AN14608

PN7160/PN7220 – Android 15 porting guide

Rev. 2.0 — 3 June 2025 Application note

Document information

Information	Content
Keywords	PN7160, PN7220, NCI, EMVCo, NFC Forum, Android, NFC
Abstract	This document describes how to port PN7160/PN7220 common middleware release to Android 15.

1 Introduction

This guide provides detailed instructions on how to integrate NXP NCI-based NFC controllers, PN7160 and PN7220, into an Android environment. The process involves installing the necessary kernel driver and configuration of MW (see ref.[1]). For further information, refer to the product page for PN7160 ref.[2] and PN7220 ref.[3].

The Android Open Source Project (AOSP) has been updated to incorporate support for both PN7160 and PN7220 NFC controllers.

The PN7220 comes in two configurations: single-host and dual-host. The stack is generally the same for both. In dual-host mode, SMCU is added that means that all EMVCo related tasks are executed on SMCU. In single-host EMVCo is executed in a dedicated EMVCo MW stack.

2 Important notice

There are multiple tags related to Android 15 released on GitHub (ref.[1]). The table below explains each version:

Table 1. GitHub tags explanation

Tag	Explanation
NFC_AR_INFRA_001E_15.03.00_OpnSrc	Fully tested release.
NFC_AR_INFRA_001E_15.02.00_OpnSrc	Early release for customers that want to integrate PN722x into their device. Not production ready.
NFC_AR_INFRA_0006_15.01.01_OpnSrc	Release for PN7160 (full testing performed). PN7220 code is still there but with very limited testing completed.
NFC_AR_INFRA_001E_15.01.00_OpnSrc	Initial release. Limited testing completed.

Note: *NXP is extending the test coverage which is why some tags have limited test coverage at the moment.*

3 Android MW stack

Figure 1 illustrates the architecture of the PN7220 Android NFC stack.

