** | **Modulation** | **Setting** | **Protocol** | **Speed** | **Modulation** |
| PCD | 0x00 | ISO14443A | 106 | Miller | 0x80 | ISO14443A | 106 | Manch SubC |
| | 0x01 | ISO14443A | 212 | Miller | 0x81 | ISO14443A | 212 | BPSK |
| | 0x02 | ISO14443A | 424 | Miller | 0x82 | ISO14443A | 424 | BPSK |
| | 0x03 | ISO14443A | 848 | Miller | 0x83 | ISO14443A | 848 | BPSK |
| | 0x04 | ISO14443B | 106 | NRZ | 0x84 | ISO14443B | 106 | BPSK |
| | 0x05 | ISO14443B | 212 | NRZ | 0x85 | ISO14443B | 212 | BPSK |
| | 0x06 | ISO14443B | 424 | NRZ | 0x86 | ISO14443B | 424 | BPSK |
| | 0x07 | ISO14443B | 848 | NRZ | 0x87 | ISO14443B | 848 | BPSK |
| | 0x08 | FeliCa | 212 | — | 0x88 | FeliCa | 212 | — |
| | 0x09 | FeliCa | 424 | — | 0x89 | FeliCa | 424 | — |
| | 0x0a | ISO15693_ASK100 | 26 | 1out4/ASK100 | 0x8a | ISO15693 | 6P6 | Manch424 |
| | 0x0b | ISO15693_ASK10 | 26 | 1out4/ASK10 | 0x8b | ISO15693 | 26 | Manch424 |
| | 0x0c | — | — | — | 0x8c | ISO15693 | 53 | Manch424 |
| | 0x0d | — | — | — | 0x8d | ISO15693 | 106 | NTAG5 |
| | 0x0e | — | — | — | 0x8e | ISO15693 | 212 | NTAG5 |
| | 0x0f | EMVCo_Felica | 212 | — | 0x8f | EMVCo_Felica | 212 | — |
| | 0x10 | EMVCo_Felica | 424 | — | 0x90 | EMVCo_Felica | 424 | — |
| | 0x11 | NA | — | — | 0x91 | EMVCo_ISO14443A | 106 | — |
| | 0x12 | NA | — | — | 0x92 | EMVCo_ISO14443B | 106 | — |
| PICC | 0x13 | ISO14443A-PICC | 106 | Manch SubC | 0x93 | ISO14443A-PICC | 106 | Miller |
| | 0x14 | ISO14443A-PICC | 212 | BPSK | 0x94 | ISO14443A-PICC | 212 | Miller |
| | 0x15 | ISO14443A-PICC | 424 | BPSK | 0x95 | ISO14443A-PICC | 424 | Miller |
| | 0x16 | ISO14443A-PICC | 848 | BPSK | 0x96 | ISO14443A-PICC | 848 | Miller |
| PCD | 0x19 | EMVCo_ISO14443A | 106 | — | 0x99 | — | — | — |
| | 0x1a | EMVCo_ISO14443B | 106 | — | 0x9a | — | — | — |
| PICC | 0x1c | GTM | All | All | 0x9c | GTM | All | All |

# 20 Annex D: List of CTS and test bus signals

Figure 57 specifies the PN722X signals used to capture CTS instructions (Table 85) and test bus instructions.

