AN13189

PN7160 Android porting guide

Rev. 1.8 — 15 March 2022 Application note

Document information

Information	Content
Keywords	Android, NFC, NXP, NCI, PN7160
Abstract	This application note describes how to add support for PN7160 NXP NCI-based NFC controller to an Android system.

1 Revision history

Revision history

Rev	Date	Description
1.8	20230315	Section 3 "Security fixes": added
1.7	20221215	Section 5.1 "Android 13": added
1.6	20220224	Adding firmware update procedure details (Section 5.2.5 and Section 5.3.5)
1.5	20220201	Adding Android12 support
1.4	20210916	Typo error in repository address fixed
1.3	20210913	Security status changed into "Company public", no content change
1.2	20210820	Security status changed into "Company restricted"
1.1	20210709	Updated with SPI support, FW update, NDEF emulation support and DTA application
1.0	20210324	Initial release

2 Introduction

This document provides guidelines for the integration of PN7160 NXP NCI-based NFC controller to an Android platform from software perspective.

It first explains how to install the required kernel driver, then it describes step by step how to adapt the Android Open Source Project sources for adding the support of PN7160 NFC controller. Figure 1 shows the architecture of the whole Android NFC stack.

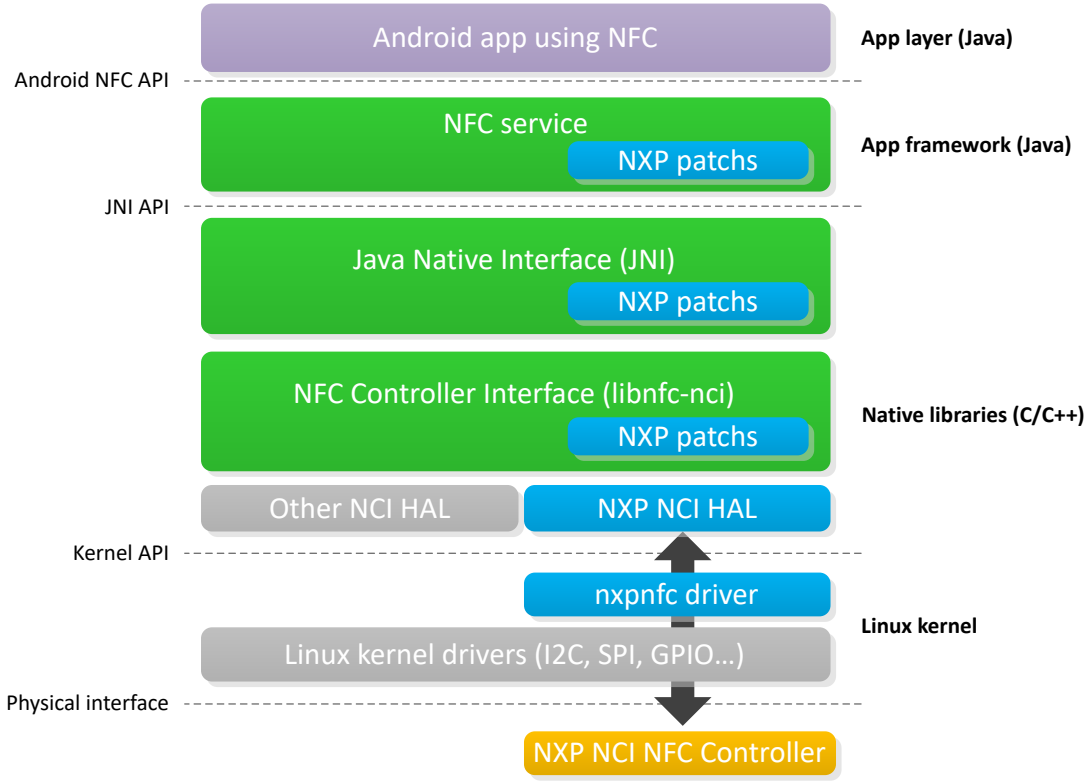

Figure 1. Android NFC stack overview

- The nxpnfc driver is the kernel module allowing to access NXP NCI-based NFC controller hardware resource
- The NXP NCI HAL module is the implementation of NXP NFC controller's specific hardware abstraction layer
- The libnfc-nci is the native library providing NFC functionality
- The JNI is a glue code between Java and Native classes
- The NFC service is the application framework module providing access to NFC functionality

3 Security fixes

Important notice:

The customer must integrate following security fixes. NXP will support them from next Android release.

CVE-2022-20471: https://android.googlesource.com/platform/hardware/nxp/nfc/+/1164ee536ecf6504e73dcaac0ccb1ee887f3a19c

CVE-2023-20945: https://android.googlesource.com/platform/packages/apps/Nfc/+/4a964908ff0bd91d93f96cdc26f7377420c58273

4 Kernel driver

The NFC Android stack uses nxpnfc kernel driver to communicate with the NXP NCI NFC controller. It is available from the following repository: https://github.com/NXPNFCLinux/nxpnfc.

4.1 Driver details

The nxpnfc kernel driver offers communication to the NFC controller connected over either I^2C or SPI physical interface.

When loaded to the kernel, this driver exposes the interface to the NFC controller through the device node named */dev/nxpnfc*.

This kernel driver is compatible with a broad range of NXP's NFC controllers, it explains specific NXP references can be found in the source code.

The provided source code allows building both versions of the kernel driver (I^2C and SPI) according to the kernel configuration.

4.2 Getting the source code

Clone the nxpnfc repository into the kernel directory, replacing existing implementation:

```
$ rm -rf drivers/nfc
$ git clone https://github.com/NXPNFCLinux/nxpnfc.git drivers/nfc
```

This will end-up with the folder *drivers/nfc* containing the following files:

- *README.md*: repository information
- *Makefile*: driver heading makefile
- *Kconfig*: driver configuration file
- *LICENSE*: driver licensing terms
- *i2c_devicetree.txt*: example of I^2C device tree definition
- *spi_devicetree.txt*: example of SPI device tree definition
- *nfc* sub folder containing:
  - *Makefile*:
  - *common.c*: generic driver implementation
  - *common.h*: generic driver interface definition
  - *i2c.c*: I^2C-specific driver implementation
  - *i2c.h*: I^2C-specific driver interface definition
  - *spi.c*: SPI-specific driver implementation
  - *spi.h*: SPI-specific driver interface definition

4.3 Including the driver into the kernel

Including the driver to the kernel, and making it loaded during device boot, is done thanks to the device tree.

After updating the device tree definition as suggested in below examples, the platform-related device tree must be rebuilt.

4.3.1 I^2C version

I^2C address (0x28 in below examples) and GPIO assignments must be adapted according to the hardware integration in the platform.