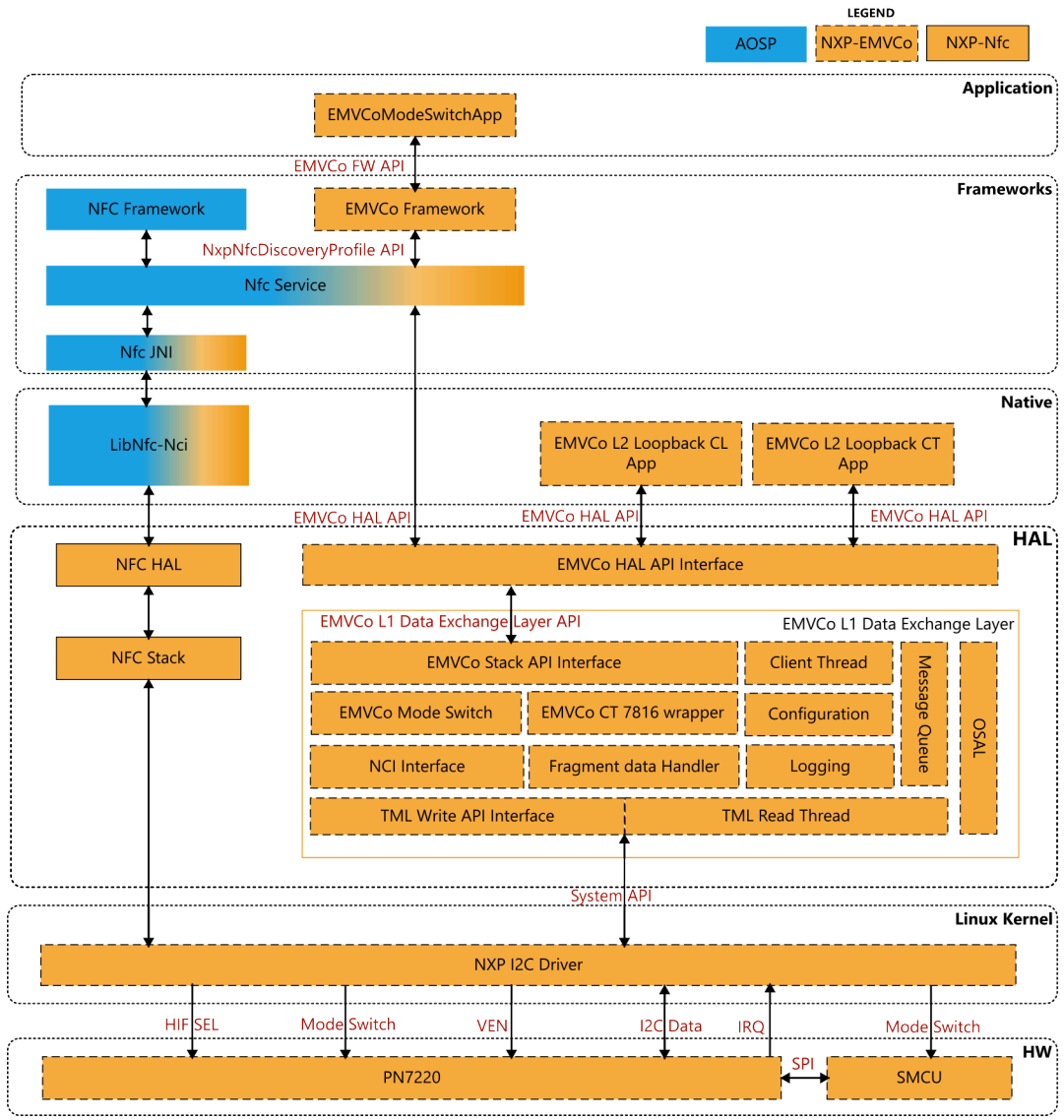

Figure 1. PN7220 Android NFC stack

- The NXP I2C Driver is a kernel module that allows access to the hardware resources of PN7220.
- The HAL module is an implementation of the NXP NFC controller-specific hardware abstraction layer.
- LibNfc-Nci is a native library that provides NFC functionality.
- NFC JNI acts as a bridge between Java and Native classes.
- The NFC and EMVCo Framework is a module of the application framework that allows access to NFC and EMVCo functionalities.

Figure 2 shows the architecture of the PN7160 Android NFC stack.

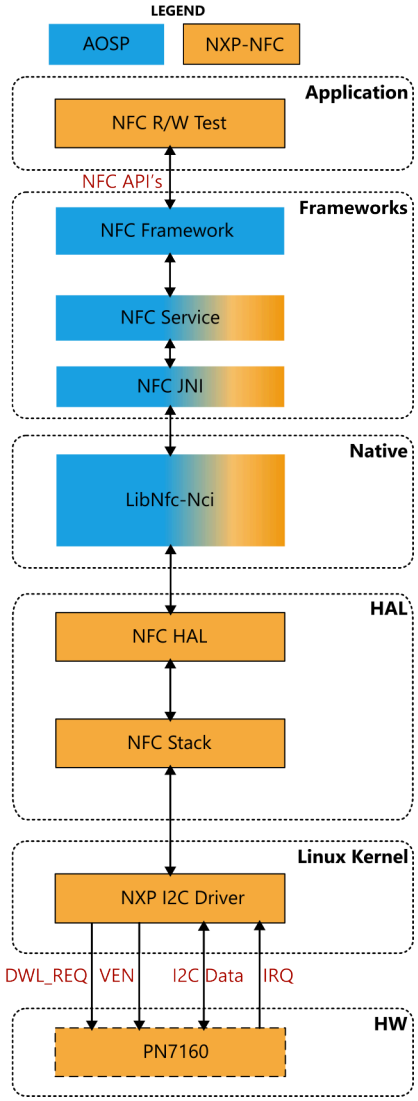

Figure 2. PN7160 Android MW stack

- The NXP I2C Driver is a kernel module that allows access to the hardware resources of PN7160.
- The HAL module is an implementation of the NXP NFC controller-specific hardware abstraction layer.
- LibNfc-nci is a native library that provides NFC functionality.
- NFC JNI acts as a bridge between Java and Native classes.
- The NFC is a module of the application framework that allows access to NFC functionalities.
- The MW source code is the same for PN7160 and PN7220, but there are a few limitations.

Table 2 shows unsupported features of each NFC controller.

Table 2. Unsupported features

NFC controller	Unsupported features
PN7160	• EMVCo MW stack • SMCU • CT feature
PN7220	• NFCEE_NDEF

Note: *From Android 14 onwards P2P is also not supported on PN7160.*

4 Kernel driver

To establish connection with the PN7220 or PN7160, the Android stack uses the nxpnfc kernel driver. It can be found in ref.[4].

4.1 Driver details

PN7220 supports I^2C physical interface, while PN7160 supports I^2C or SPI physical interface. When installed into the kernel, the driver is exposed via the device node in /dev/nxpnfc.

