AN4977
Application note

ST25DV applications based on STM32Cube

Introduction

This document describes in detail the STSW-ST25DV001 firmware, developed to run demonstrations based on ST25DV, a dynamic NFC tag with two different interfaces:

- the NFC RF interface, compliant with NFC Forum Type 5 Tag and ISO15693 standards
- an I2C interface, controlled by a MCU (executing this firmware)

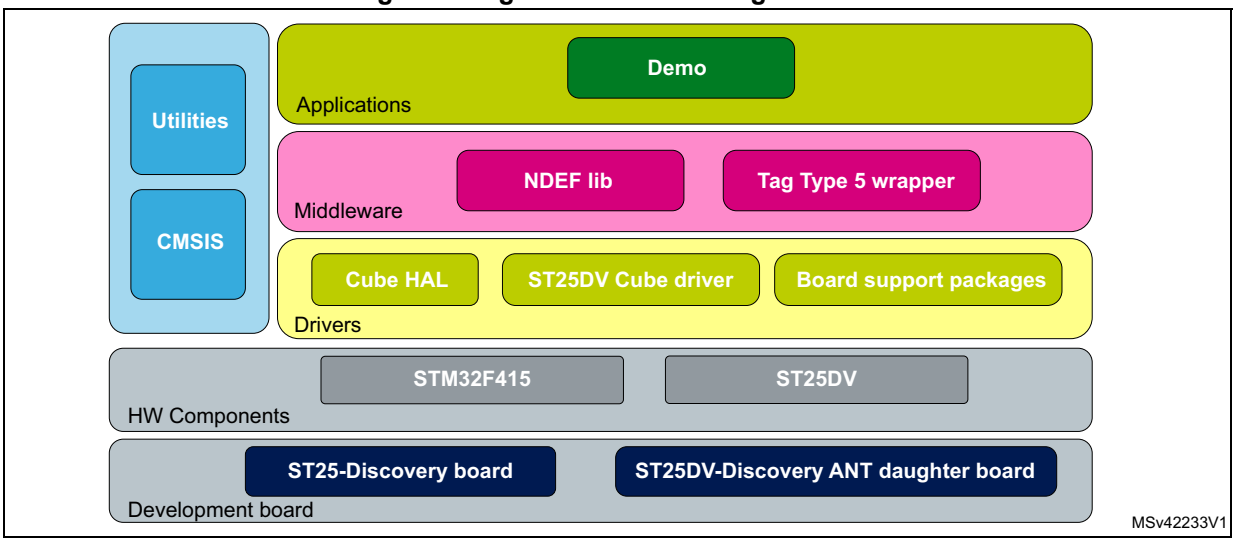
The ST25DV also supports a Fast Transfer mode to enable faster communication between the MCU and the RF world.

The firmware is designed to run on the STM32F415 microcontroller embedded on the ST25-DISCOVERY board, described in UM2062, available on *www.st.com*. Thanks to the STM32Cube methodology, the structure of the firmware enables a full porting to other STM32 microcontrollers, as well as an easy reuse of parts of it.

In particular, two important elements are designed to ease their reuse:

- The ST25DV driver, which implements the functions to control the ST25DV through the I2C interface, is completely independent from the MCU: it can thus be easily reused in any other project based on the STM32Cube methodology, and in any other HW solution interacting with a ST25DV.
- The NDEF library, which implements the standard NDEF protocol, is provided as a STM32Cube Middleware, fully independent from the HW.

Figure 1. High level firmware organization



Contents

List of tables

List of figures

1 List of acronyms and notational conventions

1.1 Acronyms

AAR: Android Application Record

API: Application Program Interface

BSP: Board Support Package

GPIO: General Purpose I/O (Input/Output)

NDEF: NFC Data Exchange Format

NFC: Near Field Communication

OOB: Out OF Band

URI: Uniform Resource Identifier

URL: Uniform Resource Locator

1.2 Representation of numbers

The following conventions and notations apply in this document unless otherwise stated:

- **Binary numbers** are represented by strings of 0 and 1 digits shown with the most significant bit (MSB) on the left, the least significant bit (LSB) on the right, and "0b" added at the beginning. Example: 0b11110101.
- **Hexadecimal numbers** are represented by using numbers 0 to 9 and characters A to F, and adding "0x" at the beginning. The Most Significant Byte (MSB) is shown on the left and the Least Significant Byte (LSB) on the right. Example: 0xF5.
- **Decimal numbers** are represented without any trailing character. Example: 245.

2 HW resources

The firmware runs on the ST25DV-DISCOVERY kit, which includes two different boards:

- the ST25-DISCOVERY motherboard, embedding a STM32F415 MCU
- the ST25DV-Ant board, including the ST25DV and an NFC antenna

2.1 ST25-DISCOVERY board

The ST25-DISCOVERY board is built around a STM32F415 MCU (running this firmware).

The board also includes

- an LCD display with a touchscreen
- two USB connectors, one for the ST-Link, the second available for any user application
- a connector for a Tag Antenna daughter board (such as one of the ST25DV antenna boards)

Available options are

- an ST Bluetooth Low Energy module
- an ST Wi-Fi module

2.2 ST25DV antenna boards

These boards are built around the ST25DV dynamic NFC tag and different NFC antennas.

3 Overview

The firmware implements three demonstration categories:

- Fast Transfer Mode demonstrations, with different use cases benefiting from the faster communication between the MCU and the reader:
  - FW upgrade.
  - Picture download/upload.
  - Data transfers (from or to the reader).
- NDEF messages demonstrations:
  - NFC well-known types: URL, phone number, SMS, Email,...
  - vCard
  - Android Application Record
  - Proprietary record (MyAPP)
  - Pairing with Out Of Band records for BLE and Wi-Fi
- Demonstrations of other ST25DV features:
  - RF interrupt through a dedicated GPO
  - Energy harvesting from the RF to power an additional device
  - Different states: RF off, RF sleep and Low power down
  - Memory mapping and password protection

The firmware also implements a user interface consisting in:

- a menu to select the demonstration
- several screens to display demonstration instructions and status.

4 Firmware structure

The structure of this document follows the firmware structure, using the comment lines embedded in the source code. The available modules are listed in *Table 1*, and are described in detail in the following sections.

Table 1. STSW-ST25DV001 firmware structure

Group	Module	Description
Main	ST25 Discovery demonstration	Main module for all the ST25Discovery board demonstrations
	Fast Transfer Mode demonstration	Provides functions to manage the protocol for the Fast Transfer Mode demonstrations
	NDEF demonstration	Implements the NDEF demonstration functions
	ST25DV Features demonstration	Implements the functions to execute the demonstrations of the ST25DV specific features
Fast Transfer Mode	ST25DV Mailbox functions	Common APIs for the ST25DV Mailbox
	Flash Command	Implements high level functions to write firmware or data to the internal Flash memory
	Flash memory API	Defines an API to access the internal Flash memory
ST25DV management related module	ST25DV common functions	Proposes a generic API for the ST25DV
ST25DV board support package and driver	Board Support Package	Container module for the BSP modules
	ST25 Discovery NFCTAG Board Support Package	Provides high-level functions to access the ST25DV
	ST25DV driver	Implements the functions to drive the ST25DV
Menus	ST25DV menu definition	This module defines the structure and the content of the ST25DV demonstration menu
	ST25DV menu interface configuration	Interface file for the menu demonstration middleware
	LibJPEG decode wrapper	Wrapper calling the libJPEG STM32Cube middleware to decode the JPEG pictures
MCU support modules	ST25 Discovery interrupt routines	Defines all the required interrupt routines for the ST25DV demonstration
	ST25 Discovery MCU support package	Defines the MCU init routines for some of the ST25 Discovery peripherals

4.1 Middlewares used in this firmware

This firmware relies on several middlewares, either provided by ST or by a third party.

The middlewares are HW-independent softwares implementing a generic feature, they will not be detailed in this document. Those used in this firmware are listed in *Table 2*.

Table 2. Middlewares used by the STSW-ST25DV001 firmware

Middleware	Description
LibNDEF	This library provides functions to read and write NDEF messages to a tag. It supports a variety of NDEF records, such as: – URI record (includes URLs) – SMS record – Email record – vCard record – Geolocation record – Bluetooth OOB record – Wi-Fi OOB record – Android Application Record The library also defines a NFC-Forum Type5 Tag wrapper to comply with the NFC-Forum Type5 Tag specification.
LibJPEG	This library implements the JPEG codec. This firmware only includes the JPEG decoding part of the library.
STM32 BlueNRG	Provides the communication stack for the ST BlueNRG module (Bluetooth Low Energy). An HID profile is used on top of it, to remotely control a mouse pointer on a paired device.
STM32 SPWF01SA	Provides the communication stack for the ST SPWF01 Wi-Fi module. It is used to set the Wi-Fi Module as a mini Access Point, to receive connections from remote devices.
Menu Demo	Implements functions to display an icon-and-text-based menu. It also manages inputs from a touchscreen, a joystick and a button, to interact with the user.

4.2 ST25DV board support package

The Board Support Package software (BSP) is defined by the STM32Cube methodology as the abstraction layer for the board specific features.

It implements all the functions required to access:

- the components on the board.
- the MCU peripherals requiring a board specific configuration.

The different parts of the BSP are described below.

4.2.1 IOBus

This part of the BSP implements the low level functions interfacing between the components drivers and the MCU peripherals (by calling the STM32Cube HAL).

The BSP IOBus functions are not detailed in this document.

4.2.2 High level APIs

This parts of the BSP provides high level functions called by the application or middlewares to access the component drivers.

This layer acts as a bridge between the application and the component drivers.

In this document, only the NFCTAG BSP is detailed.

4.3 Components

The ST25DV Discovery kit embeds different components requiring specific softwares to be correctly driven. Unless specified, these drivers are not detailed in this document.

Table 3. Drivers used by the STSW-ST25DV001 firmware

Driver	Description
ST25DV	Provides all the required functions to access the ST25DV NFC dynamic tag. This driver is covered in *Section 5.11: ST25DV driver on page 39*.
ILI9341	Implements functions to access the LCD display of the ST25 Discovery.
STMPE811	Implements functions to access the touchscreen of the ST25 Discovery.
AD5161	Implements functions to access the digital potentiometer of the ST25 Discovery

4.4 Index of files

Table 4. Files used by the STSW-ST25DV001 firmware

File	Description
ST25DVDemo/Drivers/BSP/Components/ST25DV/st25dv.c	Provides a set of driver functions to manage communication between BSP and ST25DV chip.
ST25DVDemo/Drivers/BSP/Components/ST25DV/st25dv.h	Provides a set of driver functions to manage communication.
ST25DVDemo/Drivers/BSP/ST25-Discovery/ st25_discovery_nfctag.c	Provides a set of functions needed to manage an NFC dual interface EEPROM.
ST25DVDemo/Drivers/BSP/ST25-Discovery/ st25_discovery_nfctag.h	Contains definitions for the x_nucleo_nfc04a1_nfctag.c specific functions.
ST25DVDemo/Projects/ST25DV-Discovery/ Demonstrations/ST25DVDemo/Inc/bluenrg_interface.h	Provides the code for the BlueNRG Expansion Board driver based on STM32Cube HAL for STM32 Nucleo board.s
ST25DVDemo/Projects/ST25DV-Discovery/Demonstrations/ST25DVDemo/Inc/common.h	Header for common.c module.
ST25DVDemo/Projects/ST25DV-Discovery/Demonstrations/ST25DVDemo/Inc/commonfunc.h	Header for commonfunc.c module.