Below is an example of definition to be added to the platform device tree file (*.dts* file located for instance under *arch/arm/boot/dts* kernel subfolder for arm-based platform).

```
i2c0: i2c@ffd71000 {
    ...
    status = "ok";
    nxpnfc: nxpnfc@28 {
        compatible = "nxp,nxpnfc";
        reg = <0x28>;
        nxp,nxpnfc-irq = <&gpio26 0 0>;
        nxp,nxpnfc-ven = <&gpio26 2 0>;
        nxp,nxpnfc-fw-dwnld = <&gpio26 4 0>;
    };
};
```

4.3.2 SPI version

SPI handle (0 in the below example) and GPIO assignments must be adapted according to the hardware integration in the platform.

Below is an example of definition to be added to the platform device tree file (*.dts* file located for instance under *arch/arm/boot/dts* kernel subfolder for arm-based platform).

```
spi2: spi@ffd68000 {
    ...
    status = "ok";
    nxpnfc@0 {
        compatible = "nxp,nxpnfc";
        reg = <0>;
        nxp,nxpnfc-irq = <&gpio26 0 0>;
        nxp,nxpnfc-ven = <&gpio26 2 0>;
        nxp,nxpnfc-fw-dwnld = <&gpio26 4 0>;
        spi-max-frequency = <7000000>;
    };
};
```

4.4 Building the driver

Through *menuconfig* procedure include the targeted driver (I^2C or SPI version) to the build, as built in (<*>):

```
Device Drivers --->
    < > NFC I2C Slave driver for NXP-NFCC
    < > NFC SPI Slave driver for NXP-NFCC
```

Rebuilding the complete kernel, the driver will be included in the kernel image.

5 AOSP adaptation

5.1 Android 13

Below step-by-step procedure is based on NXP's Android NFC delivery from https://github.com/NXPNFCLinux/nxpnfc_android13 repository.

The current release is based on Android AOSP version 13.0.0_r3, porting to other Android 13 subversion may require minor adaptation of API (detected when compiling).

5.1.1 Step 1: retrieving NXP's Android NFC delivery

Copy the content of the PN7160 Android 13 engineering release package to the AOSP repository:

```
$ git clone https://github.com/NXPNFCLinux/nxpnfc_android13.git ${ANDROID_BUILD_TOP}/vendor/nxp/nfc
```

Please be aware that Android 13 is using kernel version 5.10.

5.1.2 Step 2: installing NXP-NCI delivery

Run the installation script:

```
$ ${ANDROID_BUILD_TOP}/vendor/nxp/nfc/install_NFC.sh
```

This will:

- Patch the AOSP *system/nfc* implementation to add PN7160 specific support
- Patch the AOSP *hardware/nxp/nfc* implementation to add PN7160 specific support
- Patch the AOSP *packages/apps/Nfc* folder to add support for PN7160 extensions
- Patch the AOSP *frameworks/base* definitions to add specific API
- Patch the AOSP *frameworks/native* definitions to add specific permissions
- Patch the *vendor/nxp* folder to add specific API for T4T NDEF emulation
- Patch the AOSP *build/make* folder to avoid warnings when building image
- Patch the AOSP *hardware/interface* definitions to add specific interface
- Patch the AOSP *system/core* folder to force remounting partitions

5.1.3 Step 3: updating configuration files

Adapt the *libnfc-nci.conf and libnfc-nxp.conf* files located in *vendor/nxp/nfc/hw/pn7160/conf* subfolder, created at Section 5.1.1, according to the integration specificities (e.g. clock configuration, TxLDO configuration, RF settings …).

For instance if using a system clock instead of an onboard crystal, the value of parameter "NXP_SYS_CLK_SRC_SEL" in *libnfc-nxp.conf* must reflect this configuration.

More details about the configuration files can be found in Section 6.

5.1.4 Step 4: adding NFC to the build

In the *device.mk* makefile (e.g. *device/**brand**/**platform**/device.mk*), include specific makefile.

```
$(call inherit-product, vendor/nxp/nfc/device-nfc.mk)
```

In the *BoardConfig.mk* makefile (e.g. *device/**brand/platform**/BoardConfig.mk*), include specific makefile.

```
-include vendor/nxp/nfc/BoardConfigNfc.mk
```

5.1.5 Step 5: adding firmware libraries

The Android NFC stack integrates the support for updating the NFC controller firmware. To allow the update mechanism, the updated firmware version must be included on the target in a form of a library (or binary). Arm architecture libraries for NXP's NFC controller firmware are provided via dedicated repository https://github.com/NXP/nfc-NXPNFCC_FW.

Retrieve PN7160 firmware library files from repository to dedicated subfolder with following commands:

```
$ wget -r -np -nd -P ${ANDROID_BUILD_TOP}/vendor/nxp/pn7160/firmware/64-bit/ https://github.com/NXP/nfc-NXPNFCC_FW/
tree/master/InfraFW/pn7160/64-bit/libpn7160_fw.so
$ wget -r -np -nd -P ${ANDROID_BUILD_TOP}/vendor/nxp/pn7160/firmware/32-bit/ https://github.com/NXP/nfc-NXPNFCC_FW/
tree/master/InfraFW/pn7160/32-bit/libpn7160_fw.so
```

This creates *vendor/nxp/pn7160/firmware* subfolder containing 32 bits and 64 bits Arm architecture libraries. Those libraries will be included in the image when building the android image (as defined within *vendor/nxp/nfc/device-nfc.mk*).

5.1.6 Step 6: building and installing NFC

Build and flash the Android images to the target (the boot image must contain the kernel driver as instructed in Section 4). Please be aware that kernel should be at version 5.10.

5.1.7 Step 7: verifying NFC functionality

In Android “Settings” menu, check that NFC is ON. NFC functionality should be then up and running, ready to discover NFC tags or exchange data with remote NFC devices.

To further test NFC reader functionality, ***NFC TagInfo by NXP*** and ***NFC TagWriter by NXP*** are 2 applications available for free from Google Play store.

5.2 Android 12

Below step-by-step procedure is based on NXP’s Android NFC delivery from https://github.com/NXPNFCLinux/nxpnfc_android12 repository.

The current release is based on Android AOSP version 12.0.0_r9, porting to other Android 12 subversion may require minor adaptation of API (detected when compiling).