Note: *PN7160 and PN7220 use two different drivers, selection of the correct driver is required based on the chip type.*

4.2 Getting the PN7160 driver source code

Copy the *nfcandroid_platform_drivers/drivers/pn7160/nfc* driver repository into the kernel directory, replacing the existing implementation. Refer to ref.[4] for the kernel files.

```
$rm -rf drivers/nfc
$git clone "https://github.com/nxp-nfc-infra/nfcandroid_platform_drivers.git" -b
 br_ar_15_comm_infra_dev
```

This ends 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
- *nfc* subfolder containing:
  - *commoc.c*: generic driver implementation
  - *common.h*: generic driver interface definition
  - *i2c_drv.c*: i^{2}c specific driver implementation
  - *i2c_drv.h*: i^{2}c specific driver interface definition
  - *spi_drv.c*: spi specific driver implementation
  - *spi_drv.h*: spi specific driver interface definition
  - *Makefile*: makefile that is included in the makefile of the driver
  - *Kbuild* => build file
  - *Kconfig* => driver configuration file

4.3 Getting the PN7220 driver source code

Copy the *nfcandroid_platform_drivers/drivers/pn7220cs/nfc* (single-host use case) or *nfcandroid_platform_drivers/drivers/pn7220cms/nfc* (dual-host use case) into the kernel directory *drivers/nfc*, replacing the existing driver. Refer to ref.[4] for the kernel files.

```
$rm -rf drivers/nfc
$git clone "https://github.com/nxp-nfc-infra/nfcandroid_platform_drivers.git" -b
 br_ar_15_comm_infra_dev
```

Following this command, the folder drivers/nfc contains the following files:

- *README.md*: repository information
- *Makefile*: driver heading makefile
- *Kconfig*: driver configuration file
- *License*: driver licensing terms
- *nfc* subfolder containing:
  - *commoc.c*: generic driver implementation
  - *common.h*: generic driver interface definition
  - *i2c_drv.c*: i^{2}c specific driver implementation
  - *i2c_drv.h*: i^{2}c specific driver interface definition
  - *Makefile*: makefile that is included in the makefile of the driver
  - *Kbuild* => build file
  - *Kconfig* => driver configuration file

4.4 Building the driver

The devicetree is responsible for adding the driver to the kernel and loading it on device boot.

After upgrading the devicetree specification, the platform-related devicetree must be rebuilt. NXP recommends using kernel version 5.10 as it provides comprehensive validation.

To build the driver, the following steps must be performed:

1. Get the kernel driver
2. Get the source code for the driver
3. Modify the devicetree definition, which is unique to the device in use.
4. Build the driver:
   a. Through the menuconfig procedure, add the target driver into the build.

After rebuilding the completed kernel, the driver will be included in the kernel image. All new kernel images must be copied into the AOSP build.

5 AOSP adaptation

NXP adds modifications to the AOSP code. This means that the AOSP code is used as a foundation, but extended for NXP-specific features. ref.[5] is the current AOSP tag used by NXP. After obtaining the AOSP build, the existing AOSP code must be replaced, and a number of patches must be applied.

Note: *A different version of the AOSP code can be used, but additional modifications must be performed.*

5.1 AOSP build

1. Get AOSP source code.

```
$ repo init -u https://android.googlesource.com/platform/manifest -b
 android-15.0.0_r1 (check Section 2 for code releases)
$ repo sync
```

Note: *The repo tool must be installed on the system. Refer to ref.[6] for instructions.*

2. Build source code.

```
$cd Android_AROOT
$source build/envsetup.sh
$lunch select_target #target is DH we want to use for example: evk_8mn-userdebug
$make -j
```

3. Copy all NXP repositories into the target location.

Table 3. Branch for specific Android version

Android version	Branch
Android 15	br_ar_15_comm_infra_dev

Note: *While cloning, it is important to select the correct branch.*

Table 4. Clone repositories

AOSP Repos	NXP GitHub Repos
"$ANDROID_ROOT"/packages/apps/Nfc	https://github.com/nxp-nfc-infra/nxp_nci_hal_nfc/tree/br_ar_15_comm_infra_dev
"$ANDROID_ROOT"/system/nfc	>https://github.com/nxp-nfc-infra/nxp_nci_hal_libnfc-nci/tree/br_ar_15_comm_infra_dev
"$ANDROID_ROOT"/hardware/nxp/nfc	https://github.com/nxp-nfc-infra/nfcandroid_nfc_hidlimpl/tree/br_ar_15_comm_infra_dev
"$ANDROID_ROOT"/vendor/nxp/frameworks	https://github.com/nxp-nfc-infra/nfcandroid_frameworks/tree/br_ar_15_comm_infra_dev
"$ANDROID_ROOT"/hardware/nxp/emvco	https://github.com/nxp-nfc-infra/nfcandroid_emvco_aidlimpl/tree/br_ar_15_comm_infra_dev
“$ANDROID_ROOT”	https://github.com/nxp-nfc-infra/nfcandroid_platform_reference/tree/br_ar_15_comm_infra_dev

