Getting started with the X-CUBE-NFC3 near field communication transceiver software expansion for STM32Cube

Introduction

The X-CUBE-NFC3 software expansion for STM32Cube provides a complete middleware for STM32 products to control applications using ST25R95 or CR95HF near field communication transceivers.

The software is based on STM32Cube technology and expands STM32Cube-based packages. It is built on top of STM32Cube software technology to ease portability across different STM32 microcontrollers.

The software comes with sample implementations of the drivers running on the X-NUCLEO-NFC03A1 expansion board plugged on top of a NUCLEO-L476RG, NUCLEO-F401RE or NUCLEO-F103RB board.

1 General information

X-CUBE-NFC3 runs on STM32 microcontrollers, which are based on Arm® Cortex® processors.

arm

Note: Arm is a registered trademark of Arm Limited (or its subsidiaries) in the US and/or elsewhere.

1.1 Acronyms and abbreviations

Table 1. List of acronyms

Acronym	List of acronyms
BSP	Boot support package
CMSIS	Arm Cortex microcontroller software interface standard
HAL	Hardware abstraction layer
LED	Light emitting diode
MCU	Microcontroller unit
NFC	Near field communication
RFAL	RF abstract layer
SDK	Software development kit
SPI	Serial peripheral interface

1.2 Reference documents

The following documents are available on *www.st.com*:

- ST25R95 or CR95HF datasheets
- STM32 microcontroller datasheets
- Nucleo board user manuals.

2 STM32Cube overview

2.1 What is STM32Cube?

STM32Cube is an STMicroelectronics original initiative to significantly improve designer's productivity by reducing development effort, time and cost. STM32Cube covers the whole STM32 portfolio.

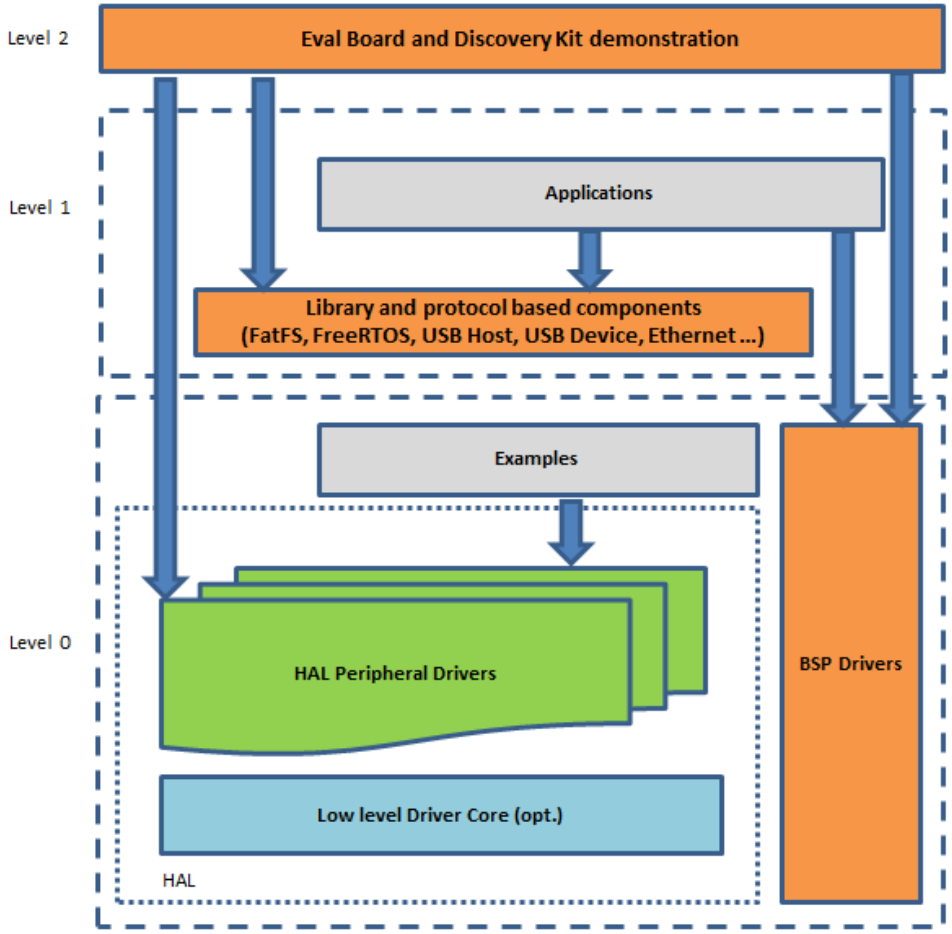
STM32Cube includes:

- A set of user-friendly software development tools to cover project development from the conception to the realization, among which:
  - STM32CubeMX, a graphical software configuration tool that allows the automatic generation of C initialization code using graphical wizards
  - STM32CubeIDE, an all-in-one development tool with peripheral configuration, code generation, code compilation, and debug features
  - STM32CubeProgrammer (STM32CubeProg), a programming tool available in graphical and command-line versions
  - STM32CubeMonitor-Power (STM32CubeMonPwr), a monitoring tool to measure and help in the optimization of the power consumption of the MCU
- STM32Cube MCU & MPU Packages, comprehensive embedded-software platforms specific to each microcontroller and microprocessor series (such as STM32CubeG4 for the STM32G4 Series), which include:
  - STM32Cube hardware abstraction layer (HAL), ensuring maximized portability across the STM32 portfolio
  - STM32Cube low-layer APIs, ensuring the best performance and footprints with a high degree of user control over the HW
  - A consistent set of middleware components such as FAT file system, RTOS, USB Device, and USB Power Delivery
  - All embedded software utilities with full sets of peripheral and applicative examples
- STM32Cube Expansion Packages, which contain embedded software components that complement the functionalities of the STM32Cube MCU & MPU Packages with:
  - Middleware extensions and applicative layers
  - Examples running on some specific STMicroelectronics development boards

2.2 STM32Cube architecture

The STM32Cube firmware solution is built around three independent levels that can easily interact with each other's as described in Figure 1. Firmware architecture:

Figure 1. **Firmware architecture**



Level 0: this level is divided into three sub-layers:

- Board support package (BSP): this layer offers a set of APIs relative to the hardware components in the hardware boards (audio codec, IO expander, Touchscreen, SRAM driver, LCD drivers, and so on) and is composed of two parts:
  - **Component**: is the driver relative to the external device on the board and not related to the STM32. The component driver provide specific APIs to the BSP driver external components and could be portable on any other board.
  - **BSP driver**: it permits to link the component driver to a specific board and provides a set of friendly used APIs. The APIs naming rule is BSP_FUNCT_Action(): ex. BSP_LED_Init(),BSP_LED_On().