Table 4. Files used by the STSW-ST25DV001 firmware (continued)

File	Description
ST25DVDemo/Projects/ST25DV-Discovery/Demonstrations/ST25DVDemo/Inc/ff.h	Simple file system API header.
ST25DVDemo/Projects/ST25DV-Discovery/Demonstrations/ST25DVDemo/Inc/flash_if.h	Header file for flash_if.c.
ST25DVDemo/Projects/ST25DV-Discovery/Demonstrations/ST25DVDemo/Inc/fw_command.h	Header file for fw_command.c.
ST25DVDemo/Projects/ST25DV-Discovery/Demonstrations/ST25DVDemo/Inc/jconfig.h	Header configuration for jpeg module.
ST25DVDemo/Projects/ST25DV-Discovery/Demonstrations/ST25DVDemo/Inc/jmorecfg.h	Header file for jpeg module.
ST25DVDemo/Projects/ST25DV-Discovery/Demonstrations/ST25DVDemo/Inc/jpeg_decode.h	Header for jpeg_decode.c module.
ST25DVDemo/Projects/ST25DV-Discovery/Demonstrations/ST25DVDemo/Inc/mailbox.h	Header for mailbox.c module.
ST25DVDemo/Projects/ST25DV-Discovery/Demonstrations/ST25DVDemo/Inc/mailboxfunc.h	Header for mailboxfunc.c module.
ST25DVDemo/Projects/ST25DV-Discovery/Demonstrations/ST25DVDemo/Inc/main.h	Header for main.c module.
ST25DVDemo/Projects/ST25DV-Discovery/Demonstrations/ST25DVDemo/Inc/Menu_definition.h	Header for Menu_definition.c.
ST25DVDemo/Projects/ST25DV-Discovery/Demonstrations/ST25DVDemo/Inc/mxconstants.h	Contains the common defines of the application.
ST25DVDemo/Projects/ST25DV-Discovery/Demonstrations/ST25DVDemo/Inc/ndef_demo.h	Defines the API for the NDEF demonstration.
ST25DVDemo/Projects/ST25DV-Discovery/Demonstrations/ST25DVDemo/Inc/st25dv_features_demo.h	Defines the API for the ST25DV features demonstration.
ST25DVDemo/Projects/ST25DV-Discovery/Demonstrations/ST25DVDemo/Inc/stm32f4xx_hal_conf.h	HAL configuration file.
ST25DVDemo/Projects/ST25DV-Discovery/Demonstrations/ST25DVDemo/Inc/stm32f4xx_it.h	This file contains the headers of the interrupt handlers.
ST25DVDemo/Projects/ST25DV-Discovery/Demonstrations/ST25DVDemo/Inc/usbd_conf.h	General low level driver configuration.
ST25DVDemo/Projects/ST25DV-Discovery/Demonstrations/ST25DVDemo/Inc/usbd_desc.h	Header for usbd_desc.c module.
ST25DVDemo/Projects/ST25DV-Discovery/Demonstrations/ST25DVDemo/Inc/usbd_storage.h	Header for usbd_storage.c module.
ST25DVDemo/Projects/ST25DV-Discovery/Demonstrations/ST25DVDemo/Inc/version.h	Header for firmware version number.
ST25DVDemo/Projects/ST25DV-Discovery/Demonstrations/ST25DVDemo/Src/commonfunc.c	Common functions for the ST25DV management.
ST25DVDemo/Projects/ST25DV-Discovery/Demonstrations/ST25DVDemo/Src/flash_if.c	This file provides all the Flash memory layer functions.

Table 4. Files used by the STSW-ST25DV001 firmware (continued)

File	Description
ST25DVDemo/Projects/ST25DV-Discovery/Demonstrations/ST25DVDemo/Src/fs_api.c	Simple files system API implementation.
ST25DVDemo/Projects/ST25DV-Discovery/Demonstrations/ST25DVDemo/Src/fw_command.c	This file provides all the Flash memory programming command functions.
ST25DVDemo/Projects/ST25DV-Discovery/Demonstrations/ST25DVDemo/Src/jpeg_decode.c	File containing the JPEG decompressing methods.
ST25DVDemo/Projects/ST25DV-Discovery/Demonstrations/ST25DVDemo/Src/mailbox.c	Mailbox functions to illustrate the Fast Transfer Mode feature.
ST25DVDemo/Projects/ST25DV-Discovery/Demonstrations/ST25DVDemo/Src/mailboxfunc.c	Common Mailbox functions used for Fast Transparent Mode protocol.
ST25DVDemo/Projects/ST25DV-Discovery/Demonstrations/ST25DVDemo/Src/main.c	Main program body.
ST25DVDemo/Projects/ST25DV-Discovery/Demonstrations/ST25DVDemo/Src/Menu_config.c	Menu configuration file.
ST25DVDemo/Projects/ST25DV-Discovery/Demonstrations/ST25DVDemo/Src/Menu_definition.c	Defines the content of the menu for the ST25DV demonstration.
ST25DVDemo/Projects/ST25DV-Discovery/Demonstrations/ST25DVDemo/Src/ndef_demo.c	This file provides functions to execute NDEF demonstrations.
ST25DVDemo/Projects/ST25DV-Discovery/Demonstrations/ST25DVDemo/Src/st25dv_features_ -demo.c	This file provides functions to execute ST25DV demonstrations.
ST25DVDemo/Projects/ST25DV-Discovery/Demonstrations/ST25DVDemo/Src/stm32f4xx_hal_msp.c	This file provides code for the MSP Initialization and de-initialization.
ST25DVDemo/Projects/ST25DV-Discovery/Demonstrations/ST25DVDemo/Src/stm32f4xx_it.c	Main Interrupt Service Routines.
ST25DVDemo/Projects/ST25DV-Discovery/Demonstrations/ST25DVDemo/Src/testmenu.c	Test menu.
ST25DVDemo/Projects/ST25DV-Discovery/Demonstrations/ST25DVDemo/Src/usbd_desc.c	This file provides the USBD descriptors and string formatting method.
ST25DVDemo/Projects/ST25DV-Discovery/Demonstrations/ST25DVDemo/Src/usbd_storage.c	Memory management layer.

5 Modules

5.1 ST25 Discovery demonstration

This is the main module for all the ST25 Discovery board demonstrations.

Table 5. ST25 Discovery demonstration sub-modules

Name	Description
ST25DV common functions	Proposes a generic API used for the ST25DV.
Flash memory API	Defines an API to access the internal Flash memory.
Simple file system	Simple implementation of a file system API.
Flash Command	This module implements high level functions to write firmware or data to Flash memory.
LibJPEG decode wrapper	Wrapper calling the libJPEG Cube middleware to decode JPEG pictures.
Fast transfer mode demonstration	Provides the functions to manage the protocol for the Fast Transfer Mode demonstrations.
ST25DV Mailbox functions	Proposes common APIs for the ST25DV Mailbox.
ST25DV Menu definition	Defines the structure and the content of the ST25DV demonstration menu.
NDEF Demo	Implements the NDEF demonstration functions.
ST25DV Features Demo	Implements the functions to execute the demonstrations of the ST25DV specific features.
ST25 Discovery MCU support package	Defines the MCU init routines for some of the ST25 Discovery peripherals.
ST25 Discovery interrupt routines	Defines all the required interrupt routines for the ST25DV demonstrations.

Table 6. ST25 Discovery demonstration module - Overview

Section	Code	Description
Files	main.c	Main program body.
Functions	`int main (void)`	Demonstration entry point.
	`void SplashScreen (void)`	Displays splash screen.
	`void MenuAbout (void)`	Displays "_about_" screen.
	`tClockTime Clock_Time (void)`	Reads and returns the system tick (ms).

5.1.1 Detailed description

This module does the default initialization for the HW (system clocks, general purpose IOs, display and touchscreen) and starts the demonstration menu

Demonstrations are divided in three different sub-modules:

- Fast transfer mode demonstrations
- ST25DV features demonstrations
- NDEF demonstrations

5.1.2 Function documentation

Clock_Time()

```
tClockTime Clock_Time (
void )
```

This function reads and returns the system tick (ms).

Return value:

tClockTime: Current system tick.

5.2 Fast Transfer Mode demonstration

This module provides functions to manage the protocol for the Fast Transfer Mode demonstrations.

Table 7. Fast Transfer Mode demonstration module - Overview

Section	Code	Description
Files	mailbox.c	Mailbox functions to illustrate the Fast Transfer Mode feature
Functions	`void FTMManagement (void)`	Manages data exchange protocol with reader through Mailbox

5.2.1 Detailed description

This module provides functions to manage the protocol for the Fast Transfer Mode demonstration. The described use cases are:

- Data transfer
- Firmware upgrade
- Download pictures
- Upload pictures

5.2.2 Function documentation

FTMManagement()

```
void FTMManagement (
void )
```

Manage data exchange protocol with reader through Mailbox.

Parameters: None

Return values: None

5.3 ST25DV features demonstration

This module implements the functions to execute the demonstrations of the ST25DV specific features.

Table 8. ST25DV features demonstration module - Overview

Section	Code	Description
Files	st25dv_features_demo.c	Provides the functions to execute the ST25DV demonstrations.
Functions	`void ST25DV_DEMO_EnergyHarvesting (void)`	Runs the Energy Harvesting demonstration with a digital potentiometer set to 240 Ω.
	`void ST25DV_DEMO_GPO (void)`	Runs the GPO interrupts demonstrations.
	`void ST25DV_DEMO_RF_Off (void)`	Sets the ST25DV RF in disabled state.
	`void ST25DV_DEMO_RF_Sleep (void)`	Sets the ST25DV RF to sleep state.
	`void ST25DV_DEMO_Low_Power_Down (void)`	Sets the ST25DV in Low Power Mode.
	`void ST25DV_DEMO_Memory_Mapping_Password (void)`	Configures the ST25DV memory mapping and sets a password protection

5.3.1 Detailed description

This module implements the functions to execute the demonstrations of the ST25DV specific features.

This module covers the following use cases:

- Energy Harvesting
- GPO interrupts from RF
- ST25DV states: RF disabled, RF sleep, Low power down
- Memory map configuration and Password protection

5.3.2 Function documentation

ST25DV_DEMO_GPO()

```
void ST25DV_DEMO_GPO (
void )
```

This function runs the GPO interrupts demonstrations. This function displays the type of received GPO interrupts. It also displays the number of interrupt received

ST25DV_DEMO_Memory_Mapping_Password()

```
void ST25DV_DEMO_Memory_Mapping_Password (
void )
```

This function configures the ST25DV memory mapping and sets a password protection. It sets two memory areas:

1. ReadOnly with a vCard NDEF record.
2. NoRead, NoWrite with another vCard NDEF record.

5.4 NDEF demonstration

This module implements the NDEF demonstration functions.

Table 9. NDEF demonstration module - Overview

Section	Code	Description
Files	ndef_demo.c	Provides the functions to execute NDEF demonstrations.
Functions	`void NDEF_DEMO_Init_Tag (void)`	Initializes the NFC tag to perform the NDEF demonstrations.
	`void NDEF_DEMO_Write_URI_URL (void)`	Writes an URL to the tag.
	`void NDEF_DEMO_Write_URI_Tel (void)`	Writes a phone number to the tag.
	`void NDEF_DEMO_Read_URI (void)`	Reads and URI from the tag and displays its content.
	`void NDEF_DEMO_Read_SMS (void)`	Reads an SMS from the tag and displays it.
	`void NDEF_DEMO_Write_SMS (void)`	Writes an SMS to the tag.
	`void NDEF_DEMO_Read_Email (void)`	Reads an Email from the tag and displays its content.
	`void NDEF_DEMO_Write_Email (void)`	Writes an Email to the tag.
	`void NDEF_DEMO_Read_Vcard (void)`	Reads a vCard record from the tag and display its content.
	`void NDEF_DEMO_Write_Vcard (void)`	Writes a vCard record to the tag.
	`void NDEF_DEMO_Write_Picture_Vcard (void)`	Writes a vCards with an embedded picture to the tag.
	`void NDEF_DEMO_Read_Geo (void)`	Reads a Geolocation record from the tag and displays it content.
	`void NDEF_DEMO_Write_Geo (void)`	Writes a Geolocation record to the tag.
	`void NDEF_DEMO_Read_MyAPP (void)`	Reads a MyApp record from the tag and starts the associated demo.
	`void NDEF_DEMO_Write_AAR (void)`	Writes an AAR record (selecting the ST NFC application) to the tag.
	`void NDEF_DEMO_MultiRecord_With_AAR (void)`	Adds an AAR record (selecting the ST NFC application) to an existing NDEF file on the tag.
	`void NDEF_DEMO_Write_BLE_OOB (void)`	Writes a Bluetooth Low Energy OOB record to the tag and starts the BLE module, waiting for a connection to occur.
	`void NDEF_DEMO_Read_Bluetooth_OOB (void)`	Reads a BLE OOB record from the tag and displays its content
	`void NDEF_DEMO_Write_Wifi_OOB (void)`	Writes a Wi-Fi OOB record to the tag and starts the Wi-Fi module, waiting for a connection to occur.
	`void NDEF_DEMO_Read_Wifi_OOB (void)`	Reads a Wi-Fi OOB record from the tag and displays its content.
	`void NDEF_DEMO_Write_empty_NDEF (void)`	Writes an empty NDEF message.

Table 9. NDEF demonstration module - Overview (continued)

Section	Code	Description
Functions	`void NDEF_DEMO_Erase_CCFile (void)`	Writes 0xFF to the four first bytes in EEPROM.
	`void NDEF_DEMO_Clear_Eeprom (void)`	Writes 0xFF to the entire EEPROM.
	`void Hid_Profile_Application (void)`	Starts the HID application, using touchscreen detection to send mouse input reports and user button as mouse button.
	`void NDEF_DEMO_Write_NoPicture_Vcard (void)`	Writes a small vCard record to the tag.

5.4.1 Detailed description

This module implements the NDEF demonstration functions, covering the following use cases:

- URI NDEF records: URL & Phone
- SMS NDEF record
- Email NDEF record
- vCard NDEF record (with and without an embedded picture)
- Geolocation NDEF record
- MyApp custom NDEF record
- Multi NDEF record with AAR
- Bluetooth Low Energy OOB NDEF record
- Wi-Fi OOB NDEF record

5.4.2 Function documentation

NDEF_DEMO_Init_Tag()

```
void NDEF_DEMO_Init_Tag (
void )
```

This function initializes the NFC Tag to perform the NDEF demonstrations.

The tag is configured to its default and a CC file is written.

5.5 ST25DV Mailbox functions

This module proposes common APIs for the ST25DV Mailbox.