5.2.1 Step 1: retrieving NXP's Android NFC delivery

Copy the content of the PN7160 Android 12 engineering release package to the AOSP repository:

```
$ git clone https://github.com/NXPNFCLinux/nxpnfc_android12.git ${ANDROID_BUILD_TOP}/vendor/nxp/nfc
```

5.2.2 Step 2: installing NXP-NCI delivery

Run the installation script:

```
$ ${ANDROID_BUILD_TOP}/vendor/nxp/nfc/install_NFC.sh
```

This will:

- Patch the AOSP *system/nfc* implementation to add PN7160 specific support
- Patch the AOSP *hardware/nxp/nfc* implementation to add PN7160 specific support
- Patch the AOSP *packages/apps/Nfc* folder to add support for PN7160 extensions
- Patch the AOSP *frameworks/base* definitions to add specific API
- Patch the AOSP *frameworks/native* definitions to add specific permissions
- Patch the *vendor/nxp* folder to add specific API for T4T NDEF emulation
- Patch the AOSP *build/make* folder to avoid warnings when building image
- Patch the AOSP *hardware/interface* definitions to add specific interface
- Patch the AOSP *system/core* folder to force remounting partitions

5.2.3 Step 3: updating configuration files

Adapt the *libnfc-nci.conf and libnfc-nxp.conf* files located in *vendor/nxp/nfc/hw/pn7160/conf* subfolder, created at Section 5.2.1, according to the integration specificities (e.g. clock configuration, TxLDO configuration, RF settings …).

For instance if using a system clock instead of an onboard crystal, the value of parameter “NXP_SYS_CLK_SRC_SEL” in *libnfc-nxp.conf* must reflect this configuration.

More details about the configuration files can be found in Section 6.

5.2.4 Step 4: adding NFC to the build

In the *device.mk* makefile (e.g. *device/**brand/platform**/device.mk*), include specific makefile

```
$(call inherit-product, vendor/nxp/nfc/device-nfc.mk)
```

In the *BoardConfig.mk* makefile (e.g. *device/**brand/platform**/BoardConfig.mk*), include specific makefile

```
-include vendor/nxp/nfc/BoardConfigNfc.mk
```

5.2.5 Step 5: adding firmware libraries

The Android NFC stack integrates the support for updating the NFC controller firmware. To allow the update mechanism, the updated firmware version must be included on the target in a form of a library (or binary). Arm architecture libraries for NXP's NFC controller firmware are provided via dedicated repository https://github.com/NXP/nfc-NXPNFCC_FW.

Retrieve PN7160 firmware library files from repository to dedicated subfolder with following commands:

```
$ wget -r -np -nd -P ${ANDROID_BUILD_TOP}/vendor/nxp/pn7160/firmware/64-bit/ https://github.com/NXP/nfc-NXPNFCC_FW/
tree/master/InfraFW/pn7160/64-bit/libpn7160_fw.so
$ wget -r -np -nd -P ${ANDROID_BUILD_TOP}/vendor/nxp/pn7160/firmware/32-bit/ https://github.com/NXP/nfc-NXPNFCC_FW/
tree/master/InfraFW/pn7160/32-bit/libpn7160_fw.so
```

This creates *vendor/nxp/pn7160/firmware* subfolder containing 32 bits and 64 bits Arm architecture libraries. Those libraries will be included in the image when building the android image (as defined within *vendor/nxp/nfc/device-nfc.mk*).

5.2.6 Step 6: building and installing NFC

Build and flash the Android images to the target (the boot image must contain the kernel driver as instructed in Section 4).

5.2.7 Step 7: verifying NFC functionality

In Android “Settings” menu, check that NFC is ON. NFC functionality should be then up and running, ready to discover NFC tags or exchange data with remote NFC devices.

To further test NFC reader functionality, ***NFC TagInfo by NXP*** and ***NFC TagWriter by NXP*** are 2 applications available for free from Google Play store.

5.3 Android 11

Below step-by-step procedure is based on NXP’s Android NFC delivery from https://github.com/NXPNFCLinux/nxpnfc_android11 repository.

The current release is based on Android AOSP version 11.0.0_r3, porting to other Android 11 subversion may require minor adaptation of API (detected when compiling).

5.3.1 Step 1: retrieving NXP's Android NFC delivery

Clone repository into AOSP source directory:

```
$ git clone https://github.com/NXPNFCLinux/nxpnfc_android11.git ${ANDROID_BUILD_TOP}/vendor/nxp/nfc
```

5.3.2 Step 2: installing NXP-NCI delivery

Run the installation script:

```
$ ${ANDROID_BUILD_TOP}/vendor/nxp/nfc/install_NFC.sh
```

This will:

- Patch the AOSP *system/nfc* implementation to add PN7160 specific support
- Patch the AOSP *hardware/nxp/nfc* implementation to add PN7160 specific support
- Patch the AOSP *packages/apps/Nfc* folder to add support for PN7160 extensions
- Patch the AOSP *frameworks/base* definitions to add specific API
- Patch the AOSP *frameworks/native* definitions to add specific permissions
- Patch the *vendor/nxp* folder to add specific API for T4T NDEF emulation

5.3.3 Step 3: updating configuration files

Adapt the *libnfc-nci.conf and libnfc-nxp.conf* files located in *vendor/nxp/nfc/hw/pn7160/conf* subfolder, created at Section 5.3.1, according to the integration specificities (e.g. clock configuration, TxLDO configuration, RF settings …).

For instance if using a system clock instead of an onboard crystal, the value of parameter “NXP_SYS_CLK_SRC_SEL” in *libnfc-nxp.conf* must reflect this configuration.

More details about the configuration files can be found in Section 6.

5.3.4 Step 4: adding NFC to the build

In the *device.mk* makefile (e.g. *device/**brand**/**platform**/device.mk*), include specific makefile

```
$(call inherit-product, vendor/nxp/nfc/device-nfc.mk)
```

In the *BoardConfig.mk* makefile (e.g. *device/**brand**/**platform**/BoardConfig.mk*), include specific makefile

```
-include vendor/nxp/nfc/BoardConfigNfc.mk
```

5.3.5 Step 5: adding firmware libraries

The Android NFC stack integrates the support for updating the NFC controller firmware. To allow the update mechanism, the updated firmware version must be included on the target in a form of a library (or binary). Arm architecture libraries for NXP's NFC controller firmware are provided via dedicated repository https://github.com/NXP/nfc-NXPNFCC_FW.

Retrieve PN7160 firmware library files from repository to dedicated subfolder with following commands:

```
$ wget -r -np -nd -P ${ANDROID_BUILD_TOP}/vendor/nxp/pn7160/firmware/64-bit/ https://github.com/NXP/nfc-NXPNFCC_FW/
tree/master/InfraFW/pn7160/64-bit/libpn7160_fw.so
$ wget -r -np -nd -P ${ANDROID_BUILD_TOP}/vendor/nxp/pn7160/firmware/32-bit/ https://github.com/NXP/nfc-NXPNFCC_FW/
tree/master/InfraFW/pn7160/32-bit/libpn7160_fw.so
```

This creates *vendor/nxp/pn7160/firmware* subfolder containing 32 bits and 64 bits Arm architecture libraries. Those libraries will be included in the image when building the android image (as defined within *vendor/nxp/nfc/device-nfc.mk*).