Table 5. Clone repositories for test applications and TDA support

Folder in GitHub	AOSP Repos	NXP GitHub	IC Supported
test_apps/SMCU_Switch	"$ANDROID_ROOT"/ packages/apps/	https://github.com/ nxp-nfc-infra/ nfcandroid_infra_test_apps	PN7220
test_apps/EMVCoMode SwitchApp	"$ANDROID_ROOT"/ packages/apps/Nfc/	https://github.com/ nxp-nfc-infra/ nfcandroid_infra_test_apps	PN7220
test_apps/Cockpit	Not applicable anymore. Use Cockpit tool from Quick start guide	Not applicable anymore. Use Cockpit tool from Quick start guide	Not applicable anymore. Use Cockpit tool from Quick start guide
test_apps/SelfTest	"$ANDROID_ROOT"/ hardware/nxp/nfc/	https://github.com/ nxp-nfc-infra/ nfcandroid_infra_test_apps	PN7220
test_apps/SelfTest_pn7160	"$ANDROID_ROOT"/ hardware/nxp/nfc/	https://github.com/ nxp-nfc-infra/ nfcandroid_infra_test_apps	PN7160
test_apps/load_unload	"$ANDROID_ROOT"/ hardware/nxp/nfc/	https://github.com/ nxp-nfc-infra/ nfcandroid_infra_test_apps	PN7220
test_apps/SelfTestAidl	"$ANDROID_ROOT"/ hardware/nxp/nfc/	https://github.com/ nxp-nfc-infra/ nfcandroid_infra_test_apps	PN7220
nfc_tda	"$ANDROID_ROOT"/system/	https://github.com/ nxp-nfc-infra/ nfcandroid_infra_comm_libs	PN7220
emvco_tda	"$ANDROID_ROOT"/ hardware/nxp/emvco/	https://github.com/ nxp-nfc-infra/ nfcandroid_infra_comm_libs	PN7220
emvco_tda_test	"$ANDROID_ROOT"/ hardware/nxp/emvco/	https://github.com/ nxp-nfc-infra/ nfcandroid_infra_comm_libs	PN7220
NfcTdaTestApp	“$ANDROID_ROOT”/ packages/apps/Nfc/	https://github.com/ nxp-nfc-infra/ nfcandroid_infra_comm_libs	PN7220

4. Apply patches.

Table 6. Apply patches

Location to apply	Patch to apply	Location of the patch
"$ANDROID_ROOT"/build/bazel/	*AROOT_build_bazel.patch*	https://github.com/nxp-nfc-infra/nfcandroid_platform_reference/tree/br_ar_15_comm_infra_dev/build_cfg/build_pf_patches/
"$ANDROID_ROOT"/build/release/	*AROOT_build_make.patch*	https://github.com/nxp-nfc-infra/nfcandroid_platform_reference/tree/br_ar_15_comm_infra_dev/build_cfg/build_pf_patches/
"$ANDROID_ROOT"/external/libchrome/	*AROOT_build_soong.patch*	https://github.com/nxp-nfc-infra/nfcandroid_platform_reference/tree/br_ar_15_comm_infra_dev/build_cfg/build_pf_patches/
"$ANDROID_ROOT"/frameworks/base/	*AROOT_frameworks_base.patch*	https://github.com/nxp-nfc-infra/nfcandroid_platform_reference/tree/br_ar_15_comm_infra_dev/build_cfg/build_pf_patches/
"$ANDROID_ROOT"/system/logging/	*AROOT_system_logging.patch*	https://github.com/nxp-nfc-infra/nfcandroid_platform_reference/tree/br_ar_15_comm_infra_dev/build_cfg/build_pf_patches/

Note: *Check the output after applying the patch, if any issue was observed during the patching.*

5. Add FW libraries. Refer to ref.[8] for FW.

Note: *Not mandatory. FW can always be updated.*

For PN7160:

```
$git clone https://github.com/NXP/nfc-NXPNFCC_FW.git
$cp -r nfc-NXPNFCC_FW/InfraFW/pn7220/64-bit/libpn7160_fw.so AROOT/vendor/
nxp/7160/firmware/lib64/libpn7160_fw.so
$cp -r nfc-NXPNFCC_FW/InfraFW/pn7220/32-bit/libpn7160_fw.so AROOT/vendor/
nxp/7160/firmware/lib/libpn7160_fw.so
```