Table 10. ST25DV Mailbox functions module - Overview

Section	Code	Description
Files	mailbox.c	Common Mailbox functions used for Fast Transparent Mode protocol
Data structures	struct MB_HEADER_T	Mailbox global header information structure definition
Functions	`NFCTAG_StatusTypeDef InitMailBoxMode (void)`	Initializes the Mailbox mode, Enables Mailbox and disables Mailbox Watchdog
	`NFCTAG_StatusTypeDef DeInitMailBoxMode (void)`	De-Initializes the Mailbox mode, disables mailbox mode
	`NFCTAG_StatusTypeDef WriteMailBoxMsg (const uint8_t const pData, const uint16_t NbBytes)`	Writes message in Mailbox.
	`NFCTAG_StatusTypeDef ReadCompleteMailBoxMsg (uint8_t const pData, uint16_t const pLength)`	Reads entire Mailbox Message from the tag.
	`NFCTAG_StatusTypeDef ReadFragmentMailBoxMsg (uint8_t const pData, const uint8_t Offset, const uint16_t NbBytes)`	Reads part of Mailbox Message from the tag
	`void MBDecodeHeader (const uint8_t const pData, MB_HEADER_T const mb_header)`	Extracts global information from header in Fast transfer mode protocol
	`void PrepareMBMsg (uint8_t const pData, const MB_HEADER_T const mb_header)`	Prepares header message to send to Mailbox.
	`bool SendMBData (MB_HEADER_T const mb_header, const uint8_t nbretry)`	Prepares and writes frame in Mailbox
Variables	`uint8_t GPO_Activated`	Polling variable for the ST25DV GPO interrupt, updated from GPO interrupt callback.

5.5.1 Detailed description

This module proposes common API for the ST25DV Mailbox. It covers the following functions:

- Mailbox init.
- Read and Write to mailbox.
- Protocol header decode.
- Protocol message preparation.
- Protocol send message.

5.5.2 Data structure

struct MB_HEADER_T

Mailbox global header information structure definition.

Table 11. Data fields

		Description
`uint8_t`	chaining	Defines if the frame is a simple (0) or a chained frame (1)
`uint16_t`	chunknb	Informs for a specific frame the current chunk number of that frame
`uint8_t`	cmdresp	Defines current frame command, answer, acknowledge
`uint8_t`	error	Error code if an error is detected
`uint8_t`	fctcode	Function code used to define the action of the requester
`uint8_t`	framelength	Informs the data length of the current framer
`uint16_t`	framesize	Size of current frame to transfer
`uint32_t`	fulllength	Informs on the total length of data to transfer
`uint8_t`	pData	Pointer to buffer data to transfer
`uint16_t`	totalchunk	Informs on the total number of chunk that will need to perform the transfer

5.5.3 Function documentation

DeInitMailBoxMode()

```
NFCTAG_StatusTypeDef DeInitMailBoxMode (
void )
```

De-initializes the Mailbox mode, disables mailbox mode.

Parameters: None

Returns: NFCTAG_StatusTypeDef status.

InitMailBoxMode()

```
NFCTAG_StatusTypeDef InitMailBoxMode (
void )
```

Initializes the Mailbox mode, Enables Mailbox and disables Mailbox Watchdog.

Parameters: None

Returns: NFCTAG_StatusTypeDef status.

MBDecodeHeader()

```
void MBDecodeHeader (
const uint8_t const pData,
MB_HEADER_T const mb_header )
```

Extracts global information from header in Fast transfer mode protocol.

Parameters

pData	Pointer to the mailbox frame
mb_header	Pointer to structure for storing global header info

Return values: None

PrepareMBMsg()

```
void PrepareMBMsg (
uint8_t const pData,
const MB_HEADER_T const mb_header )
```

Prepares header message to send to the Mailbox.

Parameters

pData	Pointer to the mailbox frame
mb_header	Pointer to the header information structure

Return values: None

ReadCompleteMailBoxMsg()

```
NFCTAG_StatusTypeDef ReadCompleteMailBoxMsg (
uint8_t const pData,
uint16_t const pLength )
```

Reads entire Mailbox Message from the tag.

Parameters

pData	Pointer to the read data to store
pLength	Number of bytes to read

Returns: NFCTAG_StatusTypeDef status.

ReadFragmentMailBoxMsg()

```
NFCTAG_StatusTypeDef ReadFragmentMailBoxMsg (
uint8_t const pData,
const uint8_t Offset,
const uint16_t NbBytes )
```

Reads part of Mailbox Message from the tag.

Parameters

pData	Pointer to the stored data
Offset	Offset in Mailbox to start read
NbBYtes	Number of bytes to read

Returns: NFCTAG_StatusTypeDef status.

SendMBData()

```
bool SendMBData (
MB_HEADER_T const mb_header,
const uint8_t nbretry )
```

Prepares and writes frame in Mailbox.

Parameters

mb_header	Pointer to structure containing frame header info
nbretry	Number of attempts

Return values

1: Message was written to Mailbox.

0: Message was not written to Mailbox.

WriteMailBoxMsg()

```
NFCTAG_StatusTypeDef WriteMailBoxMsg (
const uint8_t const pData,
const uint16_t NbBytes )
```

Writes message in Mailbox.

Parameters

pData	Pointer to the data to write
NbBYtes	Number of bytes to write

Returns: NFCTAG_StatusTypeDef status.

5.6 Flash command

This module implements high level functions to write firmware or data to the Flash memory.

Table 12. Flash command functions - Overview

Section	Code	Description
Files	fw_command.c	Provides all the Flash memory programming command functions.
Functions	uint32_t COMMAND_EraseFlash (const uint32_t Address)	Command to erase specific Flash memory area.
	uint32_t Command_WriteBufferToFlash (const uint32_t StartAddress, const uint32_t offset, const uint8_t const pData, const uint32_t size)	Writes buffer to Flash memory.
	void COMMAND_Jump (void)	Jumps to user program.

5.6.1 Detailed description

This module implements high level functions to write firmware or data to the Flash memory. The module covers the following functions:

- Erase Flash memory command
- Write buffer to Flash memory
- Jump to firmware command.

5.6.2 Function documentation

COMMAND_EraseFlash()

```
uint32_t COMMAND_EraseFlash (
const uint32_t Address )
```

Command to erase specific Flash memory areas.

Parameter

Address Start address for erasing data

Return values

0: Erase sectors done with success.

1: Erase error.

COMMAND_Jump()

```
void COMMAND_Jump (
void )
```

Jumps to user program.

Parameters: None

Return value: None

Command_WriteBufferToFlash()

```
uint32_t Command_WriteBufferToFlash (
const uint32_t StartAddress,
const uint32_t offset,
const uint8_t const pData,
const uint32_t size )
```

Writes buffer to Flash memory.

Parameters

StartAddress	Start address for writing data
offset	Offset of data to write
pData	Buffer pointer to write
size	Size of data to write

Return values

0: Erase sectors done with success.

1: Erase error

5.7 Flash memory APIs

This module defines an API to access the internal Flash memory.

Table 13. Flash memory APIs module - Overview

Section	Code	Description
Files	flash_if.c	Provides all the Flash memory layer functions.
Functions	`void FLASH_If_FlashLock (void)`	Locks the Flash memory to disable the control register access.
	`void FLASH_If_FlashUnlock (void)`	Unlocks the Flash memory to enable its control register access.
	`FlagStatus FLASH_If_ReadOutProtectionStatus (void)`	Gets Flash memory readout protection status
	`uint32_t FLASH_If_EraseSectors (const uint32_t Address, const uint32_t LastAddress)`	Erases the required sectors computed with destination address
	`uint32_t FLASH_If_WriteByte (const uint32_t Address, const uint32_t LastAddress, const uint8_t Data)`	Writes data into Flash memory (data are 8-bit aligned)
	`uint32_t FLASH_If_Write (const uint32_t Address, const uint32_t LastAddress, const uint32_t Data)`	Writes data into Flash memory (data are 32-bit aligned)
	`uint32_t FLASH_If_WriteBuffer (const uint32_t Address, const uint32_t LastAddress, const uint8_t constpData, const uint32_t Size)`	Writes a data buffer into Flash memory (manage data alignment).

5.7.1 Detailed description

This module defines an API to access the internal Flash memory.

The module covers following functions:

- Lock Flash modification
- Unlock Flash modification
- Read Flash protection status
- Erase Flash sectors
- Write Flash data

5.7.2 Function documentation

FLASH_If_EraseSectors()

```
uint32_t FLASH_If_EraseSectors (
const uint32_t Address,
const uint32_t LastAddress )
```

Erases the required Flash memory sectors computed with destination address.

Parameters

Address	Start address for erasing data
LastAddress	End address of Flash memory area

Return values

0: Erase sectors done with success

1: Erase error

FLASH_If_FlashLock()

```
void FLASH_If_FlashLock (
void )
```

Locks the Flash to disable the Flash memory control register access.

Parameters: None

Return values: None

FLASH_If_FlashUnlock()

```
void FLASH_If_FlashUnlock (
void )
```

Unlocks the Flash to enable the Flash memory control register access.

Parameters: None

Return values: None

FLASH_If_ReadOutProtectionStatus()

```
FlagStatus FLASH_If_ReadOutProtectionStatus (
void )
```

Gets Flash memory readout protection status.

Parameters: None

Returns: ReadOut protection status

FLASH_If_Write()

```
uint32_t FLASH_If_Write (
const uint32_t Address,
const uint32_t LastAddress,
const uint32_t Data )
```

Writes a data into Flash memory (data are 32-bit aligned).

Parameters

Address	Start address for writing data buffer
LastAddress	End address of Flash memory area
Data	Word data value to write

Return values

1: Data successfully written to Flash memory

0: Error occurred while writing data in Flash memory

FLASH_If_WriteBuffer()

```
uint32_t FLASH_If_WriteBuffer (
const uint32_t Address,
const uint32_t LastAddress,
const uint8_t const pData,
const uint32_t Size )
```

Writes a data buffer into Flash memory (manages data alignment).

Parameters

Address	Start address for writing data buffer
LastAddress	End address of Flash memory area
pData	Pointer on data buffer
Size	Data size

Return values

0: Data successfully written to Flash memory

1: Error occurred while writing data in Flash memory

FLASH_If_WriteByte()

```
uint32_t FLASH_If_WriteByte (
const uint32_t Address,
const uint32_t LastAddress,
const uint8_t Data )
```

Writes a data into Flash memory (data are 8-bit aligned).

Parameters

Address	Start address for writing data buffer
LastAddress	End address of Flash memory area
Data	Byte data value to write

Return values

0: Data successfully written to Flash memory

1: Error occurred while writing data in Flash memory

5.8 ST25DV common functions

This module proposes a generic API to be used for the ST25DV, covering password presentation, GPO initialization and management.

Table 14. ST25DV common functions module - Overview

Section	Code	Description
Files	commonfunc.c	Common functions for the ST25DV management.
Data structures	struct IT_GPO_STATUS	ST25DV GPO interrupt status structure.
Macros	#define ST25_RETRY(cmd)	Iterates ST25DV command depending on the command return status.
Functions	ST25DV_I2CSSO_STATUS PresentPasswd (const bool passwd)	Presents password to the ST25DV to open or close an I2C session.
	`NFCTAG_StatusTypeDef InitITGPOMode (const uint16_t ITConfig)`	Enables and initializes the GPO interrupt.
	`void DeInitITGPOMode (void)`	Disables the GPO interrupt.
	`void ManageGPO (IT_GPO_STATUS const gpo)`	Reads the GPO interrupt source.
	`void st25_error (NFCTAG_StatusTypeDef status)`	Displays an error message depending on the return value of an NFC tag driver function.
Variables	`uint8_t GPO_Activated`	Polling variable for the ST25DV GPO interrupt, updated from GPO interrupt callback.

5.8.1 Data structure

The struct IT_GPO_STATUS is the ST25DV GPO interrupt status structure, used to return the event(s) that raised the GPO interrupt.

Table 15. Data fields

Type	Field name	Description
uint8_t	FieldOff	The RF field becomes inactive.
uint8_t	FieldOn	The RF field becomes active.
uint8_t	MailboxMsgRead	Fast Transfer Mode: message read by the RF.
uint8_t	MsgInMailbox	Fast Transfer Mode: message from the RF received.
uint8_t	RfInterrupt	Interrupt generated by a dedicated RF command.
uint8_t	Rfuser	GPO level controlled through the RF.
uint8_t	WriteInEEPROM	The EEPROM has been written by the RF.

5.8.2 Macro definition

ST25_RETRY

```
#define ST25_RETRY(
cmd )
```

Iterates ST25DV command depending on the command return status.

Parameter

cmd An ST25DV function returning a NFCTAG_StatusTypeDef status

5.8.3 Function documentation

DeInitITGPOMode()

```
void DeInitITGPOMode (
void )
```

Disables the GPO interrupt.

Parameters: None

Return values: None

InitITGPOMode()

```
NFCTAG_StatusTypeDef InitITGPOMode (
const uint16_t ITConfig )
```

Enables and initializes the GPO interrupt.

Parameter

ITConfig Value of the interrupt register to configure

Return: NFCTAG_StatusTypeDef status

ManageGPO()

```
void ManageGPO (
IT_GPO_STATUS const gpo )
```

Reads the GPO interrupt source.

This function reads the interrupt status register from the ST25DV to report which interrupt(s) occured.