Level 0 is based on modular architecture allowing to port it easily on any hardware by just implementing the low level routines.

- **Hardware abstraction layer (HAL)**: this layer provides the low-level drivers and hardware interfacing methods to interact with the upper layers (application, libraries and stacks). The HAL provides a generic, multi-instance and functionality-oriented APIs that permit the user application implementation to be offloaded by providing a ready-to-use process. For example, for the communication peripherals (I2S, UART and so on), it provides APIs allowing the initialization and configuration of the peripheral, data-transfer management based on polling, interrupt or DMA process, and management of communication errors that may arise during communication.

  The HAL driver APIs are split in two categories:
  - **generic APIs**: these provide common and generic functions to all STM32 Series devices.
  - **extension APIs**: these provide customized functions for a specific device family or part number.

- **Middleware components**: set of libraries covering USB Host and Device Libraries, STemWin, FreeRTOS, FatFS, LwIP, and PolarSSL. Horizontal interactions between the components of this layer is done directly by calling the feature APIs, while the vertical interaction with the low-level drivers is done through specific callbacks and static macros implemented in the library system call interface. For example, the FatFs implements the disk I/O driver to access microSD drive or the USB Mass Storage Class.
- **Examples based on the middleware components**: each middleware component comes with one or more examples (called also applications) showing how to use it. Integration examples that use several middleware components are also provided.

Level 2: This level is composed of a single layer:

This layer is global real-time and graphical demonstration. It is based on the middleware service layer, the low-level abstraction layer and the basic peripheral usage applications for board-based functionalities.

3 X-CUBE-NFC3 software expansion for STM32Cube

3.1 Overview

X-CUBE-NFC3 is a software package that expands the functionality provided by STM32Cube.

The key features of the package are:

- Complete middleware to build applications using the ST25R95 or CR95HF near-field communication transceivers.
- Easy portability across different MCU families, thanks to STM32Cube.
- Sample application example to detect NFC tags of different class.
- Free user-friendly license terms.
- Examples implementation available on board X-NUCLEO-NFC03A1 plugged on top of one NUCLEO-L476RG, NUCLEO-F401RE or NUCLEO-F103RB.

This software, running on STM32, gathers ST25R95 or CR95HF drivers for the device. The software is built on top of STM32Cube software technology that ease portability across different STM32 microcontrollers,and comes with examples of implementation of such drivers, running on X-NUCLEO-NFC03A1 plugged into a NUCLEO-L476RG, NUCLEO-F401RE or NUCLEO-F103RB.

The sample application configures the ST25R95 or CR95HF for wakeup, followed by a polling loop for passive device detection. When a passive tag is detected, the shield signals the detected technology by lighting a corresponding LED. The demo logs all activities with ST-Link Virtual Com Port to the host system.

The supported NFC technologies in this demo are:

- NFC-A \ ISO14443A (T1T, T2T, T4TA)
- NFC-B \ ISO14443B (T4TB)
- NFC-F \ FeliCa (T3T)
- NFC-V \ ISO15693 (T5T)
- ST25TB (ISO14443-2 Type B with proprietary protocol).

3.2 Architecture

This software expansion for STM32Cube lets you develop applications using the ST25R95 or CR95HF near-field communication transceiver ICs. It is based on the STM32CubeHAL hardware abstraction layer for the STM32 microcontroller and extends STM32Cube with a board support package (BSP) for the X-NUCLEO-NFC03A1 expansion board.

Application software can access and use the X-NUCLEO-NFC03A1 expansion board through the following layers:

- **STM32Cube HAL layer**: provides a simple set of generic, multiinstance APIs (application programming interfaces) to interact with the upper layers (application, libraries and stacks). These generic and extension APIs are directly built on a common architecture and allow overlying layers like middleware to implement their functions without depending on specific microcontroller unit (MCU) hardware information. This structure improves the library code reusability and guarantees easy portability across other devices.
- **Board support package (BSP) layer**: provides support for the peripherals on the STM32 Nucleo board (apart from the MCU). This set of APIs provides a programming interface for certain board-specific peripherals like the LED, the user button etc. This interface also helps you identify the specific board version.
- **Middleware NDEF layer**: provides several functions required for NDEF message management. The NDEF layer is compliant with Motor Industry Software Reliability Association (MISRA) C 2012. It currently supports the following NFC technologies:
  - T2T
  - T3T
  - T4AT
  - T4BT
  - T5T

The NDEF library design is split into an RF technology-dependent layer and an RF technology-independent layer:

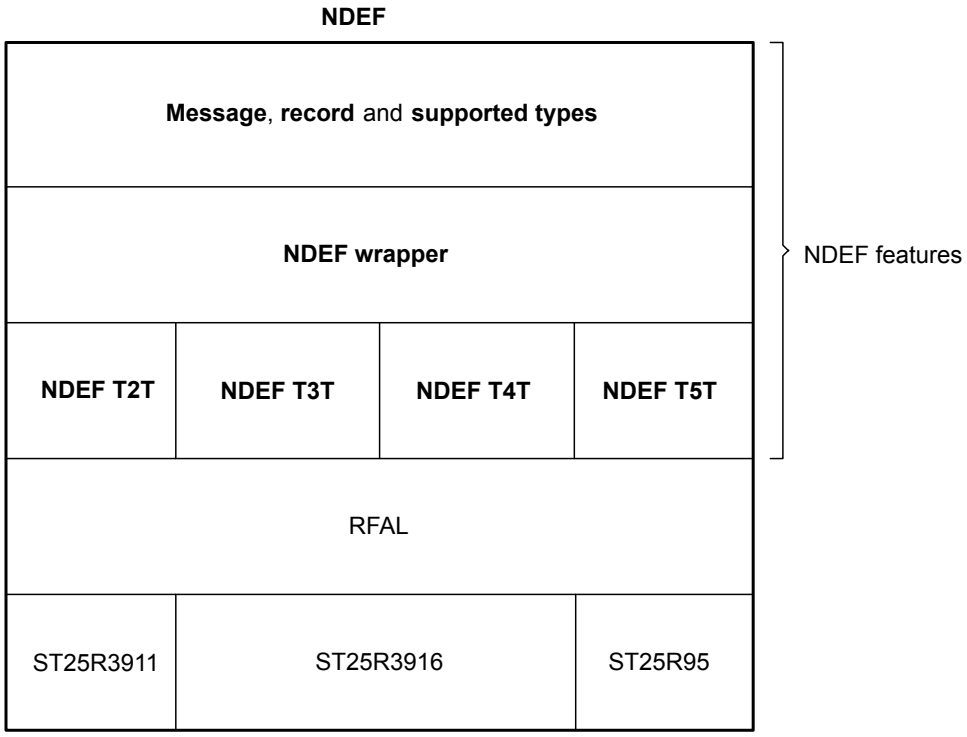
- message, record and supported type management layer (technology independent)
- NDEF technology layer defining a common API on top of the RFAL (technology dependent)
- NDEF wrapper layer abstracting the underlying technologies.

The NDEF wrapper on top of the NDEF technology-dependent components allows managing NDEF tags without taking care of the underlying NFC technologies.

The types currently supported are:

- RTD device information
- RTD text
- RTD URI
- Android application record (AAR)
- vCard
- Wi-Fi.

Figure 2. NDEF block diagram

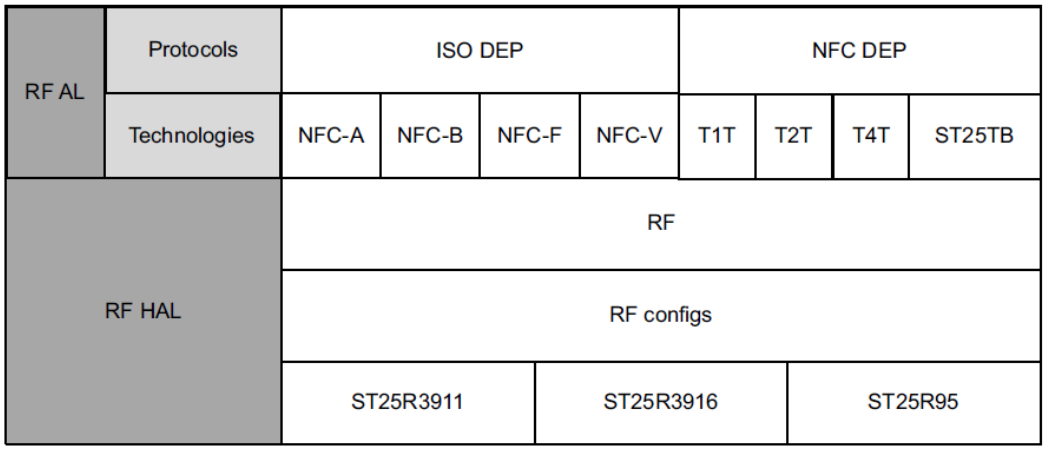


- **Middleware RF abstraction layer (RFAL)**: RFAL provides several functions for RF/NFC communication. It groups the different RF ICs (for instance ST25R95/CR95HF or ST25R3911) under a common and easy to use interface. The technologies currently supported by RFAL are:
  - NFC-A \ ISO14443A (T1T, T2T, T4TA)
  - NFC-B \ ISO14443B (T4TB)
  - NFC-F \ FeliCa (T3T)
  - NFC-V \ ISO15693 (T5T)
  - ST25TB (ISO14443-2 Type B with Proprietary Protocol).

The RFAL provides support of Data Exchange Protocols. Internally, the RFAL is divided into two sub layers:

- RF HAL- RF hardware abstraction layer
- RF AL - RF abstraction layer.

Figure 3. **RFAL block diagram**



The modules in the RF HAL are chip-dependent, they implement the RF IC driver, configuration tables and specific instructions for the HW to perform the physical RF functions. The interface for the caller is a shared RF header file which provides the same interface for upper layers (for all chips).

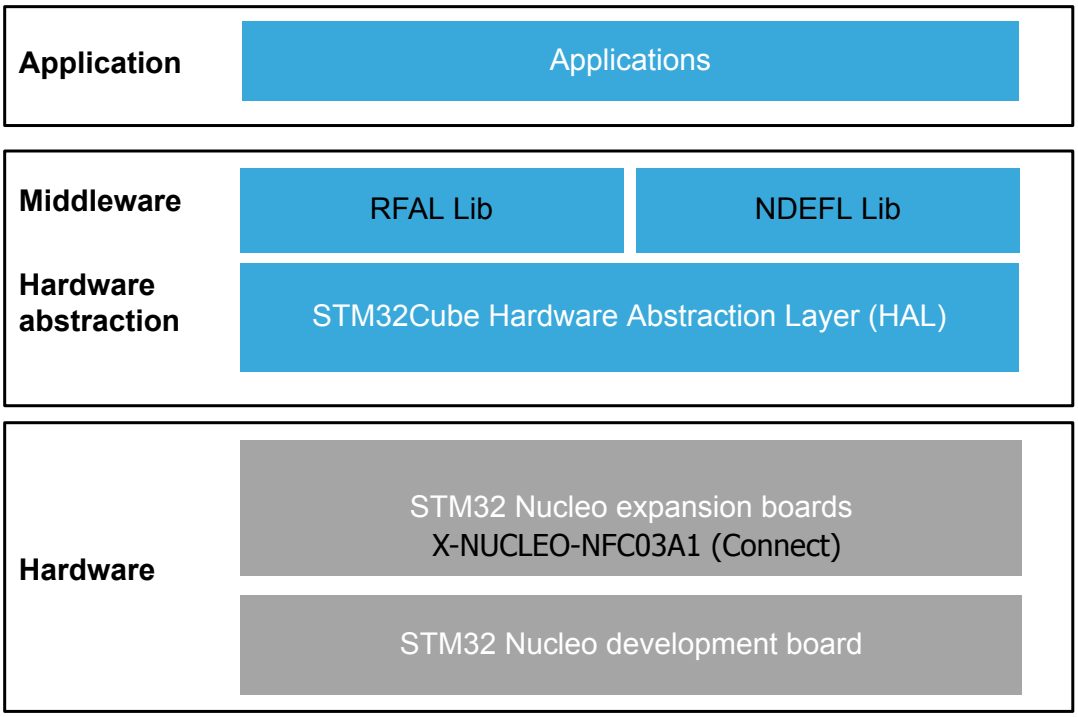
The RF AL can be broken down into two further sub layers:

- Technologies: technology modules which implement all the specifics, framing, timings, and so on
- Protocols: protocol implementation including all the framing, timings, error handling, and so on.