5.3.6 Step 6: building and installing NFC

Build and flash the Android images to the target (the boot image must contain the kernel driver as instructed in Section 4).

5.3.7 Step 7: verifying NFC functionality

In Android “Settings” menu, check that NFC is ON. NFC functionality should be then up and running, ready to discover NFC tags or exchange data with remote NFC devices.

To further test NFC reader functionality, ***NFC TagInfo by NXP*** and ***NFC TagWriter by NXP*** are 2 applications available for free from Google Play store.

6 Configuration files

Two files allow configuring the libnfc-nci library at runtime: *libnfc-nci.conf* and *libnfc-nxp.conf*. There are defining tags which are impacting library behavior. The value of the tags depends on the NFC Controller IC and the targeted platform. For more details, refer to the examples given in *vendor/nxp/nfc/hw/pn7160* subfolder of the NXP's Android NFC delivery (see Section 5.2.3 or Section 5.3.3).

These files are loaded by the library respectively from */system/etc* and */vendor/etc* directories of the target, during the NFC initialization phase.

Pay attention that the configuration files provided as example relate to the NFC Controller demo board. These files must be adapted according to the targeted integration.

Below is the description of the different useful tags in the configuration files (refer to the conf files for detailed information about the tag values).

Table 1. Tag list of libnfc-nci.conf file

Tag	Description
APPL_TRACE_LEVEL	Log levels for libnfc-nci Recommended value for debugging is 0xFF
PROTOCOL_TRACE_LEVEL	Log levels for libnfc-nci Recommended value for debugging is 0xFF
NFC_DEBUG_ENABLED	NFC debug enable setting Recommended value for debugging is 0x01
NFA_STORAGE	Set the target directory for NFC file storage
HOST_LISTEN_TECH_MASK	Configure HOST listen feature
SCREEN_OFF_POWER_STATE	Configuration of screen off power state
POLLING_TECH_MASK	Configuration of the polling technologies
P2P_LISTEN_TECH_MASK	Configuration of listen technologies for P2P
NFA_DM_DISC_DURATION_POLL	Configuration of the discovery loop TOTAL DURATION (in milliseconds)
NFA_MAX_EE_SUPPORTED	Set the maximum number of Execution Environments supported
OFFHOST_AID_ROUTE_PWR_STATE	Defines the AID routing in device Off state Recommended value is 0x3B for NDEF emulation

Table 2. Tag list of libnfc-nxp.conf file

Tag	Description
NXPLOG_EXTNS_LOGLEVEL	Set level of EXTNS logs Recommended value for debug is 0x03
NXPLOG_NCIHAL_LOGLEVEL	Set level of NCIHAL logs Recommended value for debug is 0x03
NXPLOG_NCIX_LOGLEVEL	Set level of NCIX logs Recommended value for debug is 0x03
NXPLOG_NCIR_LOGLEVEL	Set level of NCIR logs Recommended value for debug is 0x03

Table 2. Tag list of libnfc-nxp.conf file*...continued*

Tag	Description
NXPLOG_FWDNLD_LOGLEVEL	Set level of FWDNLD logs Recommended value for debug is 0x03
NXPLOG_TML_LOGLEVEL	Set level of FWDNLD logs Recommended value for debug is 0x03
NXP_NFC_DEV_NODE	Set the NFC device node name
MIFARE_READER_ENABLE	Set the support of the reader for MIFARE Classic
NXP_FW_TYPE	Set the type of file taken for the FW update procedure Recommended value is 0x01 (".so" file)
NXP_SYS_CLK_FREQ_SEL	Set the clock frequency in case of PLL clock source
NXP_SYS_CLOCK_TO_CFG	Set clock request acknowledgment time value in case of PLL clock source
NXP_AGC_DEBUG_ENABLE	Set dynamic RSSI debug information
NXP_ACT_PROP_EXTN	Set NXP's NFC controller proprietary features
NXP_CORE_STANDBY	Set the standby mode enabled or disabled
NFA_PROPRIETARY_CFG	Set Vendor proprietary configuration
NXP_EXT_TVDD_CFG	Set TVDD configuration mode
NXP_EXT_TVDD_CFG_x	Configure TVDD settings according to TVDD mode selected
NXP_NFC_PROFILE_EXTN	Set discovery profile
NXP_I2C_FRAGMENTATION_ENABLED	Configure I^2C fragmentation
NXP_RF_CONF_BLK_x	Set platform-specific RF configuration
NXP_CORE_CONF_EXTN	Configure proprietary parts of the NFC controller
NXP_CORE_CONF	Configure standardized parts of the NFC controller
NXP_CORE_MFCKEY_SETTING	Proprietary configuration for the key storage for MIFARE Classic
PRESENCE_CHECK_ALGORITHM	Set the algorithm used for T4T presence check procedure
ISO_DEP_MAX_TRANSCEIVE	Define maximum ISO-DEP extended APDU length
NXP_T4T_NFCEE_ENABLE	Define NDEF emulation configuration
DEFAULT_T4TNFCEE_AID_POWER_STATE	Defines the NDEF emulation device power state Recommended value is 0x3B

7 Factory test native application

To ease the characterization of the NFC integration in the Android device, the FactoryTestApp native application is offered providing the following functionalities:

- Continuous RF ON: puts the NFC controller into continuous unmodulated RF field
- Functional mode: puts the NFC controller in a mode where it continuously poll for tag detection
- PRBS mode: puts the NFC controller into continuous modulated RF field emission (Pseudo Random pattern)
- Standby mode: puts the NFC controller in low power consumption mode (for consumption measurement)
- Dump RF settings: lists the value of all NFC controller RF settings
- Set RF settings: allows updating the value of NFC controller RF settings

The source code is delivered together with the NXP's Android NFC delivery (see above Section 5.2.1 or Section 5.3.1).