Parameter

gpo Pointer on IT_GPO_STATUS structure, to return status of the GPO irq

Return values: None

PresentPasswd()

```
ST25DV_I2CSSO_STATUS PresentPasswd (
const bool passwd )
```

Presents password to the ST25DV to open or close an i2c session.

Parameter

passwd TRUE: open session, FALSE: close session

Return: ST25DV_I2CSSO_STATUS status

st25_error()

```
void st25_error (
NFCTAG_StatusTypeDef status )
```

Displays an error message depending on the return value of a NFCTAG driver function.

Parameter

status Return value from a NFCTAG driver function

5.9 Board support package

This is the container module for the BSP modules.

Table 16. BSP sub-modules

Name	Description
ST25 Discovery NFCTAG Board Support Package	Provides high-level functions to access the ST25DV NFC dynamic tag.
ST25DV driver	Implements the functions to drive the ST25DV NFC dynamic tag.

5.9.1 Detailed description

The Board Support Package software (BSP) is defined by the STM32Cube methodology as the abstraction layer for the board specific features. It implements all the functions required to access the components on the board and the MCU peripherals requiring a board specific configuration.

This module also includes the documentation for the ST25DV component driver.

5.10 ST25 Discovery NFCTAG board support package

Table 17. ST25 Discovery NFCTAG BSP module - Overview

Section	Code	Description
Files	st25_discovery_nfctag.c	This file provides a set of functions needed to manage an NFC dual interface EEPROM.
Macros	`#define NFCTAG_4K_SIZE ((uint32_t) 0x200)`	Number of bytes for the 4 Kbits NFC tag.
	`#define NFCTAG_16K_SIZE ((uint32_t) 0x800)`	Number of bytes for the 16 Kbits NFC tag.
	`#define NFCTAG_64K_SIZE ((uint32_t) 0x2000)`	Number of bytes for the 64 Kbits NFC tag.
Functions	`NFCTAG_StatusTypeDef BSP_NFCTAG_Init (void)`	Initializes peripherals used by the I2C NFC tag driver.
	`uint8_t BSP_NFCTAG_isInitialized (void)`	Checks if the NFC tag is initialized.
	`NFCTAG_StatusTypeDef BSP_NFCTAG_ReadID (uint8_t const wai_id)`	Reads the ID of the NFC tag.
	`uint32_t BSP_NFCTAG_GetByteSize (void)`	Returns the size of the NFC tag.
	`NFCTAG_StatusTypeDef BSP_NFCTAG_IsDeviceReady (const uint32_t Trials)`	Checks if the NFC tag is available.
	`NFCTAG_StatusTypeDef BSP_NFCTAG_ConfigIT (const uint16_t ITConfig)`	Configures the NFC tag interrupt.
	`NFCTAG_StatusTypeDef BSP_NFCTAG_GetITStatus (uint16_t const ITConfig)`	Reads the NFC tag interrupt configuration.
	`NFCTAG_StatusTypeDef BSP_NFCTAG_ReadData (uint8_t const pData, const uint16_t TarAddr, constuint16_t Size)`	Reads the data in the NFC tag at specified address.
	`NFCTAG_StatusTypeDef BSP_NFCTAG_WriteData (const uint8_t const pData, const uint16_t TarAddr,const uint16_t Size)`	Writes data to the NFC tag at specified address.
	`NFCTAG_StatusTypeDef BSP_NFCTAG_ReadRegister (uint8_t const pData, const uint16_t TarAddr, constuint16_t Size)`	Reads an NFC tag Register.
	`NFCTAG_StatusTypeDef BSP_NFCTAG_WriteRegister (const uint8_t const pData, const uint16_t TarAddr,const uint16_t Size)`	Writes an NFC tag Register.
	`NFCTAG_ExtDrvTypeDef BSP_NFCTAG_GetExtended_Drv (void)`	Access to the extended features of the NFC tag.

5.10.1 Detailed description

This module provides high-level functions to access the ST25DV NFC dynamic tag. These functions are designed to be called by the Application or by a Middleware.

5.10.2 Function documentation

BSP_NFCTAG_ConfigIT()

```
NFCTAG_StatusTypeDef BSP_NFCTAG_ConfigIT (
const uint16_t ITConfig )
```

Configures the NFC tag interrupt.

Parameter

ITConfig — Defines the interrupt mask to be configured.
- 0x01: RF BUSY
- 0x02: WIP
- 0x04: RF_INTERRUPT
- 0x08: FIELD_CHANGE
- 0x10: RF_PUT_MSG
- 0x20: RF_GET_MSG
- 0x40: RF_WRITE

Returns: NFCTAG_StatusTypeDef enum status.

BSP_NFCTAG_GetByteSize()

```
uint32_t BSP_NFCTAG_GetByteSize (
void )
```

Returns the size of the NFC tag.

Returns: Size of the NFC tag in Bytes.

BSP_NFCTAG_GetExtended_Drv()

```
NFCTAG_ExtDrvTypeDef  BSP_NFCTAG_GetExtended_Drv (
void )
```

Access to the extended features of the NFC tag.

Returns: Pointer on the Extended Component Structure for the NFC tag.

BSP_NFCTAG_GetITStatus()

```
NFCTAG_StatusTypeDef BSP_NFCTAG_GetITStatus (
uint16_t const ITConfig )
```

Reads the NFC tag interrupt configuration.

Parameter

ITConfig — Pointer on uint16_t used to return the interrupt configuration
- 0x01: RF BUSY
- 0x02: WIP
- 0x04: RF_INTERRUPT
- 0x08: FIELD_CHANGE
- 0x10: RF_PUT_MSG
- 0x20: RF_GET_MSG
- 0x40: RF_WRITE.

Returns: NFCTAG_StatusTypeDef enum status.

BSP_NFCTAG_Init()

```
NFCTAG_StatusTypeDef BSP_NFCTAG_Init (
void )
```

Initializes peripherals used by the I2C NFCTAG driver.

Returns: NFCTAG_StatusTypeDef enum status.

BSP_NFCTAG_IsDeviceReady()

```
NFCTAG_StatusTypeDef BSP_NFCTAG_IsDeviceReady (
const uint32_t Trials )
```

Checks if the NFC tag is available.

Parameter

Trials — Number of trials

Returns: NFCTAG_StatusTypeDef enum status.

BSP_NFCTAG_isInitialized()

```
uint8_t BSP_NFCTAG_isInitialized (
void )
```

Checks if the NFC tag is initialized.

Return values

0: the NFC tag is not initialized

1: the NFC tag is already initialized

BSP_NFCTAG_ReadData()

```
NFCTAG_StatusTypeDef BSP_NFCTAG_ReadData (
uint8_t const pData,
const uint16_t TarAddr,
const uint16_t Size )
```

Reads the data in the NFC tag at specified address.

Parameters

pData Pointer on the buffer used to store read data.

TarAddr I2C data memory address to be read.

Size Number of bytes to be read

Returns: NFCTAG_StatusTypeDef enum status.

BSP_NFCTAG_ReadID()

```
NFCTAG_StatusTypeDef BSP_NFCTAG_ReadID (
uint8_t const wai_id )
```

Reads the ID of the NFC tag.

Parameter

wai_id Pointer to where the who_am_i of the device is stored.

Returns: NFCTAG_StatusTypeDef enum status.

BSP_NFCTAG_ReadRegister()

```
NFCTAG_StatusTypeDef BSP_NFCTAG_ReadRegister (
uint8_t const pData,
const uint16_t TarAddr,
const uint16_t Size )
```

Reads a NFC tag Register.

Parameters

pData Pointer to the buffer used to store read data.

TarAddr I2C register address to be read.

Size Number of bytes to be read

Returns: NFCTAG_StatusTypeDef enum status.

BSP_NFCTAG_WriteData()

```
NFCTAG_StatusTypeDef BSP_NFCTAG_WriteData (
const uint8_t const pData,
const uint16_t TarAddr,
const uint16_t Size )
```

Writes data to the NFC tag at specified address.

Parameters

pData Pointer to the buffer containing the data to be written.

TarAddr I2C datamemory address to written.

Size Number of bytes to be written

Returns: NFCTAG_StatusTypeDef enum status.

BSP_NFCTAG_WriteRegister()

```
NFCTAG_StatusTypeDef BSP_NFCTAG_WriteRegister (
const uint8_t const pData,
const uint16_t TarAddr,
const uint16_t Size )
```

Writes an NFC tag Register.

Parameters

pData	Pointer to the buffer containing the data to be written.
TarAddr	I2C register address to written.
Size	Number of bytes to be written

Returns: NFCTAG_StatusTypeDef enum status.

5.11 ST25DV driver

This module implements the functions to drive the ST25DV NFC dynamic tag.

Table 18. ST25DV driver module - Overview

Section	Code	Description
Files	st25dv.c	Provides a set of driver functions to manage the communication between BSP and the ST25DV
Data structures	struct ST25DV_EH_CTRL	EH Ctrl structure definition.
	struct ST25DV_RF_MNGT	RF Management structure definition.
	struct ST25DV_RF_PROT_ZONE	RF Area protection structure definition.
	struct ST25DV_I2C_PROT_ZONE	I2C Area protection structure definition.
	struct ST25DV_MB_CTRL_DYN_STATUS	MB_CTRL_DYN register structure definition.
	struct ST25DV_LOCK_CCFILE	Lock CCFile structure definition
	struct ST25DV_MEM_SIZE	Memory size structure definition.
	struct ST25DV_UID	UID information structure definition.
	struct ST25DV_PASSWD	Password structure definition
Macros	`#define ST25DV_MAX_INSTANCE 1`	This component driver only supports 1 instance of the component.
	`#define ST25DV_ADDR_DATA_I2C 0xA6`	I2C address to be used for ST25DV data accesses.
	`#define ST25DV_ADDR_SYST_I2C 0xAE`	I2C address to be used for ST25DV System accesses.
	`#define ST25DV_I2C_TIMEOUT 400`	I2C Time out (ms), min value : (Max write bytes) / (Internal page write) tw (256/4)5.
	`#define ST25DV_MAX_WRITE_BYTE 256`	Size of the ST25DV write buffer.
	`#define ST25DV_MAX_MAILBOX_LENGTH 256`	Size of the ST25DVMailbox memory.
	`#define ST25DV_GPO_REG 0x0000`	GPO register address.
	`#define ST25DV_ITTIME_REG 0x0001`	IT duration register address.
	`#define ST25DV_EH_MODE_REG 0x0002`	Energy Harvesting register address.
	`#define ST25DV_RF_MNGT_REG 0x0003`	RF management register address.
	`#define ST25DV_RFZ1SS_REG 0x0004`	Area 1 security register address.
	`#define ST25DV_END1_REG 0x0005`	Area 1 end address register address.
	`#define ST25DV_RFZ2SS_REG 0x0006`	Area 2 security register address.
	`#define ST25DV_END2_REG 0x0007`	Area 2 end address register address.
	`#define ST25DV_RFZ3SS_REG 0x0008`	Area 3 security register address.
	`#define ST25DV_END3_REG 0x0009`	Area 3 end address register address.
	`#define ST25DV_RFZ4SS_REG 0x000A`	Area 4 security register address.
	`#define ST25DV_I2CZSS_REG 0x000B`	I2C security register address.

Table 18. ST25DV driver module - Overview (continued)

Section	Code	Description
Macros	`#define ST25DV_LOCKCCFILE_REG 0x000C`	Capability Container lock register address.
	`#define ST25DV_MB_MODE_REG 0x000D`	Mailbox mode register address.
	`#define ST25DV_MB_WDG_REG 0x000E`	Mailbox Watchdog register address.
	`#define ST25DV_LOCKCFG_REG 0x000F`	Configuration lock register address.
	`#define ST25DV_LOCKDSFID_REG 0x0010`	DSFID lock register address.
	`#define ST25DV_LOCKAFI_REG 0x0011`	AFI lock register address.
	`#define ST25DV_DSFID_REG 0x0012`	DSFID register address.
	`#define ST25DV_AFI_REG 0x0013`	AFI register address.
	`#define ST25DV_MEM_SIZE_REG 0x0014`	Memory size register address.
	`#define ST25DV_ICREF_REG 0x0017`	ICref register address.
	`#define ST25DV_UID_REG 0x0018`	UID register address.
	`#define ST25DV_ICREV_REG 0x0020`	IC revision register address.
	`#define ST25DV_I2CPASSWD_REG 0x0900`	I2C password register address.
	`#define ST25DV_EH_CTRL_DYN_REG 0x2002`	Energy Harvesting control dynamic register address.
	`#define ST25DV_RF_MNGT_DYN_REG 0x2003`	RF management dynamic register address
	`#define ST25DV_I2C_SSO_DYN_REG 0x2004`	I2C secure session opened dynamic register address.
	`#define ST25DV_ITSTS_DYN_REG 0x2005`	Interrupt status dynamic register address.
	`#define ST25DV_MB_CTRL_DYN_REG 0x2006`	Mailbox control dynamic register address.
	`#define ST25DV_MBLEN_DYN_REG 0x2007`	Mailbox message length dynamic register address.
	`#define ST25DV_MAILBOX_RAM_REG 0x2008`	Mailbox buffer address.