As well as these, the application layer uses RFAL functions such as NFC Forum Activities, and so on.

Access to the lowest functions of the ICs is granted by the RF module. The caller can make direct use of any of the RF technology or protocol layers without requiring any specific hardware configuration data.

Figure 4. **X-CUBE-NFC3 software architecture**

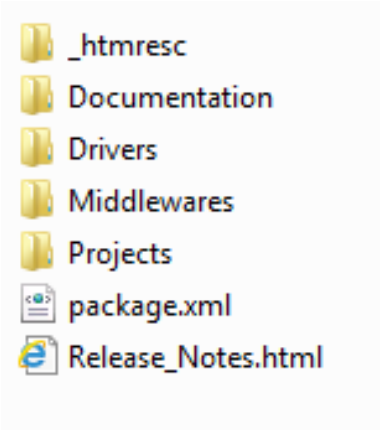


3.3 Folder structure

The following folders are included in the software package (see Figure 5. X-CUBE-NFC3 package folder structure):

- **Documentation**: this folder contains a compiled HTML file generated from the source code, which details the software components and APIs.
- **Drivers**: this folder contains the HAL drivers, the board-specific drivers for each supported board or hardware platform, including the on-board components, and the CMSIS vendor-independent hardware abstraction layer for the processor series.
- **Middlewares**: this folder contains RFAL (RF abstraction layer). RFAL provides several functions required to perform RF/NFC communication.

Figure 5. X-CUBE-NFC3 package folder structure



The RFAL groups the different ST25R RF ICs under a common and easy to use interface.

- **Projects**: this folder contains a sample application example Tag detect, provided for the NUCLEO-L476RG, NUCLEO-F103RB, and NUCLEO-F401RE platforms with three development environments (IAR™ embedded workbench for Arm® Keil® microcontroller development kit (MDK-ARM™), and the integrated development environment for STM32 (STM32CubeIDE)).

An RFAL usage example as a Poller device is provided in *exampleRfalPoller.c*. In this example, different devices are detected and activated, and data is exchanged implementing a presence check mechanism. Once removed or upon an error, the device is deactivated and the discovery loop restarts.

3.4 APIs

Detailed technical information about the APIs available to the user can be found in a compiled CHM files located inside the *rfal\doc* and *ndef\doc* folders of the software package where all the functions and parameters are fully described.

3.5 Sample applications

Two sample applications using the X-NUCLEO-NFC0531 expansion board with the NUCLEO-L476RG, NUCLEO-F401RE or NUCLEO-F103RB development board are provided in the *Projects* directory. For the MDK-ARM development kit, the project file contains the 2 applications (one target per application).

In the first application, NFC tags of different types are detected by the ST25R95 or CR95HF near field communication transceivers (see the CHM documentation file generated from the source code for more details regarding the sample application).

In the second sample application, the ST25R95 or CR95HF waits for an NFC to be detected. By default, it reads its content. The user can press the blue user button to cycle among the different features:

- Write a text record
- Write a URI record and an Android application record (AAR)

- Format an ST tag
- NDEF records are read and decoded, and their content is displayed and tuned to their type, as well as stored in a message and written to the tag.