The binary is generated while building the system image, but it can also be independently built using following command:

```
$ mmm vendor/nxp/nfc/FactoryTestApp
```

Then copy the binary file (generated under *out/target/product/**platform**/system/bin/NfcFactoryTestApp)* to the Android target, using adb tool:

```
$ adb push NfcFactoryTestApp /data
```

On the Android target, update the file rights to allow execution and, after making sure that the NFC service is disabled (in “Settings” application NFC must be off, or else you can disable it using command *"adb shell svc nfc disable"*), run the application:

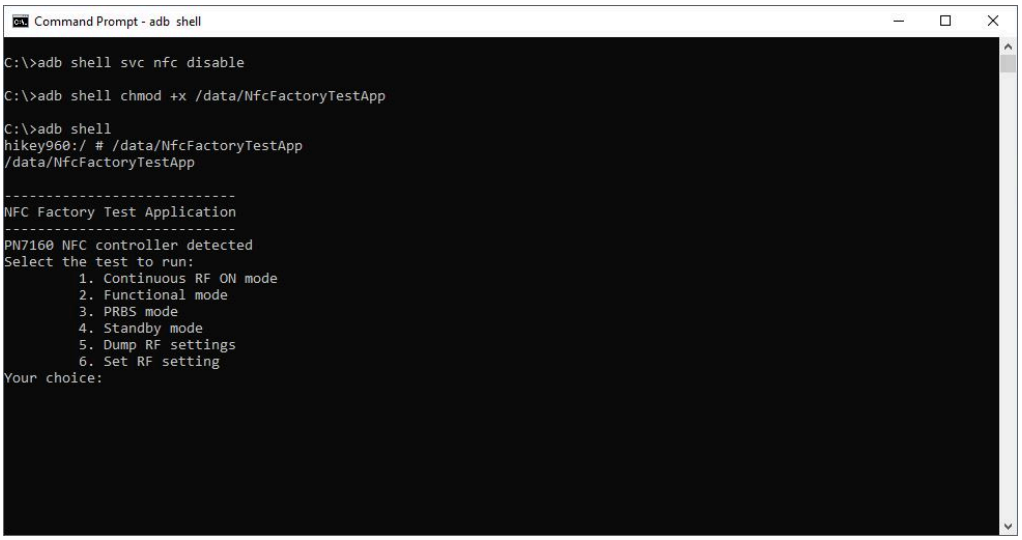

Figure 2. Running factory test native application on Android target

8 NFC Forum DTA application

To allow NFC Forum certification testing, a Device Test Application is provided. It is comprised of several components in the different Android layers (see Figure 1) which must be built and included to the Android image.

Below is the recommended procedure:

- For Android 12:
  1. Retrieve DTA application source code, depending of the Android version integration:

```
$ git clone -b NFC_DTA_v12.18_OpnSrc https://github.com/NXPNFCProject/NXPAndroidDTA.git
 ${ANDROID_BUILD_TOP}/vendor/nxp/NXPAndroidDTA
```

  2. Patch the source code for PN7160:

```
$ cd ${ANDROID_BUILD_TOP}/vendor/nxp/NXPAndroidDTA/nfc-dta
$ patch -p1 <${ANDROID_BUILD_TOP}/vendor/nxp/nfc/patches/AROOT_system_nfc-dta.patch
```

  3. Create symlink to reference the DTA application:

```
$ ln -s ${ANDROID_BUILD_TOP}/vendor/nxp/NXPAndroidDTA/nfc-dta ${ANDROID_BUILD_TOP}/system/nfc-dta
```

  4. Build the DTA application components:

```
$ cd ${ANDROID_BUILD_TOP}/system/nfc-dta
$ mm -j
```

  5. Rebuild android system image to include DTA application:

```
$ croot
$ make snod
```

- For Android 11:
  1. Retrieve DTA application source code, depending of the Android version integration:

```
$ git clone -b NFC_DTA_v12.15_OpnSrc https://github.com/NXPNFCProject/NXPAndroidDTA.git
 ${ANDROID_BUILD_TOP}/vendor/nxp/NXPAndroidDTA
```

  2. Patch the source code for PN7160:

```
$ cd ${ANDROID_BUILD_TOP}/vendor/nxp/NXPAndroidDTA/nfc-dta
$ patch -p1 <${ANDROID_BUILD_TOP}/vendor/nxp/nfc/patches/nfc-dta.patch
```

  3. Create symlink to reference the DTA application:

```
$ ln -s ${ANDROID_BUILD_TOP}/vendor/nxp/NXPAndroidDTA/nfc-dta ${ANDROID_BUILD_TOP}/system/nfc-dta
```

  4. Build the DTA application components:

```
$ cd ${ANDROID_BUILD_TOP}/system/nfc-dta
$ mm -j
```

  5. Rebuild android system image to include DTA application:

```
$ croot
$ make snod
```

After flashing the target, the DTA application should then be present from the list of installed applications.

When started, the DTA application requests user to disable NFC function from the "settings" menu. Indeed, DTA application directly access the NFC function from the low-level libraries, thus NFC service must be disabled to prevent conflicts.