Table 18. ST25DV driver module - Overview (continued)

Section	Code	Description
Enumerations	enum NFCTAG_StatusTypeDef	NFC tag status enumerator definition.
	enum ST25DV_EN_STATUS	Enable Disable enumerator definition.
	enum ST25DV_EH_MODE_STATUS	Energy Harvesting mode enumerator definition.
	enum ST25DV_FIELD_STATUS	FIELD status enumerator definition.
	enum ST25DV_VCC_STATUS	VCC status enumerator definition.
	enum ST25DV_PROTECTION_CONF	Protection status enumerator definition.
	enum ST25DV_PROTECTION_ZONE	Area protection enumerator definition.
	enum ST25DV_PASSWD_PROT_STATUS	Password protection status enumerator definition.
	enum ST25DV_LOCK_STATUS	Lock status enumerator definition.
	enum ST25DV_CCFILE_BLOCK	Number of blocks for the CCFile enumerator definition.
	enum ST25DV_I2CSSO_STATUS	Session status enumerator definition.
	enum ST25DV_END_ZONE	Area end address enumerator definition.
	enum ST25DV_PULSE_DURATION	IT pulse duration enumerator definition.
	enum ST25DV_CURRENT_MSG	Mailbox current message enumerator definition.
Functions	NFCTAG_StatusTypeDef ST25DV_i2c_Init (void)	NFC tag initialization.
	NFCTAG_StatusTypeDef ST25DV_i2c_ReadID (uint8_t const pICRef)	Reads the ST25DV ID.
	NFCTAG_StatusTypeDef ST25DV_i2c_ReadICRev (uint8_t const pICRev)	Reads the ST25DV IC revision
	NFCTAG_StatusTypeDef ST25DV_i2c_IsDeviceReady (const uint32_t Trials)	Checks the ST25DV availability.
	NFCTAG_StatusTypeDef ST25DV_i2c_GetGPOStatus (uint16_t const pGPOStatus)	Reads the ST25DV GPO configuration.
	NFCTAG_StatusTypeDef ST25DV_i2c_ConfigureGPO (const uint16_t ITConf)	Configures the ST25DV GPO.
	NFCTAG_StatusTypeDef ST25DV_i2c_ReadITPulse (ST25DV_PULSE_DURATION const pITtime)	Reads the ST25DV ITtime duration for the GPO pulses.

Table 18. ST25DV driver module - Overview (continued)

Section	Code	Description
Functions	`NFCTAG_StatusTypeDef ST25DV_i2c_WriteITPulse (const ST25DV_PULSE_DURATION ITtime)`	Configures the ST25DV ITtime duration for the GPO pulse.
	`NFCTAG_StatusTypeDef ST25DV_i2c_ReadData (uint8_t const pData, const uint16_t TarAddr, const uint16_t NbByte)`	Reads N bytes of data, starting from the specified I2C address.
	`NFCTAG_StatusTypeDef ST25DV_i2c_WriteData (const uint8_t const pData, const uint16_t TarAddr, const uint16_t NbByte)`	Writes N bytes of data, starting from the specified I2C Address.
	`NFCTAG_StatusTypeDef ST25DV_i2c_ReadDataCurrentAddr (uint8_t const pData, const uint16_t NbByte)`	Reads N bytes of data, starting at current address.
	`NFCTAG_StatusTypeDef ST25DV_i2c_ReadRegister (uint8_t const pData, const uint16_t TarAddr, const uint16_t NbByte)`	Reads N bytes from registers, starting at the specified I2C address.
	`NFCTAG_StatusTypeDef ST25DV_i2c_WriteRegister (const uint8_t const pData, const uint16_t TarAddr, const uint16_t NbByte)`	Writes N bytes to the specified register.
	`NFCTAG_StatusTypeDef ST25DV_i2c_ReadUID (ST25DV_UID const pUid)`	Reads the ST25DV UID.
	`NFCTAG_StatusTypeDef ST25DV_i2c_ReadDSFID (uint8_t const pDsfid)`	Reads the ST25DV DSFID.
	`NFCTAG_StatusTypeDef ST25DV_i2c_ReadDsfidRFProtection (ST25DV_LOCK_STATUS const pLock - Dsfid)`	Reads the ST25DV DSFID RF Lock state.
	`NFCTAG_StatusTypeDef ST25DV_i2c_ReadAFI (uint8_t const pAfi)`	Reads the ST25DV AFI.
	`NFCTAG_StatusTypeDef ST25DV_i2c_ReadAfiRFProtection (ST25DV_LOCK_STATUS const pLockAfi)`	Reads the AFI RF Lock state.
	`NFCTAG_StatusTypeDef ST25DV_i2c_ReadI2CProtectZone (ST25DV_I2C_PROT_ZONE const pProtZone)`	Reads the I2C Protected Area state.

Table 18. ST25DV driver module - Overview (continued)

Section	Code	Description
Functions	`NFCTAG_StatusTypeDef ST25DV_i2c_WriteI2CProtectZonex (const ST25DV_PROTECTION_ZONE Zone, const ST25DV_PROTECTION_CONF ReadWriteProtection)`	Sets the I2C write-protected state to an EEPROM area
	`NFCTAG_StatusTypeDef ST25DV_i2c_ReadLockCCFile (ST25DV_LOCK_CCFILE const pLockCCFile)`	Reads the CCfile protection state.
	`NFCTAG_StatusTypeDef ST25DV_i2c_WriteLockCCFile (const ST25DV_CCFILE_BLOCK NbBlockCCFile, const ST25DV_LOCK_STATUS LockCCFile)`	Locks the CCfile to prevent any RF write access.
	`NFCTAG_StatusTypeDef ST25DV_i2c_ReadLockCFG (ST25DV_LOCK_STATUS const pLockCfg)`	Reads the Cfg registers protection.
	`NFCTAG_StatusTypeDef ST25DV_i2c_WriteLockCFG (const ST25DV_LOCK_STATUS LockCfg)`	Locks/unlocks the Cfg registers, to prevent any RF write access.
	`NFCTAG_StatusTypeDef ST25DV_i2c_PresentI2CPassword (const ST25DV_PASSWD PassWord)`	Presents I2C password, to authorize I2C writes to the protected areas.
	`NFCTAG_StatusTypeDef ST25DV_i2c_WriteI2CPassword (const ST25DV_PASSWD PassWord)`	Writes a new I2C password.
	`NFCTAG_StatusTypeDef ST25DV_i2c_ReadRFZxSS (const ST25DV_PROTECTION_ZONE Zone, ST25 - DV_RF_PROT_ZONE const pRfprotZone)`	Reads the RF Zone Security Status (defining the allowed RF accesses).
	`NFCTAG_StatusTypeDef ST25DV_i2c_WriteRFZxSS (const ST25DV_PROTECTION_ZONE Zone, const ST25DV_RF_PROT_ZONE RfProtZone)`	Writes the RF Zone Security Status (defining the allowed RF accesses)
	`NFCTAG_StatusTypeDef ST25DV_i2c_ReadEndZonex (const ST25DV_END_ZONE EndZone, uint8_-t const pEndZ)`	Reads the value of the an area end address.
	`NFCTAG_StatusTypeDef ST25DV_i2c_WriteEndZonex (const ST25DV_END_ZONE EndZone, const uint8_t EndZ)`	Sets the end address of an area.
	`NFCTAG_StatusTypeDef ST25DV_i2c_InitEndZone (void)`	Initializes the end address of the ST25DV areas with their default values (end of memory).

Table 18. ST25DV driver module - Overview (continued)

Section	Code	Description
Functions	`NFCTAG_StatusTypeDef ST25DV_i2c_CreateUserZone (uint16_t Zone1Length, uint16_t Zone2Length, uint16_t Zone3Length, uint16_t Zone4Length)`	Creates user areas with defined lengths.
	`NFCTAG_StatusTypeDef ST25DV_i2c_ReadMemSize (ST25DV_MEM_SIZE const pSizeInfo)`	Reads the ST25DV memory size.
	`NFCTAG_StatusTypeDef ST25DV_i2c_ReadEHMode (ST25DV_EH_MODE_STATUS const pEH_mode)`	Reads the Energy harvesting mode.
	`NFCTAG_StatusTypeDef ST25DV_i2c_WriteEHMode (const ST25DV_EH_MODE_STATUS EH_mode)`	Sets the Energy harvesting mode.
	`NFCTAG_StatusTypeDef ST25DV_i2c_ReadRFMngt (ST25DV_RF_MNGT const pRF_Mngt)`	Reads the RF Management configuration.
	`NFCTAG_StatusTypeDef ST25DV_i2c_WriteRFMngt (const uint8_t Rfmngt)`	Sets the RF Management configuration.
	`NFCTAG_StatusTypeDef ST25DV_i2c_GetRFDisable (ST25DV_EN_STATUS const pRFDisable)`	Reads the RFDisable register information.
	`NFCTAG_StatusTypeDef ST25DV_i2c_SetRFDisable (void)`	Sets the RF Disable configuration.
	`NFCTAG_StatusTypeDef ST25DV_i2c_ResetRFDisable (void)`	Resets the RF Disable configuration.
	`NFCTAG_StatusTypeDef ST25DV_i2c_GetRFSleep (ST25DV_EN_STATUS const pRFSleep)`	Reads the RFSleep register information.
	`NFCTAG_StatusTypeDef ST25DV_i2c_SetRFSleep (void)`	Sets the RF Sleep configuration.
	`NFCTAG_StatusTypeDef ST25DV_i2c_ResetRFSleep (void)`	Resets the RF Sleep configuration.
	`NFCTAG_StatusTypeDef ST25DV_i2c_ReadMBMode (ST25DV_EN_STATUS const pMB_mode)`	Reads the Mailbox mode.
	`NFCTAG_StatusTypeDef ST25DV_i2c_WriteMBMode (const ST25DV_EN_STATUS MB_mode)`	Sets the Mailbox mode.
	`NFCTAG_StatusTypeDef ST25DV_i2c_ReadMBWDG (uint8_t const pWdgDelay)`	Reads the Mailbox watchdog duration coefficient.

Table 18. ST25DV driver module - Overview (continued)

Section	Code	Description
Functions	`NFCTAG_StatusTypeDef ST25DV_i2c_WriteMBWDG (const uint8_t WdgDelay)`	Writes the Mailbox watchdog coefficient delay.
	`NFCTAG_StatusTypeDef ST25DV_i2c_ReadMailboxData (uint8_t const pData, const uint16_t Offset, const uint16_t NbByte)`	Reads N bytes of data from the Mailbox, starting at the specified byte offset.
	`NFCTAG_StatusTypeDef ST25DV_i2c_WriteMailboxData (const uint8_t const pData, const uint16_t Nb -Byte)`	Writes N bytes of data in the Mailbox, starting from first Mailbox Address.
	`NFCTAG_StatusTypeDef ST25DV_i2c_ReadMailboxRegister (uint8_t const pData, const uint16_t TarAddr, const uint16_t NbByte)`	Reads N bytes from the mailbox registers, starting at the specified I2C address.
	`NFCTAG_StatusTypeDef ST25DV_i2c_WriteMailboxRegister (const uint8_t const pData, const uint16_ -t TarAddr, const uint16_t NbByte)`	Writes N bytes to the specified mailbox register.
	`NFCTAG_StatusTypeDef ST25DV_i2c_ReadI2CSecuritySession_Dyn (ST25DV_I2CSSO_STATUS const pSession)`	Reads the status of the security session open register.
	`NFCTAG_StatusTypeDef ST25DV_i2c_ReadITSTStatus_Dyn (uint8_t const pITStatus)`	Reads the IT status register from the ST25DV.
	`NFCTAG_StatusTypeDef ST25DV_i2c_ReadEHCtrl_Dyn (ST25DV_EH_CTRL const pEH_CTRL)`	Reads the value of dynamic EH Ctrl register configuration.
	`NFCTAG_StatusTypeDef ST25DV_i2c_GetEHENMode_Dyn (ST25DV_EN_STATUS const pEH_Val)`	Reads the Energy Harvesting dynamic status.
	`NFCTAG_StatusTypeDef ST25DV_i2c_SetEHENMode_Dyn (void)`	Dynamically sets the Energy Harvesting mode.
	`NFCTAG_StatusTypeDef ST25DV_i2c_ResetEHENMode_Dyn (void)`	Dynamically unsets the Energy Harvesting mode.
	`NFCTAG_StatusTypeDef ST25DV_i2c_GetEHON_Dyn (ST25DV_EN_STATUS const pEHON)`	Reads the EH_ON status from the EH_CTRL_DYN register.
	`NFCTAG_StatusTypeDef ST25DV_i2c_GetRFField_Dyn (ST25DV_FIELD_STATUS const pRF_Field)`	Checks if RF field is present in front of the ST25DV.