Figure 6. NDEF sample application: write text

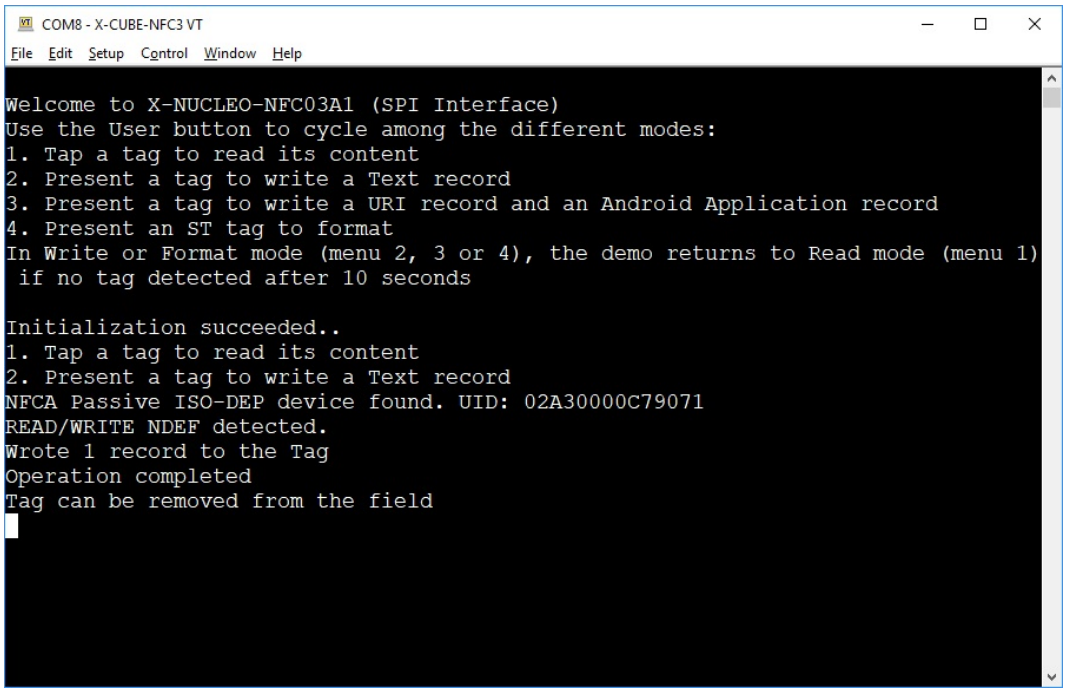


Figure 7. NDEF sample application: read text

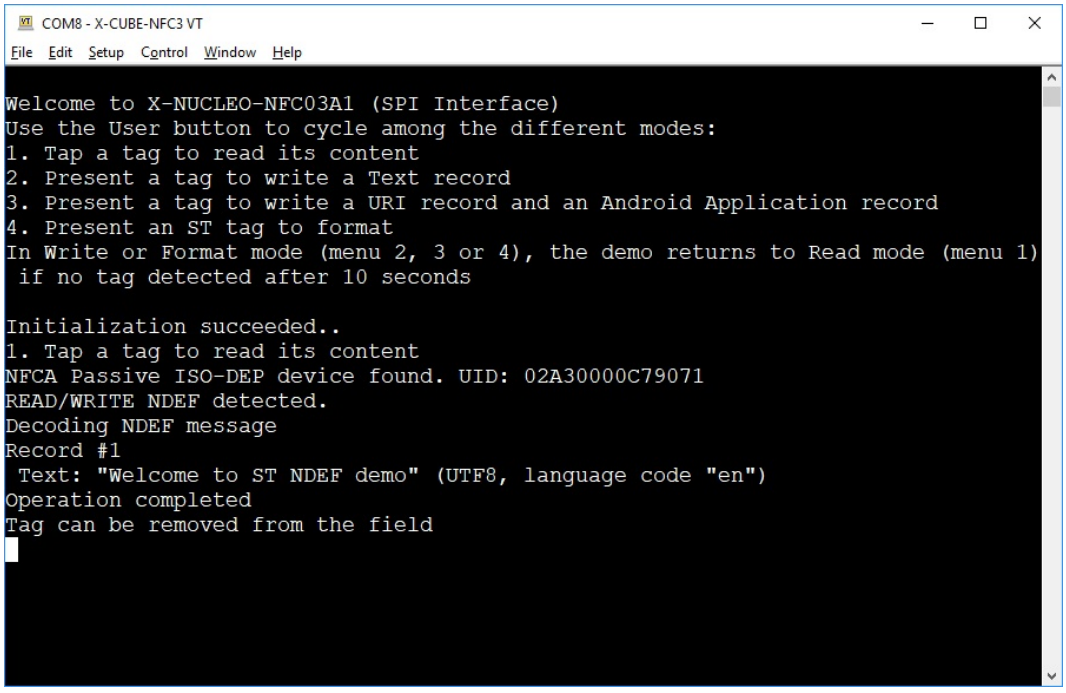


Figure 8. **NDEF sample application: write a URI record and an AAR**

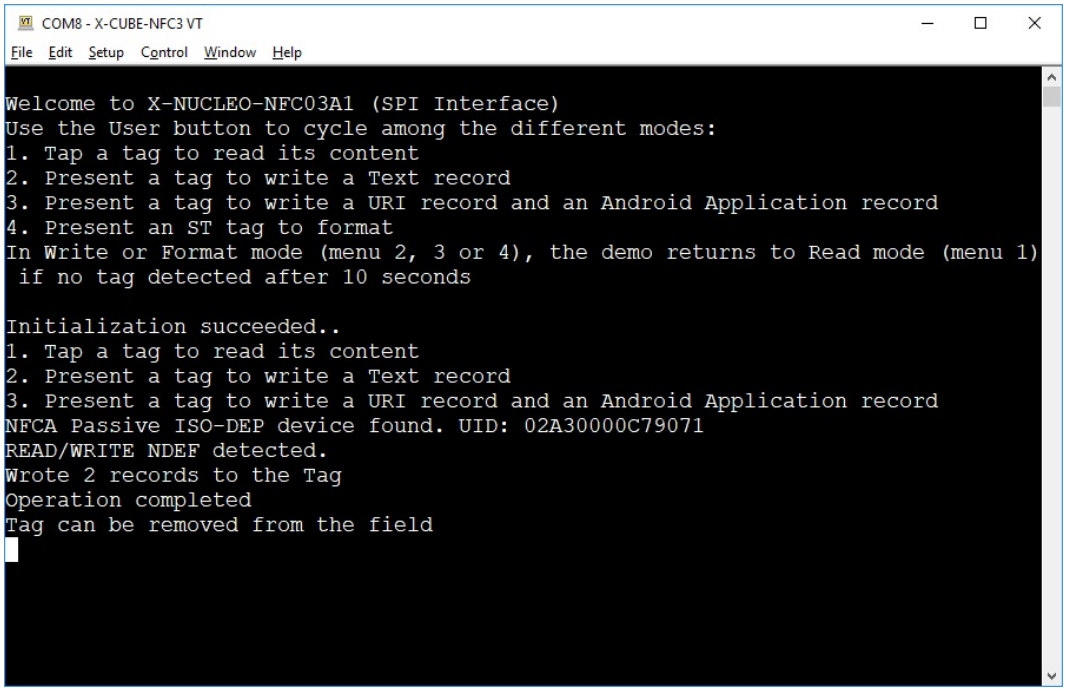


Figure 9. **NDEF sample application: read a URI record and an AAR**