For PN7220:

```
$git clone https://github.com/NXP/nfc-NXPNFCC_FW.git
$cp -r nfc-NXPNFCC_FW/InfraFW/pn7220/64-bit/libpn7220_64bit.so AROOT/vendor/nxp/
pn7220/firmware/lib64/libpn72xx_fw.so
```

6. Adding NFC to the build

In the *device.mk* makefile (for example, device/brand/platform/device.mk), include specific makefiles:

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

In the *BoardConfig.mk* makefile (for example, *device/brand/platform/BoardConfig.mk*), include a specific makefile:

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

7. Adding the DTA application

```
$git clone https://github.com/NXPNFCProject/NXPAndroidDTA.git
$cd NXPAndroidDTA
$git checkout br_ar_new_dta_arch
$cp -r NXPAndroidDTA /vendor/nxp/ #User can clone it into vendor/nxp/
NXPAndroidDTA directly
$<AROOT>/vendor/nxp/NXPAndroidDTA$ mm -j
```

8. Build AOSP with changes:

```
$cd framework/base
$mm
$cd ../..
$cd vendor/nxp/frameworks
$mm #after this one, com.nxp.emvco.jar and com.nxp.nfc.jar should be inside out/
target/product/xxxx/system/framwework/
$cd ../../..
$cd hardware/nxp/nfc
$mm
$cd ../../..
$make -j
```

Now, flash the device with new Android images.

5.2 Android 15 NFC Apps and Lib on targets

After the build, the created libraries must be installed on the target device. Section 5.2 specifies the project location, the corresponding library, and the target device location where to be installed.

Note: *EMVCo binaries are applicable only for PN7220.*

Table 7. Compiled files with device target

Project location	Compiled Files	Comments	Location in target device
"$ANDROID_ROOT"/ packages/apps/Nfc	*NfcNci.odex* *NfcNci.vdex* *lib/NfcNci.apk* *oat/libnfc_nci_jni.so*		*/system/priv-app/NfcNci/oat/arm64/* */system/priv-app/NfcNci/oat/arm64/* */system/priv-app/NfcNci/* */system/lib64/*
"$ANDROID_ROOT"/ system/nfc	*libnfc_nci.so*		*/system/lib64/*
"$ANDROID_ROOT"/ system/nfc_tda"	*nfc_tda.so*	Applicable only for CT feature.	*/system/lib64/*
"$ANDROID_ROOT"/ hardware/nxp/nfc	*nfc_nci_nxp_pn72xx.so* *android.hardware.nfc-service.nxp* *nfc-service-nxp.rc* *vendor.nxp.nxpnfc_aidl-V2-ndk.so* *vendor.nxp.nxpnfc_aidl-V1-ndk.so* *vendor.nxp.nxpnfc@1.0.so* *vendor.nxp.nxpnfc@2.0.so*		*/vendor/lib64* */vendor/bin/hw/* */vendor/etc/init* */vendor/lib64/* */vendor/lib64/* */vendor/lib64/* */vendor/lib64/*
"$ANDROID_ROOT/ hardware/interfaces/nfc"	*android.hardware.nfc-V1-ndk.so* *android.hardware.nfc@1.0.so* *android.hardware.nfc@1.1.so* *android.hardware.nfc@1.2.so*		*/vendor/ib64/* */vendor/lib64/* */vendor/lib64/* */vendor/lib64/*
"$ANDROID_ROOT"/ vendor/nxp/frameworks	*com.nxp.emvco.jar (PN7220)* *com.nxp.nfc.jar*		*/system/framework/* */system/framework/*
"$ANDROID_ROOT"/ hardware/nxp/emvco	*emvco_poller.so (PN7220)* *vendor.nxp.emvco-V1-ndk.so* *vendor.nxp.emvco-V2-ndk.so* *vendor.nxp.emvco-V2-ndk.so* *vendor.nxp.emvco-service* *vendor.nxp.emvco-service.rc*		*/vendor/lib64/* */system/lib64/* */system/lib64/* */vendor/lib64/* */vendor/bin/hw/* */vendor/etc/init/*
"$ANDROID_ROOT/ hardware/nxp/emvco_tda"	*emvco_tda.so*	Applicable only for CT feature.	*/vendor/lib64/*

5.3 Block mapping

Mapping the block name from Section 1 to target location in AOSP code.