Table 18. ST25DV driver module - Overview (continued)

Section	Code	Description
Functions	`NFCTAG_StatusTypeDef ST25DV_i2c_GetVCC_Dyn (ST25DV_VCC_STATUS const pVCC)`	Check if VCC is supplying the ST25DV.
	`NFCTAG_StatusTypeDef ST25DV_i2c_ReadRFMngt_Dyn (ST25DV_RF_MNGT const pRF_Mngt)`	Read value of dynamic RF Management configuration.
	`NFCTAG_StatusTypeDef ST25DV_i2c_WriteRFMngt_Dyn (const uint8_t RF_Mngt)`	Writes a value to the RF Management dynamic register.
	`NFCTAG_StatusTypeDef ST25DV_i2c_GetRFDisable_Dyn (ST25DV_EN_STATUS const pRFDisable)`	Reads the RFDisable dynamic register information.
	`NFCTAG_StatusTypeDef ST25DV_i2c_SetRFDisable_Dyn (void)`	Sets the RF Disable dynamic configuration.
	`NFCTAG_StatusTypeDef ST25DV_i2c_ResetRFDisable_Dyn (void)`	Unsets the RF Disable dynamic configuration.
	`NFCTAG_StatusTypeDef ST25DV_i2c_GetRFSleep_Dyn (ST25DV_EN_STATUS const pRFSleep)`	Reads the RFSleep dynamic register information.
	`NFCTAG_StatusTypeDef ST25DV_i2c_SetRFSleep_Dyn (void)`	Sets the RF Sleep dynamic configuration.
	`NFCTAG_StatusTypeDef ST25DV_i2c_ResetRFSleep_Dyn (void)`	Unsets the RF Sleep dynamic configuration.
	`NFCTAG_StatusTypeDef ST25DV_i2c_ReadMBCtrl_Dyn (ST25DV_MB_CTRL_DYN_STATUS const p - CtrlStatus)`	Reads the Mailbox ctrl dynamic register.
	`NFCTAG_StatusTypeDef ST25DV_i2c_GetMBEN_Dyn (ST25DV_EN_STATUS const pMBEN)`	Reads the Mailbox Enable dynamic configuration.
	`NFCTAG_StatusTypeDef ST25DV_i2c_SetMBEN_Dyn (void)`	Sets the Mailbox Enable dynamic configuration.
	`NFCTAG_StatusTypeDef ST25DV_i2c_ResetMBEN_Dyn (void)`	Unsets the Mailbox Enable dynamic configuration.
	`NFCTAG_StatusTypeDef ST25DV_i2c_ReadMBLength_Dyn (uint8_t const pMBLength)`	Reads the Mailbox message length dynamic register.

Table 18. ST25DV driver module - Overview (continued)

Section	Code	Description
Variables	`NFCTAG_DrvTypeDef St25Dv_i2c_Drv`	Standard NFC tag driver API for the ST25DV.
	`NFCTAG_ExtDrvTypeDef St25Dv_i2c_ExtDrv`	Extended NFC tag driver API for the ST25DV.
	`uint8_t aSt25Dv [ST25DV_MAX_INSTANCE] = {0}`	ST25DV instances by address.
	`NFCTAG_DrvTypeDef St25Dv_i2c_Drv`	Standard NFC tag driver API for the ST25DV.
	`NFCTAG_ExtDrvTypeDef St25Dv_i2c_ExtDrv`	Extended NFC tag driver API for the ST25DV

5.11.1 Detailed description

This module implements the functions to drive the ST25DV NFC dynamic tag.

As recommended by the STM32Cube methodology, this driver provides a standard structure to expose the NFC tag standard API. It also provides an extended API through its extended driver structure. To be usable on any MCU, this driver calls several IOBus functions. The IOBus functions are implemented outside this driver, and are in charge of accessing the MCU peripherals used for the communication with the tag.

5.11.2 Data structure documentation

struct ST25DV_EH_CTRL

ST25DV EH Ctrl structure definition.

struct ST25DV_GPO

ST25DV GPO structure definition.

struct ST25DV_RF_MNGT

ST25DV RF Management structure definition.

struct ST25DV_RF_PROT_ZONE

ST25DV RF Area protection structure definition.

struct ST25DV_I2C_PROT_ZONE

ST25DV I2C Area protection structure definition.

struct ST25DV_MB_CTRL_DYN_STATUS

ST25DV MB_CTRL_DYN register structure definition.

struct ST25DV_LOCK_CCFILE

ST25DV Lock CCFile structure definition.

struct ST25DV_MEM_SIZE

ST25DV Memory size structure definition.

struct ST25DV_UID

ST25DV UID information structure definition.

struct ST25DV_PASSWD

ST25DV Password structure definition.

5.11.3 Function documentation

ST25DV_i2c_ConfigureGPO()

```
NFCTAG_StatusTypeDef ST25DV_i2c_ConfigureGPO (
const uint16_t ITConf )
```

Configures the ST25DV GPO. Needs the I2C Password presentation to be effective.

Parameter

ITConf — Provides the GPO configuration to apply:
- RFUSERSTATE = 0x01
- RFBUSY = 0x02
- RFINTERRUPT = 0x04
- FIELDFALLING = 0x08
- FIELDRISING = 0x10
- RFPUTMSG = 0x20
- RFGETMSG = 0x40
- RFWRITE = 0x80

Returns: NFCTAG_StatusTypeDef enum status.

ST25DV_i2c_CreateUserZone()

```
NFCTAG_StatusTypeDef ST25DV_i2c_CreateUserZone (
uint16_t Zone1Length,
uint16_t Zone2Length,
uint16_t Zone3Length,
uint16_t Zone4Length )
```

Creates user areas with defined lengths. Needs the I2C Password presentation to be effective.

Parameters

Zone1Length — Length of area1 in bytes (32 to 8192, 0x20 to 0x2000)

Zone2Length — Length of area2 in bytes (0 to 8128, 0x00 to 0x1FC0)

Zone3Length — Length of area3 in bytes (0 to 8064, 0x00 to 0x1F80)

Zone4Length — Length of area4 in bytes (0 to 8000, 0x00 to 0x1F40)

Returns: NFCTAG_StatusTypeDef enum status.

ST25DV_i2c_GetEHENMode_Dyn()

```
NFCTAG_StatusTypeDef ST25DV_i2c_GetEHENMode_Dyn (
ST25DV_EN_STATUS const pEH_Val )
```

Reads the Energy Harvesting dynamic status.

Parameter

pEH_Val — Pointer to a ST25DV_EN_STATUS value used to return the Energy Harvesting dynamic status.

Returns: NFCTAG_StatusTypeDef enum status.

ST25DV_i2c_GetEHON_Dyn()

```
NFCTAG_StatusTypeDef ST25DV_i2c_GetEHON_Dyn (
ST25DV_EN_STATUS const pEHON )
```

Reads the EH_ON status from the EH_CTRL_DYN register.

Parameter

pEHON — Pointer to a ST25DV_EN_STATUS value used to return the EHON status.

Returns: NFCTAG_StatusTypeDef enum status.

ST25DV_i2c_GetGPO_en_Dyn()

```
NFCTAG_StatusTypeDef ST25DV_i2c_GetGPO_en_Dyn (
ST25DV_EN_STATUS const pGPO_en )
```

Gets dynamic GPO enable status.

Parameter

pGPO_enl — ST25DV_EN_STATUS pointer of the GPO enable status to store.

Returns: NFCTAG enum status

ST25DV_i2c_GetGPOStatus()

```
NFCTAG_StatusTypeDef ST25DV_i2c_GetGPOStatus (
uint16_t const pGPOStatus )
```

Reads the ST25DV GPO configuration.

Parameter

pGPOStatus — Pointer to a uint16_t used to return the current GPO configuration, as:

- RFUSERSTATE = 0x01
- RFBUSY = 0x02
- RFINTERRUPT = 0x04
- FIELDFALLING = 0x08
- FIELDRISING = 0x10
- RFPUTMSG = 0x20
- RFGETMSG = 0x40
- RFWRITE = 0x80

Returns: NFCTAG_StatusTypeDef enum status.

ST25DV_i2c_GetMBEN_Dyn()

```
NFCTAG_StatusTypeDef ST25DV_i2c_GetMBEN_Dyn (
ST25DV_EN_STATUS const pMBEN )
```

Reads the Mailbox Enable dynamic configuration.

Returns: NFCTAG_StatusTypeDef enum status.

ST25DV_i2c_GetRFDisable()

```
NFCTAG_StatusTypeDef ST25DV_i2c_GetRFDisable (
ST25DV_EN_STATUS const pRFDisable )
```

Reads the RFDisable register information.

Parameter

pRFDisable — Pointer to an ST25DV_EN_STATUS value corresponding to the RF Disable status.

Returns: NFCTAG_StatusTypeDef enum status.

ST25DV_i2c_GetRFDisable_Dyn()

```
NFCTAG_StatusTypeDef ST25DV_i2c_GetRFDisable_Dyn (
ST25DV_EN_STATUS const pRFDisable )
```

Reads the RFDisable dynamic register information.

Parameter

pRFDisable — Pointer to an ST25DV_EN_STATUS value used to return the RF Disable state.

Returns: NFCTAG_StatusTypeDef enum status.

ST25DV_i2c_GetRFField_Dyn()

```
NFCTAG_StatusTypeDef ST25DV_i2c_GetRFField_Dyn (
ST25DV_FIELD_STATUS const pRF_Field )
```

Checks if RF field is present in front of the ST25DV.

Parameter

pRF_Field — Pointer to an ST25DV_FIELD_STATUS value used to return the field presence.

Returns: NFCTAG_StatusTypeDef enum status.

ST25DV_i2c_GetRFSleep()

```
NFCTAG_StatusTypeDef ST25DV_i2c_GetRFSleep (
ST25DV_EN_STATUS const pRFSleep )
```

Reads the RFSleep register information.

Parameter

pRFSleep — Pointer to an ST25DV_EN_STATUS value corresponding to the RF Sleep status.

Returns: NFCTAG_StatusTypeDef enum status.

ST25DV_i2c_GetRFSleep_Dyn()

```
NFCTAG_StatusTypeDef ST25DV_i2c_GetRFSleep_Dyn (
ST25DV_EN_STATUS const pRFSleep )
```

Reads the RFSleep dynamic register information.

Parameter

pRFSleep — Pointer to an ST25DV_EN_STATUS values used to return the RF Sleep state.

Returns: NFCTAG_StatusTypeDef enum status.

ST25DV_i2c_GetVCC_Dyn()

```
NFCTAG_StatusTypeDef ST25DV_i2c_GetVCC_Dyn (
ST25DV_VCC_STATUS const pVCC )
```

Checks if VCC is supplying the ST25DV.

Parameter

pVCC — ST25DV_VCC_STATUS pointer of the VCC status to store

Returns: NFCTAG enum status.

ST25DV_i2c_Init()

```
NFCTAG_StatusTypeDef ST25DV_i2c_Init (
void )
```

ST25DV NFC tag Initialization.

Returns: NFCTAG_StatusTypeDef enum status.

ST25DV_i2c_InitEndZone()

```
NFCTAG_StatusTypeDef ST25DV_i2c_InitEndZone (
void )
```

Initializes the end address of the ST25DV areas with their default values (end of memory). Needs the I2C Password presentation to be effective. The ST25DV answers a NACK when setting the EndZone2 and EndZone3 to the same value of, respectively, EndZone1 and EndZone2. These NACKs are ok.

Returns: NFCTAG_StatusTypeDef enum status.

ST25DV_i2c_IsDeviceReady()

```
NFCTAG_StatusTypeDef ST25DV_i2c_IsDeviceReady (
const uint32_t Trials )
```

Checks the ST25DV availability.

The ST25DV I2C is NACKed when a RF communication is on-going. This function determines if the ST25DV is ready to answer an I2C request.

Parameter

Trials Maximum number of trials

Returns: NFCTAG_StatusTypeDef enum status.

ST25DV_i2c_PresentI2CPassword()

```
NFCTAG_StatusTypeDef ST25DV_i2c_PresentI2CPassword (
const ST25DV_PASSWD PassWord )
```

Presents I2C password, to authorize the I2C writes to protected areas.

Parameter

PassWord Password value (32 bits)

Returns: NFCTAG_StatusTypeDef enum status.

ST25DV_i2c_ReadAFI()

```
NFCTAG_StatusTypeDef ST25DV_i2c_ReadAFI (
uint8_t const pAfi )
```

Reads the ST25DV AFI.

Parameter

pAfi Pointer used to return the ST25DV AFI value.

Returns: NFCTAG_StatusTypeDef enum status.

ST25DV_i2c_ReadAfiRFProtection()

```
NFCTAG_StatusTypeDef ST25DV_i2c_ReadAfiRFProtection (
ST25DV_LOCK_STATUS const pLockAfi )
```

Reads the AFI RF Lock state.

Parameter

pLockAfi Pointer to ST25DV_LOCK_STATUS used to return the ASFID lock state.

Returns: NFCTAG_StatusTypeDef enum status.

ST25DV_i2c_ReadData()

```
NFCTAG_StatusTypeDef ST25DV_i2c_ReadData (
uint8_t const pData,
const uint16_t TarAddr,
const uint16_t NbByte )
```

Reads N bytes of Data, starting from the specified I2C address.

Parameters

pData Pointer used to return the read data.

TarAddr I2C data memory address to read.

NbByte Number of bytes to be read.

Returns: NFCTAG_StatusTypeDef enum status.

ST25DV_i2c_ReadDataCurrentAddr()

```
NFCTAG_StatusTypeDef ST25DV_i2c_ReadDataCurrentAddr (
uint8_t const pData,
const uint16_t NbByte )
```

Reads N bytes of Data, starting at current address.

Parameters

pData Pointer used to return the read data.

NbByte Number of bytes to be read.