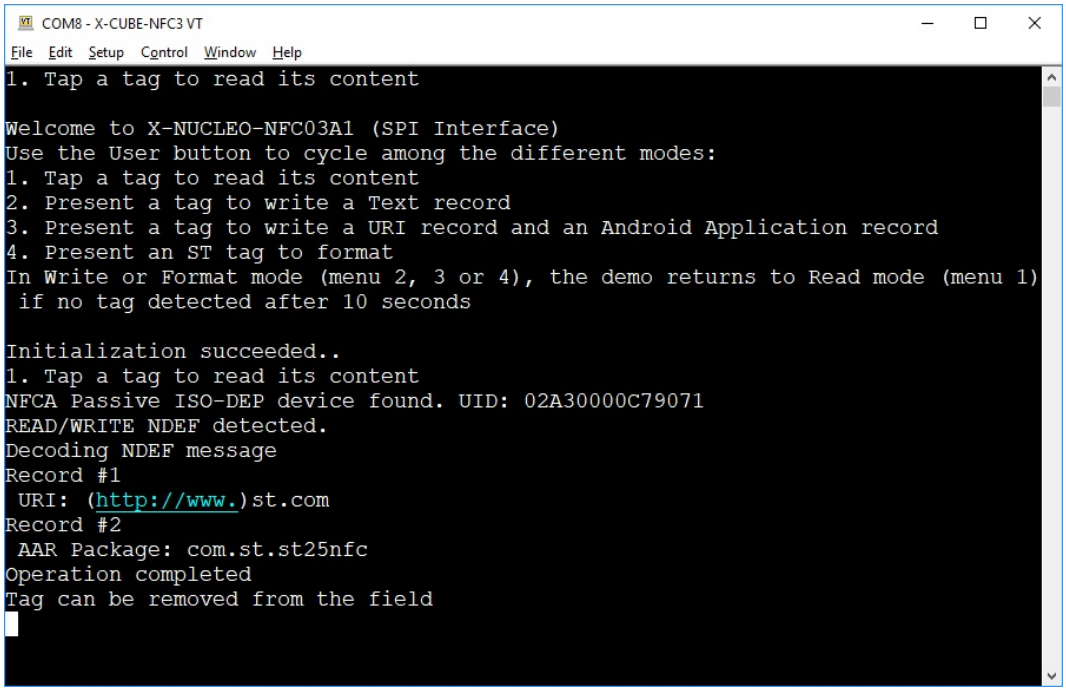


Figure 10. **NDEF sample application: format tag**

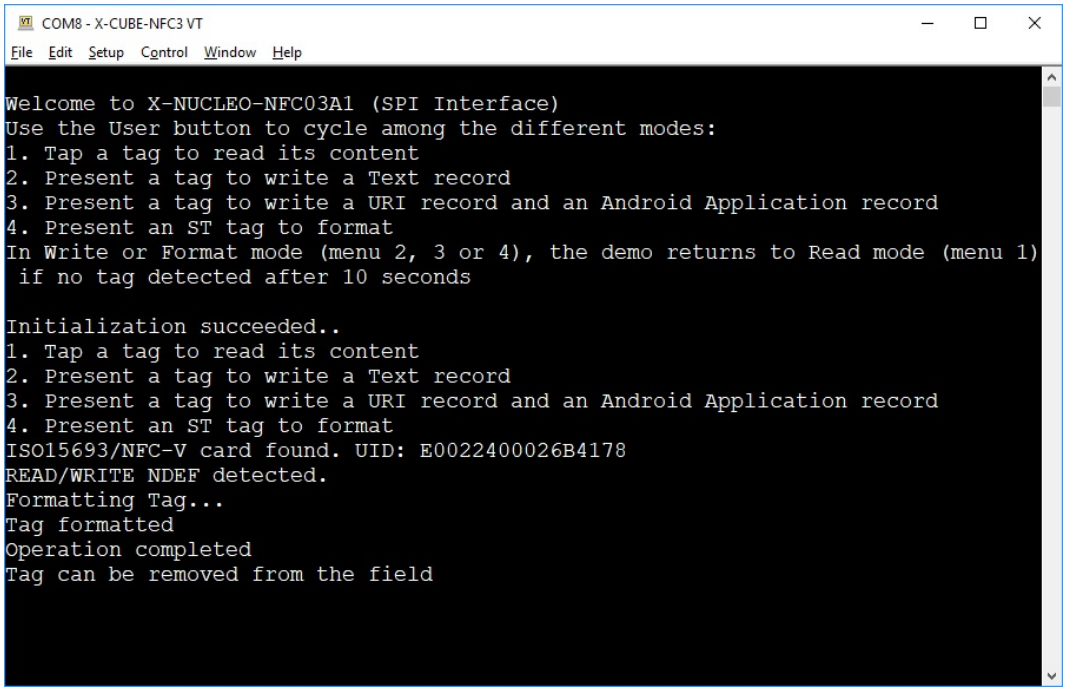
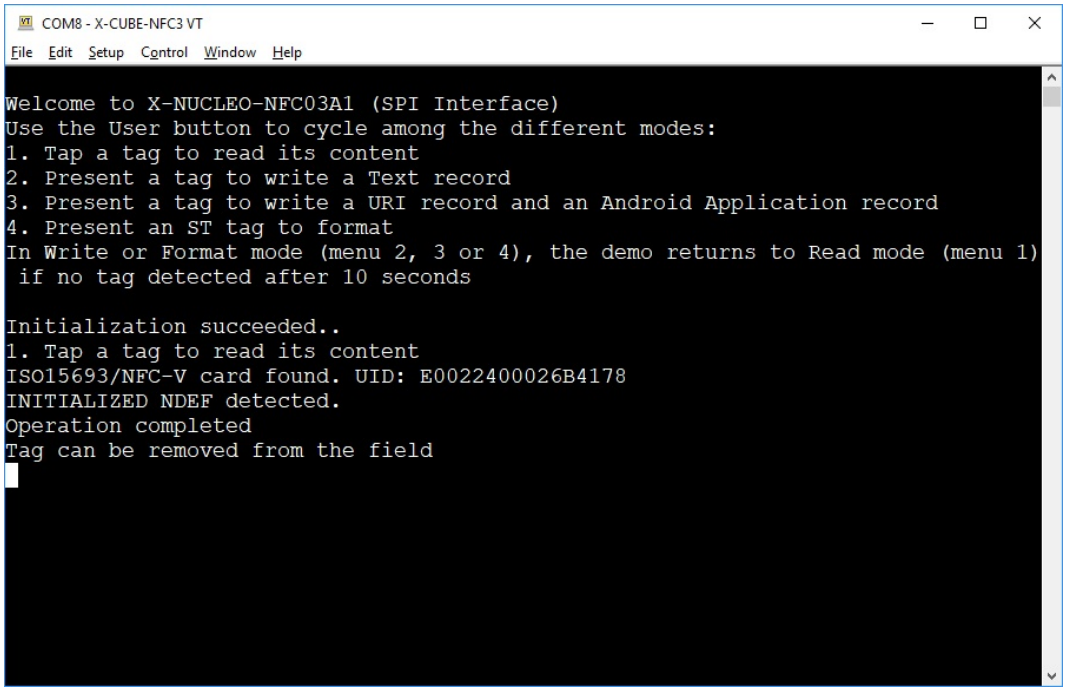


Figure 11. **NDEF sample application: read formatted tag**



After system initialization and clock configuration, LED1, LED2, LED3 and LED4 blink 3 times.