Table 8. Patch location in NFC Stack

Block name	Location in AOSP code
NFC HAL and EMVCo HAL	*hardware/interfaces/*
NFC Stack	*hardware/nxp/nfc/*
EMVCo L1 Data Exchange Layer = EMVCo Stack	*hardware/nxp/emvco/*
LibNfc-Nci	*system/nfc/*
NFC JNI	*packages/apps/nfc/*
NFC Service	*packages/apps/nfc/*
NFC Framework	*frameworks/base/*
EMVCo Framework	*vendor/nxp/frameworks/*

5.4 EMVCo API

PN7220 MW stack extends AOSP code with EMVCo MW stack. This section describes the EMVCo APIs.

Note: *APIs can be called only when using PN7220 IC. If calling it with PN7160 IC, the API does not work.*

EMVCo Profile Discovery. Those APIs can be used with contact and contactless profiles.

- **registerEMVCoEventListener()**
  - ```
    ndk::ScopedAStatus registerEMVCoEventListener ( const std::shared_ptr<
     INxpEmvcoClientCallback > & in_clientCallback,
    bool * in_aidl_return
    )
    ```
  - **Description**: Register EMVCo callback function to receive the events from a listener device
  - ***Note:*** *This function is must to ball before invoking any other api.*
  - **Parameters**:
    - *[in] *in_clientCallback:* has EMVCo client HAL callback
    - *[in] *in_aidl_return:* indicates register status in return to caller
  - **Returns**
    - boolean returns true, if success and returns false, if failed to register
- **getCurrentDiscoveryMode()**
  - ```
    ndk::ScopedAStatus
     getCurrentDiscoveryMode(::aidl::vendor::nxp::emvco::NxpDiscoveryMode *
    _aidl_return)
    ```
  - **Description**: returns the current active profile type.
  - **Returns**
    - NxpDiscoveryMode - NFC/EMVCo/Unknown

- **onNfcStateChange()**
  - ```
    ndk::ScopedAStatus onNfcStateChange(NxpNfcState in_nfcState)
    ```
  - **Description**: updated NFC state to EMVCo HAL.
  - **Parameters**:
    - *[in] in_nfcState:* specifies the NFC state
  - **Returns**:
    - void
- **registerNFCStateChangeCallback()**
  - ```
    ndk::ScopedAStatus registerNFCStateChangeCallback ( const
     std::shared_ptr< ::aidl::vendor::nxp::emvco::INxpNfcStateChangeRequestCallback
     > & in_nfcStateChangeRequestCallback,
    bool * _aidl_return
    )
    ```
  - **Description**: Register an NFC callback function to receive the events from a listener device.
  - ***Note:*** *This function is must call before invoking any other api.*
  - **Parameters**:
    - *[in] in_nfcStateChangeCallback:* INxpNfcStateChangeRequestCallback the event callback function to be passed by the caller. It should implement to turn ON/OFF NFC based on the request received.
  - **Returns**: boolean returns true, if success and returns false, if failed to register.
- **setByteConfig()**

```
ndk::ScopedAStatus setByteConfig ( ::aidl::vendor::nxp::emvco::NxpConfigType
 in_type,
 int32_t in_length,
 int8_t in_value,
 ::aidl::vendor::nxp::emvco::NxpEmvcoStatus * _aidl_return
)
```

- **setEMVCoMode()**

```
ndk::ScopedAStatus setEMVCoMode ( int8_t in_disc_mask,
 bool in_isStartEMVCo
)
```

  - **Description**: Starts the EMVCo mode with the Device-Controller. Once the Application Data Channel is established, the Application may send start the EMVCo mode with the Device-Controller.
  - **Parameters**:
    - *[in] in_disc_mask EMVCo:* polling technologies are configured through this parameter
    - *[in]in_isStartEMVCo:* specifies to start or stop the EMVCo mode
  - **Returns**:
    - void
- **setLed()**

```
ndk::ScopedAStatus setLed ( ::aidl::vendor::nxp::emvco::NxpLedControl
 in_ledControl,
 ::aidl::vendor::nxp::emvco::NxpEmvcoStatus * emvco_status
)
```

For Contact EMVCo, the following APIs can be used on top of the previous ones.