Returns: NFCTAG_StatusTypeDef enum status.

ST25DV_i2c_ReadDSFID()

```
NFCTAG_StatusTypeDef ST25DV_i2c_ReadDSFID (
uint8_t const pDsfid )
```

Reads the ST25DV DSFID.

Parameter

pDsfid Pointer used to return the ST25DV DSFID value.

Returns: NFCTAG_StatusTypeDef enum status.

ST25DV_i2c_ReadDsfidRFProtection()

```
NFCTAG_StatusTypeDef ST25DV_i2c_ReadDsfidRFProtection (
ST25DV_LOCK_STATUS const pLockDsfid )
```

Reads the ST25DV DSFID RF Lock state.

Parameter

pLockDsfid Pointer to ST25DV_LOCK_STATUS used to return the DSFID lock state.

Returns: NFCTAG_StatusTypeDef enum status.

ST25DV_i2c_ReadEHCtrl_Dyn()

```
NFCTAG_StatusTypeDef ST25DV_i2c_ReadEHCtrl_Dyn (
ST25DV_EH_CTRL const pEH_CTRL )
```

Reads the value of dynamic EH Ctrl register configuration.

Parameter

pEH_CTRL ST25DV_EH_CTRL pointer of the dynamic EH Ctrl configuration to store

Returns: NFCTAG enum status

ST25DV_i2c_ReadEHMode()

```
NFCTAG_StatusTypeDef ST25DV_i2c_ReadEHMode (
ST25DV_EH_MODE_STATUS const pEH_mode )
```

Reads the Energy harvesting mode.

Parameter

pEH_mode — Pointer to an ST25DV_EH_MODE_STATUS value corresponding to the Energy Harvesting state.

Returns: NFCTAG_StatusTypeDef enum status.

ST25DV_i2c_ReadEndZonex()

```
NFCTAG_StatusTypeDef ST25DV_i2c_ReadEndZonex (
const ST25DV_END_ZONE EndZone,
uint8_t const pEndZ )
```

Reads the value of an area end address.

Parameters

EndZone — ST25DV_END_ZONE value corresponding to an area end address.

pEndZ — Pointer used to return the end address of the area.

Returns: NFCTAG_StatusTypeDef enum status.

ST25DV_i2c_ReadGPO_Dyn()

```
NFCTAG_StatusTypeDef ST25DV_i2c_ReadGPO_Dyn (
uint8_t  GPOConfig )
```

Reads the value of the dynamic GPO register configuration.

Parameter

pGPO — ST25DV_GPO pointer of the dynamic GPO configuration to store.

Returns: NFCTAG enum status.

ST25DV_i2c_ReadI2CProtectZone()

```
NFCTAG_StatusTypeDef ST25DV_i2c_ReadI2CProtectZone (
ST25DV_I2C_PROT_ZONE const pProtZone )
```

Reads the I2C Protected Area state.

Parameter

pProtZone — Pointer to an ST25DV_I2C_PROT_ZONE structure used to return the Protected Area state.

Returns: NFCTAG_StatusTypeDef enum status.

ST25DV_i2c_ReadI2CSecuritySession_Dyn()

```
NFCTAG_StatusTypeDef ST25DV_i2c_ReadI2CSecuritySession_Dyn (
ST25DV_I2CSSO_STATUS const pSession )
```

Reads the status of the security session open register.

Parameter

pSession — Pointer to an ST25DV_I2CSSO_STATUS value used to return the session status.

Returns: NFCTAG_StatusTypeDef enum status.

ST25DV_i2c_ReadICRev()

```
NFCTAG_StatusTypeDef ST25DV_i2c_ReadICRev (
uint8_t const pICRev )
```

Reads the ST25DV IC Revision.

Parameter

pICRev — Pointer on the uint8_t used to return the ST25DV IC Revision number.

Returns: NFCTAG_StatusTypeDef enum status.

ST25DV_i2c_ReadID()

```
NFCTAG_StatusTypeDef ST25DV_i2c_ReadID (
uint8_t const pICRef )
```

Reads the ST25DV ID.

Parameter

pICRef — Pointer to an uint8_t used to return the ST25DV ID.

Returns: NFCTAG_StatusTypeDef enum status.

ST25DV_i2c_ReadITPulse()

```
NFCTAG_StatusTypeDef ST25DV_i2c_ReadITPulse (
ST25DV_PULSE_DURATION const pITtime )
```

Reads the ST25DV ITtime duration for the GPO pulses.

Parameter

pITtime — Pointer used to return the coefficient for the GPO Pulse duration, equal to 302.06 µs - ITtime * 512 / fc).

Returns: NFCTAG_StatusTypeDef enum status.

ST25DV_i2c_ReadITSTStatus_Dyn()

```
NFCTAG_StatusTypeDef ST25DV_i2c_ReadITSTStatus_Dyn (
uint8_t const pITStatus )
```

Reads the IT status register from the ST25DV.

Parameter

pITStatus — Pointer on uint8_t, used to return the IT status, such as:

- RFUSERSTATE = 0x01
- RFBUSY = 0x02
- RFINTERRUPT = 0x04
- FIELDFALLING = 0x08
- FIELDRISING = 0x10
- RFPUTMSG = 0x20
- RFGETMSG = 0x40
- RFWRITE = 0x80

Returns: NFCTAG_StatusTypeDef enum status.

ST25DV_i2c_ReadLockCCFile()

```
NFCTAG_StatusTypeDef ST25DV_i2c_ReadLockCCFile (
ST25DV_LOCK_CCFILE const pLockCCFile )
```

Reads the CCile protection state.

Parameter

pLockCCFile — Pointer to an ST25DV_LOCK_CCFILE value corresponding to the lock state of the CCFile.

Returns: NFCTAG_StatusTypeDef enum status.

ST25DV_i2c_ReadLockCFG()

```
NFCTAG_StatusTypeDef ST25DV_i2c_ReadLockCFG (
ST25DV_LOCK_STATUS const pLockCfg )
```

Reads the Cfg registers protection.

Parameter

pLockCfg — Pointer to an ST25DV_LOCK_STATUS value corresponding to the Cfg registers lock state.

Returns: NFCTAG_StatusTypeDef enum status.

```
ST25DV_i2c_ReadMailboxData()
NFCTAG_StatusTypeDef ST25DV_i2c_ReadMailboxData (
uint8_t const pData,
const uint16_t Offset,
const uint16_t NbByte )
```

Reads N bytes of data from the Mailbox, starting at the specified byte offset.

Parameters

pData — Pointer on the buffer used to return the read data.

Offset — Offset in the Mailbox memory, byte number to start the read.

nbByte — Number of bytes to be read.

Returns: NFCTAG_StatusTypeDef enum status.

ST25DV_i2c_ReadMailboxRegister()

```
NFCTAG_StatusTypeDef ST25DV_i2c_ReadMailboxRegister (
uint8_t const pData,
const uint16_t TarAddr,
const uint16_t NbByte )
```

Reads N bytes from the mailbox registers, starting at the specified I2C address.

Parameters

pData — Pointer on the buffer used to return the data.

TarAddr — I2C memory address to be read.

nbByte — Number of bytes to be read.

Returns: NFCTAG_StatusTypeDef enum status.

ST25DV_i2c_ReadMBCtrl_Dyn()

```
NFCTAG_StatusTypeDef ST25DV_i2c_ReadMBCtrl_Dyn (
ST25DV_MB_CTRL_DYN_STATUS const pCtrlStatus )
```

Reads the Mailbox ctrl dynamic register.

Parameter

pCtrlStatus — Pointer to an ST25DV_MB_CTRL_DYN_STATUS structure used to return the dynamic Mailbox ctrl information.

Returns: NFCTAG_StatusTypeDef enum status.

ST25DV_i2c_ReadMBLength_Dyn()

```
NFCTAG_StatusTypeDef ST25DV_i2c_ReadMBLength_Dyn (
uint8_t const pMBLength )
```

Reads the Mailbox message length dynamic register.

Parameter

pMBLength — Pointer to an uint8_t used to return the Mailbox message length.

Returns: NFCTAG_StatusTypeDef enum status.

ST25DV_i2c_ReadMBMode()

```
NFCTAG_StatusTypeDef ST25DV_i2c_ReadMBMode (
ST25DV_EN_STATUS const pMB_mode )
```

Reads the Mailbox mode.

Parameter

pMB_mode — Pointer to an ST25DV_EH_MODE_STATUS value used to return the Mailbox mode.

Returns: NFCTAG_StatusTypeDef enum status.

ST25DV_i2c_ReadMBWDG()

```
NFCTAG_StatusTypeDef ST25DV_i2c_ReadMBWDG (
uint8_t const pWdgDelay )
```

Reads the Mailbox watchdog duration coefficient.

Parameter

pWdgDelay — Pointer on a uint8_t used to return the watchdog duration coefficient.

Returns: NFCTAG_StatusTypeDef enum status.

ST25DV_i2c_ReadMemSize()

```
NFCTAG_StatusTypeDef ST25DV_i2c_ReadMemSize (
ST25DV_MEM_SIZE const pSizeInfo )
```

Reads the ST25DV Memory Size.

Parameter

pSizeInfo — Pointer to an ST25DV_MEM_SIZE structure used to return the Memory size information.

Returns: NFCTAG_StatusTypeDef enum status.

ST25DV_i2c_ReadRegister()

```
NFCTAG_StatusTypeDef ST25DV_i2c_ReadRegister (
uint8_t const pData,
const uint16_t TarAddr,
const uint16_t NbByte )
```

Reads N bytes from Registers, starting at the specified I2C address.

Parameters

pData — Pointer used to return the read data.

TarAddr — I2C memory address to be read.

NbByte — Number of bytes to be read.

Returns: NFCTAG_StatusTypeDef enum status.

ST25DV_i2c_ReadRFMngt()

```
NFCTAG_StatusTypeDef ST25DV_i2c_ReadRFMngt (
ST25DV_RF_MNGT const pRF_Mngt )
```

Reads the RF Management configuration.

Parameter

pRF_Mngt — Pointer to an ST25DV_RF_MNGT structure used to return the RF Management configuration.

Returns: NFCTAG_StatusTypeDef enum status.

ST25DV_i2c_ReadRFMngt_Dyn()

```
NFCTAG_StatusTypeDef ST25DV_i2c_ReadRFMngt_Dyn (
ST25DV_RF_MNGT const pRF_Mngt )
```

Reads the value of dynamic RF Management configuration.

Parameter

pRF_Mngt — ST25DV_RF_MNGT pointer of the dynamic RF Management configuration to store.

Returns: NFCTAG enum status

ST25DV_i2c_ReadRFZxSS()

```
NFCTAG_StatusTypeDef ST25DV_i2c_ReadRFZxSS (
const ST25DV_PROTECTION_ZONE Zone,
ST25DV_RF_PROT_ZONE const pRfprotZone )
```

Reads the RF Zone Security Status (defining the allowed RF accesses).

Parameters

Zone	ST25DV_PROTECTION_ZONE value corresponding to the protected area.
pRfprotZone	Pointer to an ST25DV_RF_PROT_ZONE value corresponding to the area protection state.

Returns: NFCTAG_StatusTypeDef enum status.

ST25DV_i2c_ReadUID()

```
NFCTAG_StatusTypeDef ST25DV_i2c_ReadUID (
ST25DV_UID const pUid )
```

Reads the ST25DV UID.

Parameter

pUid	Pointer used to return the ST25DV UID value.

Returns: NFCTAG_StatusTypeDef enum status.

ST25DV_i2c_ResetEHENMode_Dyn()

```
NFCTAG_StatusTypeDef ST25DV_i2c_ResetEHENMode_Dyn (
void )
```

Dynamically unsets the Energy Harvesting mode.

Returns: NFCTAG_StatusTypeDef enum status.

ST25DV_i2c_ResetGPO_en_Dyn()

```
NFCTAG_StatusTypeDef ST25DV_i2c_ResetGPO_en_Dyn (
void )
```

Reset dynamic GPO enable configuration.

Parameters: None

Returns: NFCTAG enum status.

ST25DV_i2c_ResetMBEN_Dyn()

```
NFCTAG_StatusTypeDef ST25DV_i2c_ResetMBEN_Dyn (
void )
```

Unsets the Mailbox Enable dynamic configuration.

Returns: NFCTAG_StatusTypeDef enum status.

ST25DV_i2c_ResetRFDisable()

```
NFCTAG_StatusTypeDef ST25DV_i2c_ResetRFDisable (
void )
```

Resets the RF Disable configuration.

Needs the I2C Password presentation to be effective.

Returns: NFCTAG_StatusTypeDef enum status.

ST25DV_i2c_ResetRFDisable_Dyn()

```
NFCTAG_StatusTypeDef ST25DV_i2c_ResetRFDisable_Dyn (
void )
```

Unsets the RF Disable dynamic configuration.

Returns: NFCTAG_StatusTypeDef enum status.

ST25DV_i2c_ResetRFSleep()

```
NFCTAG_StatusTypeDef ST25DV_i2c_ResetRFSleep (
void )
```

Resets the RF Sleep configuration.

Needs the I2C Password presentation to be effective.

Returns: NFCTAG_StatusTypeDef enum status.