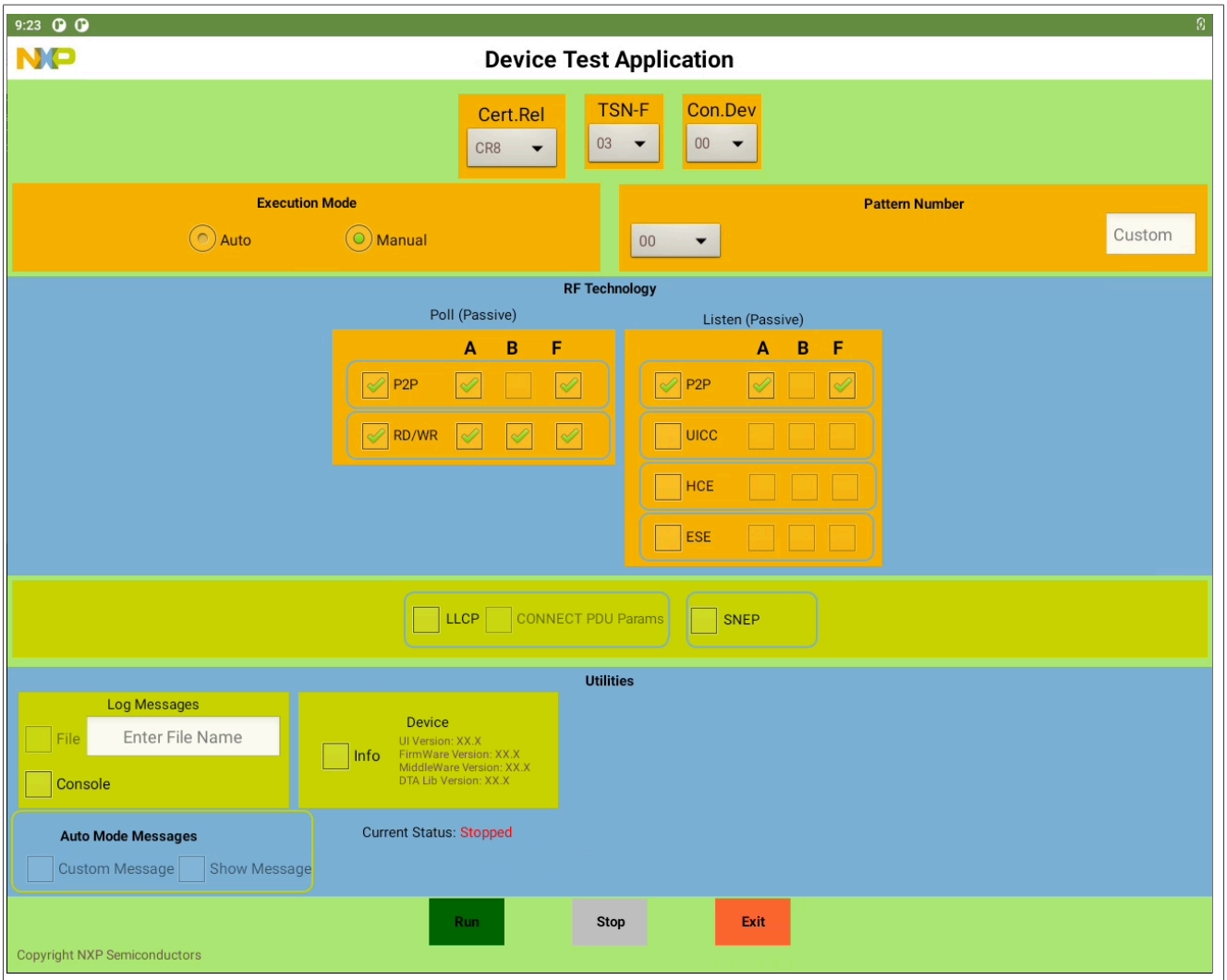

Figure 3. DTA application

"Cert.Rel" field must reflect Certification Release version targeted.

"TSN-F" field defines NFC-F technology Time Slot Number and must be set according to the test execution requirement.

"Con.Dev" field defines Connection Device Limit and must be set according to the test execution requirement.

Only "Manual" mode of "Execution Mode" is available for now, "Auto" mode being reserved for future use.

"Pattern Number" must be set according to the test execution requirement.

The RF technology tabs allow selecting individually each technology for each possible mode.

"LLCP" field allows enabling specific "Pattern Number" for dedicated test execution.

"SNEP" field allows running dedicated tests, requiring also "Android Beam" feature been enabled in the Android device settings.

"Log messages" field allows outputting the trace to a file (under “/sdcard/nxpdtalog/” folder) and/or a console.

9 NDEF emulation T4TDemo application

To demonstrate NDEF emulation feature in the Android device, the T4TDemo application is offered showing the use of specific API:

- doWriteT4tData: API to set the NDEF content to be exposed
- doReadT4tData: API to get the NDEF content currently exposed

The NDEF emulation feature is enabled according to the value of NXP_T4T_NFCEE_ENABLE parameter from *libnfc-nxp.conf* configuration file (see Section 6).

The source code is delivered together with the NXP's Android NFC delivery (see above Section 5.2.1, or Section 5.3.1).

The application is generated while building the system image, but it can also be independently built using following command:

```
$ mmm vendor/nxp/nfc/T4TDemo
```

Then install the application (generated under *out/target/product/**platform**/system/app/T4TDemo.apk)* to the Android target, using adb tool:

```
$ adb install T4TDemo.apk
```

On the Android target, the T4TDemo is then visible in the list of applications. Run it by clicking the related icon:

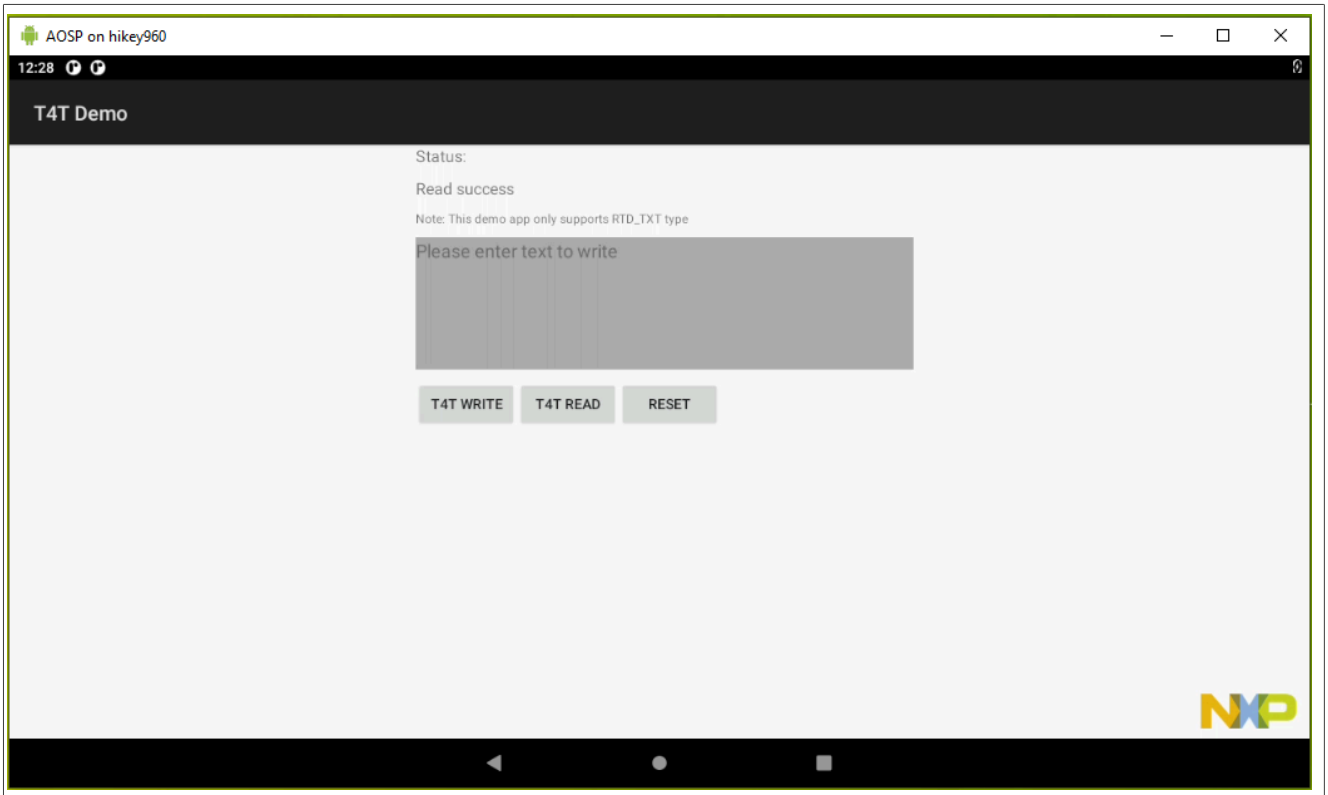

Figure 4. Running T4TDemo application on Android target

10 Troubleshooting

The following items may help figuring out what is going wrong in case NFC is not working as expected when starting the Android device.