- **closeTDA()**

```
ndk::ScopedAStatus closeTDA ( int8_t in_tdaID,
bool in_standBy
)
```

  - **Description**: Closes the smart card connected over TDA
  - **Parameters**:
    - *[in] tdaID:* id of the tda slot to be closed
  - **Exceptions**:
    - EMVCO_STATUS_INVALID_PARAMETER, if provided tdaID is in-valid
    - EMVCO_STATUS_FEATURE_NOT_SUPPORTED when the contact card feature is not supported.
  - **Returns**:
    - void

- **discoverTDA()**

```
ndk::ScopedAStatus discoverTDA
 ( std::vector<::aidl::vendor::nxp::emvco::NxpEmvcoTDAInfo > * emvcoTDAInfo )
```

  Description: discoverTDA provides all the details of smart card connected over TDA
  - **Parameters**:
    - *[in]*in_clientCallback:* provides EMVCo state and TDA state as callback
  - **Exceptions**:
    - – EMVCO_STATUS_FEATURE_NOT_SUPPORTED when the contact card feature is not supported.
  - **Returns**:
    - NxpEmvcoTDAInfo[] returns all the smart card connected over TDA. valid emvcoTDAInfo is received only when the status is EMVCO_STATUS_OK

- **openTDA()**

```
ndk::ScopedAStatus openTDA ( int8_t in_tdaID,
bool in_standBy,
int8_t * out_connID
)
```

  Description: opens the smart card connected over TDA
  - **Parameters:**
    - *[in]tdaID:* tda id of the smart card received through discoverTDA
  - **Exceptions**:
    - EMVCO_STATUS_INVALID_PARAMETER, if provided tdaID is in-valid
    - EMVCO_STATUS_FEATURE_NOT_SUPPORTED when the contact card feature is not supported.
  - **Returns**:
    - byte returns the connection id of the smart card. valid connection id received only when status is EMVCO_STATUS_OK

- **registerEMVCoCTListener()**

```
ndk::ScopedAStatus registerEMVCoCTListener ( const
 std::shared_ptr<::aidl::vendor::nxp::emvco::INxpEmvcoTDACallback > &
 in_in_clientCallback,
bool * _aidl_return
)
```

  - **Description**: registers the EMVCoCT callback to the EMVCo stack
  - **Parameters**:
    - *[in]*in_in_clientCallback:* provides EMVCo state and TDA state as callback
  - **Returns**:
    - void

- **transceive()**

```
ndk::ScopedAStatus transceive ( const std::vector< uint8_t > & in_cmd_data,
std::vector< uint8_t > * out_rsp_data
)
```

  - **Description**: sends application data with the Device-Controller and receives response data from the controller
  - ***Note:*** *connection id of the TDA should be added as part of the NCI header.*
  - **Parameters**:
    - *[in]in_cmd_data:* Application command data buffer
  - **Exceptions**:
    - EMVCO_STATUS_INVALID_PARAMETER, if provided connection id is in-valid
    - EMVCO_STATUS_FEATURE_NOT_SUPPORTED when the contact card feature is not supported.
  - **Returns**:
    - Response APDU received from controller. valid Response APDU received only when status is EMVCO_STATUS_OK

For EMVCo contactless, the following APIs can be called:

- **registerEMVCoEventListener()**

```
ndk::ScopedAStatus registerEMVCoEventListener ( const std::shared_ptr<
 INxpEmvcoClientCallback > & in_clientCallback,
bool * _aidl_return
)
```

  - **Description**: Register an EMVCo callback function to receive the events from a listener device.
  - ***Note:*** *This function is must call before invoking any other api.*
  - **Parameters**:
    - *[in]*in_clientCallback:* has EMVCo client HAL callback
    - *[in]*in_aidl_return:* indicates register status in return to caller
  - **Returns**:
    - boolean returns true, if success and returns false, if failed to register

- **setEMVCoMode()**

```
ndk::ScopedAStatus setEMVCoMode ( int8_t in_config,
bool in_isStartEMVCo
)
```

  - **Description**: Starts the EMVCo mode with the Device-Controller. Once the Application Data Channel is established, the Application may send start the EMVCo mode with the Device-Controller.
  - **Parameters**:
    - *[in]in_config:* EMVCo polling technologies are configured through this parameter
    - *[in]in_isStartEMVCo:* specifies to start or stop the EMVCo mode
  - **Returns**:
    - void

- **stopRFDisovery()**

```
ndk::ScopedAStatus stopRFDisovery
 ( ::aidl::vendor::nxp::emvco::NxpDeactivationType in_deactivationType,
::aidl::vendor::nxp::emvco::NxpEmvcoStatus * emvco_status
)
```