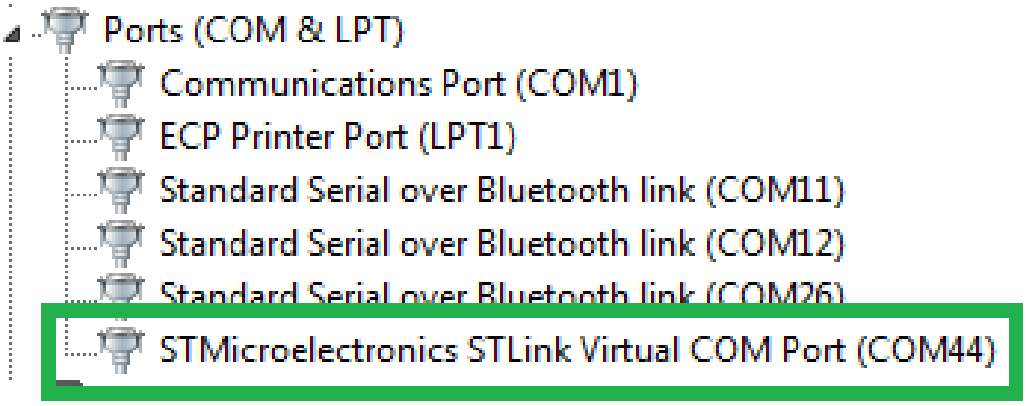
When a tag is detected in the vicinity, a LED is lit on the NFC3 shield according to Table 2. LED lit on tag detection.

Table 2. **LED lit on tag detection**

NFC Tag Type	LEDs lit on tag detection
NFC TYPE A	LED1
NFC TYPE B	LED2
NFC TYPE F	LED3
NFC TYPE V	LED4

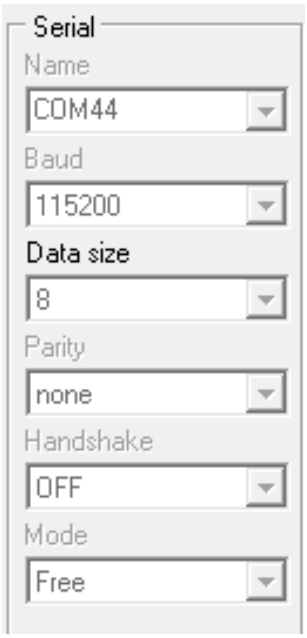
ST's virtual comport interface is also included: following system initialization, the board is configured and enumerated as an ST Virtual comport.

Figure 12. **ST virtual communication port enumeration**



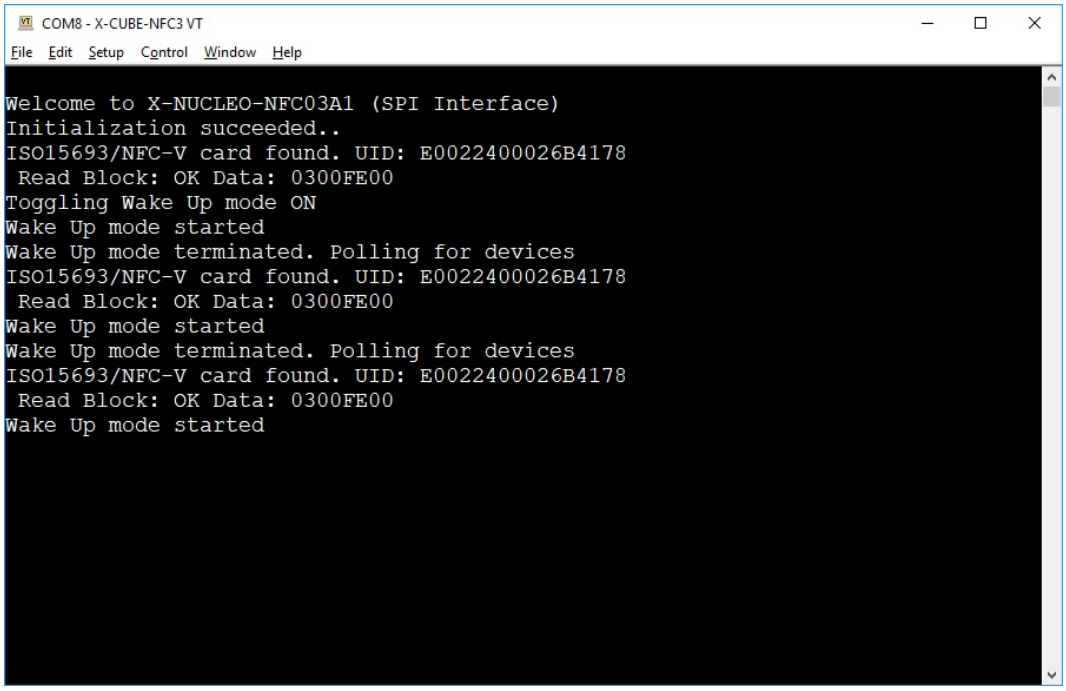
After checking the virtual COM port number, open a connection on Hyperterminal (or similar) with the configuration shown below (enable option: Implicit CR on LF, if available).

Figure 13. **UART serial communication configuration**



Following successful connection, the user can view the messages on the hyperterminal, as shown in Figure 14.

Figure 14. UART serial communication displayed on Hyperterminal



4 System setup guide

4.1 Hardware description

4.1.1 STM32 Nucleo platform

The STM32 Nucleo boards provide an affordable and flexible way for users to test new ideas and build prototypes with any STM32 microcontroller lines.