10.1 Missing kernel driver or wrong device node rights

The following ADB logs may indicate nxpnfc driver is missing in the kernel or wrong rights are applied:

```
…
D NxpExtns: find found NXP_NFC_DEV_NODE=/dev/nxpnfc
D NxpTml  : getTransport Requested transportType: 2
D NxpTml  : OpenAndConfigure Opening port=/dev/nxpnfc
E NxpTml  : _i2c_open() Failed: retval ffffffff
E NxpHal  : phTmlNfc_Init Failed
D NxpHal  : Failed to deallocate (list empty)
D NxpHal  : Node dump:
D NxpHal  : Failed to deallocate (list empty)
D NxpHal  : Node dump:
E NxpHal  : phNxpNciHal_MinOpen failed
E NxpHal  : nxpncihal_monitor is null
…
```

The nxpnfc device node should usually appear with the following rights:

```
$ adb shell ls -als /dev/nxpnfc
crw-rw---- nfc      nfc       10,  54 2016-05-03 13:05 nxpnfc
```

If not listed in the */dev* folder, it means the module is not properly loaded as depicted in Section 4.3. Check kernel logs to see if error occurs during the module load or refer to the device tree definition.

In case the device node is seen with wrong rights, check the correct definition is present in */vendor/etc/init/init.**TargetProduct**.nfc.rc* file:

```
$ adb shell cat vendor/etc/init/init.TargetProduct.nfc.rc
on post-fs-data
    setprop ro.nfc.port "I2C"
    mkdir /data/vendor 0777 nfc nfc
    mkdir /data/vendor/nfc 0777 nfc nfc
    mkdir /data/vendor/nfc/param 0777 nfc nfc
    chmod 0660 /dev/pn544
    chown nfc nfc /dev/pn544
    chmod 0660 /dev/nxpnfc
    chown nfc nfc /dev/nxpnfc
```

10.2 Missing configuration files

The following ADB logs may indicate the absence of the libnfc-nci.conf file:

```
…
I com.android.nf: [0224/072605.700111:INFO:NfcJniUtil.cpp(47)] NFC Service: loading nci JNI
I com.android.nf: loadConfigEntry
F com.android.nf: nfc_config.cc:93] Check failed: config_path != ""
F com.android.nf: runtime.cc:655] Runtime aborting...
F com.android.nf: runtime.cc:655] Dumping all threads without mutator lock held
F com.android.nf: runtime.cc:655] All threads:
F com.android.nf: runtime.cc:655] DALVIK THREADS (17):
F com.android.nf: runtime.cc:655] "main" prio=5 tid=1 Runnable
F com.android.nf: runtime.cc:655]   | group="" sCount=0 dsCount=0 flags=0 obj=0x7215b448 self=0xb40000748b64f010
…
```

The libnfc-nci.conf configuration file should be present in the android system under */system/etc*, if this is not the case, refer to related procedure in Section 5.2.4 or Section 5.3.4.

```
$ adb shell ls -als /system/etc/libnfc*
4 -rw-r--r-- 1 root root 3428 2021-02-24 07:34 /system/etc/libnfc-nci.conf
```

The following ADB logs may indicate the absence of the libnfc-nxp.conf file:

```
…
I com.android.nf: [0224/072835.710998:INFO:NfcJniUtil.cpp(47)] NFC Service: loading nci JNI
I com.android.nf: loadConfigEntry
I com.android.nf: ConfigFile - Parsing file '/etc/libnfc-nci.conf'
I com.android.nf: ConfigFile - [APPL_TRACE_LEVEL] = 0xFF
I com.android.nf: ConfigFile - [PROTOCOL_TRACE_LEVEL] = 0xFFFFFFFF
I com.android.nf: ConfigFile - [NFC_DEBUG_ENABLED] = 0x01
I com.android.nf: ConfigFile - [NFA_STORAGE] = "/data/vendor/nfc"
I com.android.nf: ConfigFile - [HOST_LISTEN_TECH_MASK] = 0x07
I com.android.nf: ConfigFile - [SCREEN_OFF_POWER_STATE] = 1
I com.android.nf: ConfigFile - [NCI_HAL_MODULE] = "nfc_nci.pn54x"
I com.android.nf: ConfigFile - [POLLING_TECH_MASK] = 0xEF
I com.android.nf: ConfigFile - [P2P_LISTEN_TECH_MASK] = 0xC5
I com.android.nf: ConfigFile - [PRESERVE_STORAGE] = 0x01
I com.android.nf: ConfigFile - [AID_MATCHING_MODE] = 0x03
I com.android.nf: [0224/072835.713008:INFO:NfcAdaptation.cc(633)] Failed to retrieve the NXP NFC HAL!
I com.android.nf: [0224/072835.713158:INFO:NfcAdaptation.cc(639)] NfcAdaptation::InitializeHalDeviceContext:
 INfc::getService()
I com.android.nf: [0224/072835.714035:INFO:NfcAdaptation.cc(650)] NfcAdaptation::InitializeHalDeviceContext:
 INfc::getService() returned 0xb4000077ae5bc710 (remote)
I com.android.nf: [0224/072835.714403:INFO:NfcAdaptation.cc(657)] NfcAdaptation::InitializeHalDeviceContext:
 INfc::getService() returned 0xb4000077ae5bc710 (remote)
…
```

The libnfc-nxp.conf configuration file should be present in the android system under */vendor/etc*, if this is not the case, refer to related procedure in Section 5.2.4 or Section 5.3.4.

```
$ adb shell ls -als /vendor/etc/libnfc*
12 -rw-r--r-- 1 root root 9775 2021-03-11 11:02 /vendor/etc/libnfc-nxp.conf
```