  - **Description**: stops the RF field and moves in to the specified deactivation state.
  - **Parameters**:
    - *[in]in_deactivationType:* specifies the state to be in after RF deactivation
  - **Returns**:
    - NxpEmvcoStatus returns EMVCO_STATUS_OK if command processed successfully and returns EMVCO_STATUS_FAILED, if command is not processed due to in-valid state. EMVCo mode should be ON to call this API

- **transceive()**

```
ndk::ScopedAStatus transceive ( const std::vector< uint8_t > & in_data,
int32_t * _aidl_return
)
```

  - **Description**: send application data with the Device-Controller.
  - ***Note:*** *In case if send data is failed, the Application shall again invoke open() before invoking this API.*
  - **Parameters**:
    - *[in]in_data:* Application data buffer
  - **Returns**:
    - NxpEmvcoStatus indicating execution status

5.5 Configuration files PN7160

For PN7160, there are two different configuration files.

1. *libnfc-nci.conf*
2. *libnfc-nxp.conf*

Note: Configuration files provided by NXP are examples related to the NFC controller demo board. These files must be adopted according to the targeted integration.

Configuration files must be placed in the target location (see Table 9).

Table 9. Locations of configuration files

Name of configuration file	Location in device
libnfc-nci.conf	*system/etc*
libnfc-nxp.conf	*vendor/etc*

To get more informations on the configuration files, see ref.[9].

5.6 Configuration files PN7220

For PN7220, there are five different configuration files.

1. *libemvco-nxp.conf*
2. *libnfc-nci.conf*
3. *libnfc-nxp.conf*
4. *libnfc-nxp-eeprom.conf*
5. *libnfc-nxp-rfExt.conf*

Note: Configuration files provided by NXP are examples related to the NFC controller demo board. These files must be adopted according to the targeted integration.

Configuration files need to be placed in the target location (see Table 10).

Table 10. Locations of configuration files

Name of configuration file	Location in device
libemvco-nxp.conf	*vendor/etc*
libnfc-nci.conf	*system/etc*
libnfc-nxp.conf	*vendor/etc*
libnfc-nxp-eeprom.conf	*vendor/etc*
libnfc-nxprfExt.conf	*vendor/etc*

To get more informations on the configuration files, see ref.[9].

5.7 DTA application

To allow NFC Forum certification testing, a device test application is provided. It is composed of several components in the different Android layers, which must be built and included in the Android image.

To push the DTA application, the following steps must be executed:

1. Copy DTA apk to one location:

```
$cp -rf "out/target/product/evk_8mm/vendor/app/NXPDTA/NXPDATA.apk" /DTA-
PN7220
```

2. Install the apk:

```
adb install NXPDTA.apk
```

After flashing the target, the DTA application should then be present in the list of installed applications. Refer to ref.[7] for a detailed description of how to use the application.

6 Abbreviations

Table 11. Abbreviations

Acronym	Description
APDU	application protocol data unit
AOSP	Android Open Source Project
DH	device host
HAL	hardware abstraction layer
FW	firmware
I2C	Inter-Integrated Circuit
LPCD	lower powered card detection
NCI	NFC controller interface
NFC	near-field communication
MW	middleware
PLL	phase-locked loop
P2P	peer to peer
RF	radio frequency
SDA	serial data
SMCU	secure microcontroller
SW	software

7 References

[1] GitHub repository – PN7160 and PN7220 Common MW (link)
[2] Web page – PN7160 – NFC Plug and Play Controller with Integrated Firmware and NCI Interface (link)
[3] Web page – PN7220 – EMV L1 Compliant NFC Controller with NCI Interface Supporting EMV and NFC Forum Applications (link)
[4] GitHub repository – PN7160 and PN7220 kernel driver (link)
[5] Resources – AOSP r2 tag (link)
[6] Resources – Source control tools (link)
[7] User guide – UG10068 – PN7220 – Quick start guide (link)
[8] GitHub repository – PN7160 and PN7220 FW location (link)
[9] Application note – AN14431 – PN7160/PN7220 configuration files (link)

8 Note about the source code in the document

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

Copyright 2025 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.

9 Revision history

Table 12. Revision history

Document ID	Release date	Description
AN14608 v.2.0	3 June 2025	• Section 2 "Important notice": updated • Section 5.2 "Android 15 NFC Apps and Lib on targets": updated
AN14608 v.1.0	14 April 2025	• 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.

HTML publications — An HTML version, if available, of this document is provided as a courtesy. Definitive information is contained in the applicable document in PDF format. If there is a discrepancy between the HTML document and the PDF document, the PDF document has priority.

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.

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

Tables

Figures

Contents