| Serial Number | TB_Signal_Index | Name | TB_Bit_Index | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | Bit7 | Bit6 | Bit5 | Bit4 | Bit3 | Bit2 | Bit1 | Bit0 |
| 17 | 0x6c | obs_clif_sigpro_rm4 | signal_active | mf_pt_s0_d[14:8] | | | | | | |
| 18 | 0x6d | obs_clif_sigpro_rm5 | mf_pt_s0_d[7:0] | | | | | | | |
| 19 | 0x6e | obs_clif_sigpro_rm6 | rm_dsp_enable_o | mf_pt_s1_d[14:8] | | | | | | |
| 20 | 0x6f | obs_clif_sigpro_rm7 | mf_pt_s1_d[7:0] | | | | | | | |
| 21 | 0x70 | obs_clif_sigpro_rm8 | rm_sdata_o | sync_filt_out[14:8] | | | | | | |
| 22 | 0x71 | obs_clif_sigpro_rm9 | sync_filt_out[7:0] | | | | | | | |
| 29 | 0x78 | obs_clif_tbcontrol_patchbox0 | adc_data_i_i[9:2] | | | | | | | |
| 30 | 0x79 | obs_clif_tbcontrol_patchbox1 | adc_data_i_i[9] | adc_data_i_i[6:0] | | | | | | |
| 31 | 0x7a | obs_clif_tbcontrol_patchbox2 | adc_data_q_i[9:2] | | | | | | | |
| 32 | 0x7b | obs_clif_tbcontrol_patchbox3 | adc_data_q_i[9] | adc_data_q_i[6:0] | | | | | | |
| 35 | 0x86 | obs_clif_tbcontrol_patchbox14 | | | | rx_cl_error_i | | | | |
| 41 | 0x9b | obs_clif_txenc1 | bit_gen_enable | symbol_enable | tx_enable_o | tx_active | tx_state[2:0] | | | tx_rate_enable |
| 44 | 0x42 | obs_clif_rxdec0 | mpp_imsg[3:0] | | | | mpp_idata[1:0] | | mstate[1:0] | |
| 45 | 0x45 | obs_clif_rxdec3 | fp_flg_par_ignored | fp_ieof | fp_flg_par_irrelevant | fp_flg_collon_detected | fp_byte_dmark | fp_state | fp_phase[1:0] | |
| 47 | 0x68 | obs_clif_sigpro_rm0 | rm_dsp_enable_o | rm_cor_adc_i_o[14:8] | | | | | | |
| 48 | 0x69 | obs_clif_sigpro_rm1 | rm_cor_adc_i_o[7:0] | | | | | | | |
| 49 | 0x6a | obs_clif_sigpro_rm2 | rm_dsp_enable_o | rm_cor_adc_q_o[14:8] | | | | | | |
| 50 | 0x6b | obs_clif_sigpro_rm3 | rm_cor_adc_q_o[7:0] | | | | | | | |

**Figure 57. PN722X signals available for CTS and test bus instructions**

# 21 Annex E: EEPROM configuration description

EEPROM settings are used for configurations which do not change often. EEPROM configuration is performed once during the trimming or configuration of a product. The EEPROM has a limited number of erase/write cycles.

Use the standard registers which do not keep their value during reset and power off for configurations that change often.

EEPROM configuration of the PN722X is described in PN722X data sheet. Write to the EEPROM with Read-Modify-Write for all the memory addresses which contain RFU bits.

## 22 Note about the source code in the document

Example code shown in this document has the following copyright and BSD-3-Clause license:

Copyright 2024 NXP Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials must be provided with the distribution.
3. Neither the name of the copyright holder nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

# 23 Revision history

**Table 125. Revision history**

| Document ID | Release date | Description |
|---|---|---|
| UM11810 v.1.0 | 2 February 2024 | • Initial version |

# Legal information

## 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.

## 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 https://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.

**Suitability for use in non-automotive qualified products** — Unless this document expressly states that this specific NXP Semiconductors product is automotive qualified, the product is not suitable for automotive use. It is neither qualified nor tested in accordance with automotive testing or application requirements. NXP Semiconductors accepts no liability for inclusion and/or use of non-automotive qualified products in automotive equipment or applications.

In the event that customer uses the product for design-in and use in automotive applications to automotive specifications and standards, customer (a) shall use the product without NXP Semiconductors' warranty of the product for such automotive applications, use and specifications, and (b) whenever customer uses the product for automotive applications beyond NXP Semiconductors' specifications such use shall be solely at customer's own risk, and (c) customer fully indemnifies NXP Semiconductors for any liability, damages or failed product claims resulting from customer design and use of the product for automotive applications beyond NXP Semiconductors' standard warranty and NXP Semiconductors' product specifications.

**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.

## 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.

## 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.

**FeliCa** — is a trademark of Sony Corporation.

**I2C-bus** — logo is a trademark of NXP B.V.

**JCOP** — is a trademark of NXP B.V.

**MIFARE** — is a trademark of NXP B.V.

**MIFARE Classic** — is a trademark of NXP B.V.

**MIFARE Plus** — is a trademark of NXP B.V.

**MIFARE Ultralight** — is a trademark of NXP B.V.

# Tables

## Figures

# Contents

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

**Date of release: 2 February 2024**
**Document identifier: UM11810**