10.3 Missing NXP's NFC libraries

The following ADB logs may indicate missing NFC-specific libraries:

```
…
I NfcService: Starting NFC service
D AndroidRuntime: Shutting down VM
E AndroidRuntime: FATAL EXCEPTION: main
E AndroidRuntime: Process: com.android.nfc, PID: 3503
E AndroidRuntime: java.lang.UnsatisfiedLinkError: dlopen failed: library "libnfc_nci_jni.so" not found
E AndroidRuntime:  at java.lang.Runtime.loadLibrary0(Runtime.java:1087)
E AndroidRuntime:  at java.lang.Runtime.loadLibrary0(Runtime.java:1008)
E AndroidRuntime:  at java.lang.System.loadLibrary(System.java:1664)
E AndroidRuntime:  at com.android.nfc.dhimpl.NativeNfcManager.<clinit>(NativeNfcManager.java:47)
E AndroidRuntime:  at com.android.nfc.NfcService.<init>(NfcService.java:438)
E AndroidRuntime:  at com.android.nfc.NfcApplication.onCreate(NfcApplication.java:66)
E AndroidRuntime:  at android.app.Instrumentation.callApplicationOnCreate(Instrumentation.java:1192)
E AndroidRuntime:  at android.app.ActivityThread.handleBindApplication(ActivityThread.java:6712)
E AndroidRuntime:  at android.app.ActivityThread.access$1300(ActivityThread.java:237)
E AndroidRuntime:  at android.app.ActivityThread$H.handleMessage(ActivityThread.java:1913)
E AndroidRuntime:  at android.os.Handler.dispatchMessage(Handler.java:106)
E AndroidRuntime:  at android.os.Looper.loop(Looper.java:223)
E AndroidRuntime:  at android.app.ActivityThread.main(ActivityThread.java:7656)
E AndroidRuntime:  at java.lang.reflect.Method.invoke(Native Method)
E AndroidRuntime:  at com.android.internal.os.RuntimeInit$MethodAndArgsCaller.run(RuntimeInit.java:592)
E AndroidRuntime:  at com.android.internal.os.ZygoteInit.main(ZygoteInit.java:947)

…
```

The library should be located under */system/lib/hw* android target subdirectory (or under */system/lib64/hw* if the platform is 64 bits).

```
$ adb shell ls -als /system/lib64/*libnfc*
844 -rw-r--r-- 1 root root 861528 2019-12-18 17:55 /system/lib64/libnfc-nci.so
680 -rw-r--r-- 1 root root 695952 2019-12-19 16:25 /system/lib64/libnfc_nci_jni.so

$ adb shell ls -als /system/lib64/vendor.nxp.nxpnfc@1.0.so
88 -rw-r--r-- 1 root root 88368 2019-11-20 14:13 /system/lib64/vendor.nxp.nxpnfc@1.0.so
```

If this is not the case, insure it is properly built:

```
$ croot
```

```
$ mmm system/nfc
$ mmm package/apps/Nfc
$ make systemimage
```

You can then either flash the newly created *system.img* or just copy the library to the android target:

```
$ adb push $OUT/system/lib64/libnfc-nci.so /system/lib64/
$ adb push $OUT/system/lib64/libnfc-nci.so /system/lib64/
$ adb push $OUT/system/lib64/vendor.nxp.nxpnfc@1.0.so /system/lib64/
```

10.4 Missing modules

The following ADB logs may indicate missing declaration of required NFC libraries:

```
…
W ActivityManager: Re-adding persistent process ProcessRecord{f8a0220 28966:com.android.nfc/1027}
I ActivityManager: Start proc 28995:com.android.nfc/1027 for restart com.android.nfc
I com.android.nf: ConfigFile - Parsing file '/etc/libnfc-nci.conf'
I com.android.nf: ConfigFile - [NFA_STORAGE] = "/data/vendor/nfc"
I com.android.nf: ConfigFile - [NCI_HAL_MODULE] = "nfc_nci.pn54x"
I hwservicemanager: getTransport: Cannot find entry vendor.nxp.nxpnfc@1.0::INxpNfc/default in either framework or
 device manifest.
I hwservicemanager: getTransport: Cannot find entry android.hardware.nfc@1.2::INfc/default in either framework or
 device manifest.
I hwservicemanager: getTransport: Cannot find entry android.hardware.nfc@1.1::INfc/default in either framework or
 device manifest.
I hwservicemanager: getTransport: Cannot find entry android.hardware.nfc@1.0::INfc/default in either framework or
 device manifest.
F libc : Fatal signal 11 (SIGSEGV), code 1 (SEGV_MAPERR), fault addr 0x0 in tid 28995 (com.android.nfc), pid 28995
 (com.android.nfc)
F DEBUG : pid: 28995, tid: 28995, name: com.android.nfc >>> com.android.nfc <<<
F DEBUG : #00 pc 00000000000ad8c8 /system/lib64/libnfc-nci.so (NfcAdaptation::InitializeHalDeviceContext()+1736)
 (BuildId: bc889132110efe73c4fc9e58e8776b54)
F DEBUG : #1 pc 00000000000ad1dc /system/lib64/libnfc-nci.so
…
```

Make sure that the related libraries are present on the target and also properly declared in manifest file (/vendor/etc/vintf/manifest.xml).

10.5 VTS testing

The following issues may be encountered during VTS testing.

10.5.1 Wrong interface

Wrong interface may be subject to test while it should not (for instance below “nfc-nci” while only “default” interface must be considered):

```
VtsHalNfcV1_0Target#NfcHidlTest.OpenAndClose(nfc_nci)_64bit fail Unknown failure.
VtsHalNfcV1_0Target#NfcHidlTest.WriteCoreReset(nfc_nci)_64bit fail Unknown error: test case requested but not
 executed.
```

“nfc-nci” interface must be undefined from “fqname” tag inside /vendor/etc/vintf/manifest.xml file.

10.5.2 Missing declaration

GetConfig test may fail because of missing declaration.

```
VtsHalNfcV1_1Target#NfcHidlTest.GetConfig(default)_64bit fail hardware/interfaces/nfc/1.1/vts/functional/
VtsHalNfcV1_1TargetTest.cpp:223
```

To fix this, add “ISO_DEP_MAX_TRANSCEIVE=0xFEFF” definition to “libnfc-nxp.conf” configuration file.

10.5.3 Wrong vendor properties namespace

testVendorPropertyNamespace test may fail because of wrong definition.

```
VtsTrebleSysProp#testVendorPropertyNamespace fail 2 != 0 vendor propertes (cts_gts.media.gts persist.nfc.) have
 wrong namespace armeabi-v7a VtsTrebleSysProp
Update sepolicy/property_contexts file with “persist.vendor.nfc.” instead of “persist.nfc.”.
```

11 Legal information

11.1 Definitions

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

11.2 Disclaimers

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

11.3 Licenses

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

11.4 Trademarks

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

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

AMBA, Arm, Arm7, Arm7TDMI, Arm9, Arm11, Artisan, big.LITTLE, Cordio, CoreLink, CoreSight, Cortex, DesignStart, DynamIQ, Jazelle, Keil, Mali, Mbed, Mbed Enabled, NEON, POP, RealView, SecurCore, Socrates, Thumb, TrustZone, ULINK, ULINK2, ULINK-ME, ULINK-PLUS, ULINKpro, µVision, Versatile — are trademarks and/or registered trademarks of Arm Limited (or its subsidiaries or affiliates) in the US and/or elsewhere. The related technology may be protected by any or all of patents, copyrights, designs and trade secrets. All rights reserved.

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

MIFARE — is a trademark of NXP B.V.

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

Tables

Figures

Contents

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

Date of release: 15 March 2022
Document identifier: AN13189