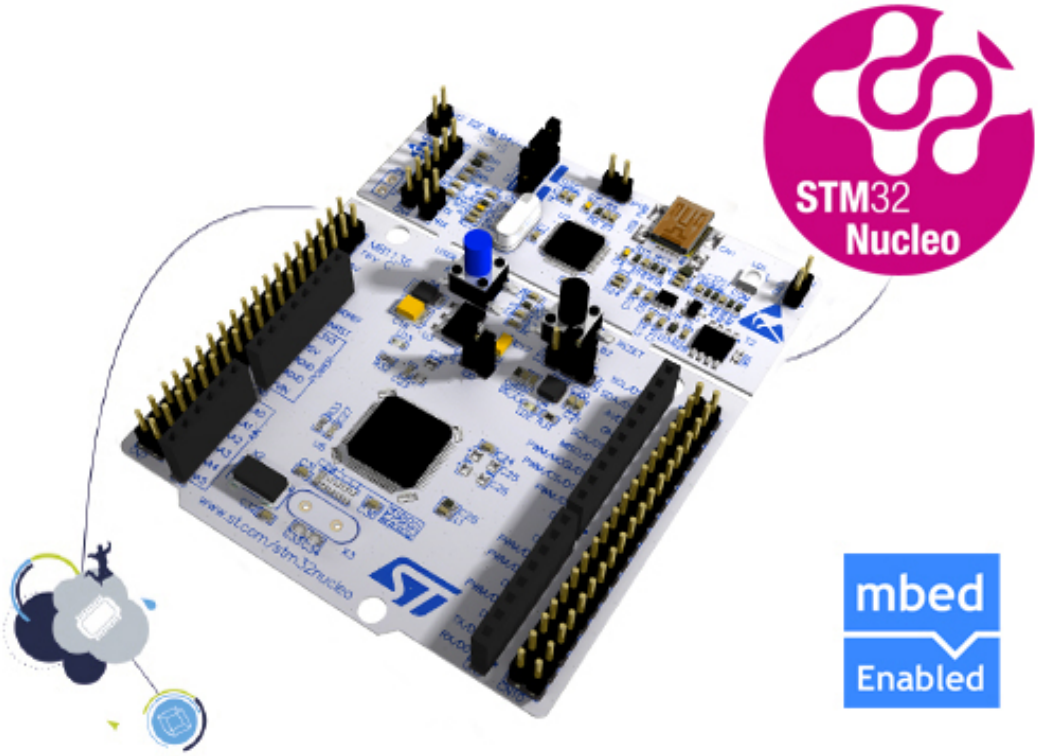
The ARDUINO® connectivity support and ST morpho headers make it easy to expand the functionality of the STM32 Nucleo open development platform with a wide choice of specialized expansion boards.

The STM32 Nucleo board does not require any separate probe, as it integrates the ST-LINK/V2-1 debugger/ programmer.

The STM32 Nucleo board comes with the STM32 comprehensive software HAL library together with various packaged software examples.

Information about the STM32 Nucleo boards is available on STMicroelectronics website www.st.com.

Figure 15. STM32 Nucleo board



4.1.2 X-NUCLEO-NFC03A1 expansion board

The X-NUCLEO-NFC03A1 is a contactless transceiver IC expansion board that can be used with the STM32 Nucleo platform. It is also compatible with the Arduino UNO R3 connector layout, and is designed around the STMicroelectronics ST25R95 or CR95HF near field communication transceiver. The X-NUCLEO-NFC03A1 interfaces with the STM32 MCU via the SPI/UART pin.

Figure 16. X-NUCLEO-NFC03A1 expansion board

Information about the STM32 Nucleo boards is available on STMicroelectronics website www.st.com.

4.2 Hardware and software setup

This section describes the hardware and software setup procedures. It also describes the required system setup.

4.2.1 Hardware setup

The following hardware components are needed:

- One STM32 Nucleo Development platform (order code: NUCLEO-L476RG, NUCLEO-F401RE or NUCLEO-F103RB)
- One ST25R95 (CR95HF) near field communication transceiver expansion board (order code: X-NUCLEO-NFC03A1)
- One USB type A to Mini-B USB cable to connect the STM32 Nucleo to the PC.

4.2.2 Software setup

This section lists the minimum requirements for the developer to setup the SDK.

Development tool-chains and compilers

Select one of the integrated development environments supported by the STM32Cube expansion software.

Read the system requirements and setup information provided by the selected IDE provider.

4.2.3 System setup guide

This section describes how to setup different hardware parts before writing and executing an application on the STM32 Nucleo board with the sensor expansion board.

STM32 Nucleo, ST25R95 (CR95HF) expansion boards setup

The STM32 Nucleo board integrates the ST-LINK/V2-1 debugger/programmer.

The developer can download the ST-LINK/V2-1 USB driver by looking at the STSW-LINK009 software on www.st.com.

The X-NUCLEO-NFC03A1 expansion board can be easily connected to the STM32 Nucleo motherboard through the Arduino UNO R3 extension connector. It is capable of interfacing with the external STM32 microcontroller on STM32 Nucleo board using SPI/UART transport layer. By default it runs using the SPI interface. The user can choose to compile with one of the two interfaces by selecting the right target under the project properties.

4.3 Software description

The following software components are needed in order to setup the suitable development environment to create applications for the STM32 Nucleo with the NFC expansion board:

- X-CUBE-NFC3: an expansion for STM32Cube dedicated to NFC applications development. The X-CUBE-NFC3 firmware and its related documentation are available on www.st.com.
- Development tool-chain and Compiler: the STM32Cube expansion software supports the three following environments:
  - IAR embedded workbench for Arm (EWARM) toolchain + ST-LINK
  - Keil microcontroller development kit (MDK-ARM) toolchain + ST-LINK
  - Integrated development environment for STM32 (STM32CubeIDE) + ST-LINK.

Revision history

Table 3. **Document revision history**

Date	Version	Changes
26-May-2016	1	Initial release.
15-Jan-2019	2	Added ST25R95. Updated: • Introduction • Table 1: List of acronyms • Section 4.1: Overview • Section 4.2: Architecture • Section 4.3: Folders structure • Section 4.4: APIs • Section 4.5: Sample application description • Section 5.3.1: Hardware setup
27-Jan-2020	3	Updated: • Introduction • STM32Cube overview • Section 3.2 Architecture • Section 3.4 APIs • Section 3.5 Sample applications.

Contents

List of tables

List of figures