ST25DV_i2c_ResetRFSleep_Dyn()

```
NFCTAG_StatusTypeDef ST25DV_i2c_ResetRFSleep_Dyn (
void )
```

Unsets the RF Sleep dynamic configuration.

Returns: NFCTAG_StatusTypeDef enum status.

ST25DV_i2c_SetEHENMode_Dyn()

```
NFCTAG_StatusTypeDef ST25DV_i2c_SetEHENMode_Dyn (
void )
```

Dynamically sets the Energy Harvesting mode.

Returns: NFCTAG_StatusTypeDef enum status.

ST25DV_i2c_SetGPO_en_Dyn()

```
NFCTAG_StatusTypeDef ST25DV_i2c_SetGPO_en_Dyn (
void )
```

Sets dynamic GPO enable configuration.

Parameters: None

Returns: NFCTAG enum status.

ST25DV_i2c_SetMBEN_Dyn()

```
NFCTAG_StatusTypeDef ST25DV_i2c_SetMBEN_Dyn (
void )
```

Sets the Mailbox Enable dynamic configuration.

Returns: NFCTAG_StatusTypeDef enum status.

ST25DV_i2c_SetRFDisable()

```
NFCTAG_StatusTypeDef ST25DV_i2c_SetRFDisable (
void )
```

Sets the RF Disable configuration.

Needs the I2C Password presentation to be effective.

Returns: NFCTAG_StatusTypeDef enum status.

ST25DV_i2c_SetRFDisable_Dyn()

```
NFCTAG_StatusTypeDef ST25DV_i2c_SetRFDisable_Dyn (
void )
```

Sets the RF Disable dynamic configuration.

Returns: NFCTAG_StatusTypeDef enum status.

ST25DV_i2c_SetRFSleep()

```
NFCTAG_StatusTypeDef ST25DV_i2c_SetRFSleep (
void )
```

Sets the RF Sleep configuration. Needs the I2C Password presentation to be effective.

Returns: NFCTAG_StatusTypeDef enum status.

ST25DV_i2c_SetRFSleep_Dyn()

```
NFCTAG_StatusTypeDef ST25DV_i2c_SetRFSleep_Dyn (
void )
```

Sets the RF Sleep dynamic configuration.

Returns: NFCTAG_StatusTypeDef enum status.

ST25DV_i2c_WriteData()

```
NFCTAG_StatusTypeDef ST25DV_i2c_WriteData (
const uint8_t const pData,
const uint16_t TarAddr,
const uint16_t NbByte )
```

Writes N bytes of Data starting from the specified I2C Address.

Parameters

pData	Pointer on the data to be written.
TarAddr	I2C data memory address to be written.
NbByte	Number of bytes to be written.

Returns: NFCTAG_StatusTypeDef enum status.

ST25DV_i2c_WriteEHMode()

```
NFCTAG_StatusTypeDef ST25DV_i2c_WriteEHMode (
const ST25DV_EH_MODE_STATUS EH_mode )
```

Sets the Energy harvesting mode. Needs the I2C Password presentation to be effective.

Parameter

EH_mode — ST25DV_EH_MODE_STATUS value for the Energy harvesting mode to be set.

Returns: NFCTAG_StatusTypeDef enum status.

ST25DV_i2c_WriteEndZonex()

```
NFCTAG_StatusTypeDef ST25DV_i2c_WriteEndZonex (
const ST25DV_END_ZONE EndZone,
const uint8_t EndZ )
```

Sets the end address of an area. Needs the I2C Password presentation to be effective.

Note: The ST25DV answers a NACK when setting the EndZone2 & EndZone3 to same value than repectively End Zone1 and EndZone2. These NACKs are ok.

Parameters

EndZone — ST25DV_END_ZONE value corresponding to an area.

EndZ — End zone value to be written.

Returns: NFCTAG_StatusTypeDef enum status.

ST25DV_i2c_WriteI2CPassword()

```
NFCTAG_StatusTypeDef ST25DV_i2c_WriteI2CPassword (
const ST25DV_PASSWD PassWord )
```

Writes a new I2C password. Needs the I2C Password presentation to be effective.

Parameter

PassWord — New I2C password value on 32bits.

Returns: NFCTAG_StatusTypeDef enum status.

ST25DV_i2c_WriteI2CProtectZonex()

```
NFCTAG_StatusTypeDef ST25DV_i2c_WriteI2CProtectZonex (
const ST25DV_PROTECTION_ZONE Zone,
const ST25DV_PROTECTION_CONF ReadWriteProtection )
```

Sets the I2C write-protected state to an EEPROM Area. Needs the I2C Password presentation to be effective.

Parameters

Zone — ST25DV_PROTECTION_ZONE value corresponding to the area to protect.

ReadWriteProtection — ST25DV_PROTECTION_CONF value corresponding to the protection to be set.

Returns: NFCTAG_StatusTypeDef enum status.

ST25DV_i2c_WriteITPulse()

```
NFCTAG_StatusTypeDef ST25DV_i2c_WriteITPulse (
const ST25DV_PULSE_DURATION ITtime )
```

Configures the ST25DV ITtime duration for the GPO pulse. Needs the I2C Password presentation to be effective.

Parameter

ITtime — Coefficient for the pulse duration to be written, equal to 302,06 µs- ITtime * 512 / fc)

Returns: NFCTAG_StatusTypeDef enum status.

ST25DV_i2c_WriteLockCCFile()

```
NFCTAG_StatusTypeDef ST25DV_i2c_WriteLockCCFile (
const ST25DV_CCFILE_BLOCK NbBlockCCFile,
const ST25DV_LOCK_STATUS LockCCFile )
```

Locks the CCFile to prevent any RF write access. Needs the I2C Password presentation to be effective.

Parameters

NbBlockCCFile — ST25DV_CCFILE_BLOCK value corresponding to the number of blocks to be locked.

LockCCFile — ST25DV_LOCK_CCFILE value corresponding to the lock state to apply on the CCFile.

Returns: NFCTAG_StatusTypeDef enum status.

ST25DV_i2c_WriteLockCFG()

```
NFCTAG_StatusTypeDef ST25DV_i2c_WriteLockCFG (
const ST25DV_LOCK_STATUS LockCfg )
```

Locks/unlocks the Cfg registers, to prevent any RF write access. Needs the I2C Password presentation to be effective.

Parameter

LockCfg — ST25DV_LOCK_STATUS value corresponding to the lock state to be written.

Returns: NFCTAG_StatusTypeDef enum status.

ST25DV_i2c_WriteMailboxData()

```
NFCTAG_StatusTypeDef ST25DV_i2c_WriteMailboxData (
const uint8_t const pData,
const uint16_t NbByte )
```

Writes N bytes of data in the Mailbox, starting from first Mailbox Address.

Parameters

pData — Pointer to the buffer containing the data to be written.

NbByte — Number of bytes to be written.

Returns: NFCTAG_StatusTypeDef enum status.

ST25DV_i2c_WriteMailboxRegister()

```
NFCTAG_StatusTypeDef ST25DV_i2c_WriteMailboxRegister (
const uint8_t const pData,
const uint16_t TarAddr,
const uint16_t NbByte )
```

Writes N bytes to the specified mailbox register.

Parameters

pData	Pointer on the data to be written.
TarAddr	I2C register address to be written.
NbByte	Number of bytes to be written.

Returns: NFCTAG_StatusTypeDef enum status.

ST25DV_i2c_WriteMBMode()

```
NFCTAG_StatusTypeDef ST25DV_i2c_WriteMBMode (
const ST25DV_EN_STATUS MB_mode )
```

Sets the Mailbox mode. Needs the I2C Password presentation to be effective.

Parameter

MB_mode	ST25DV_EN_STATUS value corresponding to the Mailbox mode to be set.

Returns: NFCTAG_StatusTypeDef enum status.

ST25DV_i2c_WriteMBWDG()

```
NFCTAG_StatusTypeDef ST25DV_i2c_WriteMBWDG (
const uint8_t WdgDelay )
```

Writes the Mailbox watchdog coefficient delay. Needs the I2C Password presentation to be effective.

Parameter

WdgDelay	Watchdog duration coefficient to be written, equal to MB_WDG * 30 ms +/- 6%.

Returns: NFCTAG_StatusTypeDef enum status.

ST25DV_i2c_WriteRegister()

```
NFCTAG_StatusTypeDef ST25DV_i2c_WriteRegister (
const uint8_t const pData,
const uint16_t TarAddr,
const uint16_t NbByte )
```

Writes N bytes to the specified register. Needs the I2C Password presentation to be effective.

Parameters

pData	Pointer on the data to be written.
TarAddr	I2C register address to written.
NbBYte	Number of bytes to be written.

Returns: NFCTAG_StatusTypeDef enum status.

ST25DV_i2c_WriteRFMngt()

```
NFCTAG_StatusTypeDef ST25DV_i2c_WriteRFMngt (
const uint8_t Rfmngt )
```

Sets the RF Management configuration. Needs the I2C Password presentation to be effective.

Parameter

Rfmngt	Value of the RF Management configuration to be written.

Returns: NFCTAG_StatusTypeDef enum status.

ST25DV_i2c_WriteRFMngt_Dyn()

```
NFCTAG_StatusTypeDef ST25DV_i2c_WriteRFMngt_Dyn (
const uint8_t RF_Mngt )
```

Writes a value to the RF Management dynamic register.

Parameter

RF_Mngt	Value to be written to the RF Management dynamic register.

Returns

NFCTAG_StatusTypeDef enum status.

ST25DV_i2c_WriteRFZxSS()

```
NFCTAG_StatusTypeDef ST25DV_i2c_WriteRFZxSS (
const ST25DV_PROTECTION_ZONE Zone,
const ST25DV_RF_PROT_ZONE RfProtZone )
```

Writes the RF Zone Security Status (defining the allowed RF accesses). Needs the I2C Password presentation to be effective.

Parameters

Zone	ST25DV_PROTECTION_ZONE value corresponding to the area on which to set the RF protection.
RfProtZone	Pointer to an ST25DV_RF_PROT_ZONE value defining the protection to be set on the area.

Returns: NFCTAG_StatusTypeDef enum status.

5.11.4 Variables documentation

St25Dv_i2c_Drv

```
NFCTAG_DrvTypeDef St25Dv_i2c_Drv
```

Standard NFC tag driver API for the ST25DV. Provides a generic way to access the ST25DV implementation of the NFC tag standard driver functions.

St25Dv_i2c_ExtDrv

NFCTAG_ExtDrvTypeDef St25Dv_i2c_ExtDrv

Extended NFC tag driver API for the ST25DV. Provides a generic way to access the ST25DV extended driver functions.

5.12 ST25DV menu definition

This module defines the structure and content of the ST25DV demonstration menu.

Table 19. ST25DV menu definition module - Overview

Section	Code	Description
Files	Menu_definition.c	Defines the content of the menu for the ST25DV demonstration.
Functions	`void Menu_Start (void)`	Starts the main loop for the demonstration menu.

5.12.1 Detailed description

This module defines the structure and content of the ST25DV demonstration menu. The menu structure is statically defined in the module, and complies with the expected structure of the menu_demo middleware. Call Menu_Start() to start the menu main loop.

5.13 ST25DV menu interface configuration

Interface file for the menu_demo middleware.

Table 20. ST25DV menu interface configuration module - Overview

Section	Code	Description
Files	Menu_config.c	Menu configuration file.
Functions	`uint16_t Menu_GetDisplayWidth (void)`	Interface function to retrieve the display width.
	`uint16_t Menu_GetDisplayHeight (void)`	Interface function to retrieve the display height.
	`void Menu_SetStyle (ColorStyles_t style)`	Interface function to set the display front and back colors.
	`uint32_t Menu_GetFontHeight (void)`	Interface function to get the font height.
	`uint32_t Menu_GetFontWidth (void)`	Interface function to get the font width.
	`void Menu_DisplayPicture (uint32_t PosX, uint32_t PosY, const char pict)`	Interface function to print a picture (bmp or jpeg) on the display.
	`void Menu_GetPictureDim (const char pict, uint32_t width, uint32_t height)`	Interface function to compute the dimensions of a picture.
	`void Menu_DisplayRectangle (uint32_t PosX, uint32_t PosY, uint32_t Height, uint32_t Width)`	Interface function to print a rectangle on the display.
	`void Menu_DisplayString (uint32_t Line, const char Str)`	Interface function to print a string on the display.
	`void Menu_DisplayChar (uint32_t Line, uint32_t Column, uint8_t Ascii)`	Interface function to print a string on the display.
	`void Menu_DisplayClear (void)`	Interface function to clear the display.
	`void Menu_DisplayClearLine (uint16_t Line)atency in the menu.`	Interface function to clear a line on display.
	`uint8_t Menu_ReadPosition (Menu_Position_t State)`	Interface function to get the position of a touch on a touchscreen.
	`uint8_t Menu_ReadDirection (Menu_Direction_t State)`	Interface function to get the direction and push (select) of a joystick.
	`uint8_t Menu_ReadSelection (uint8_t State)`	Interface function to get the state of a simple button.
	`void Menu_Delay (uint32_t duration)`	Interface function to add latency in the menu

5.13.1 Detailed description

This module is the interface file for the menu_demo middleware. It implements the functions for accessing the display and for reading user inputs (touchscreen, joystick and simple button).

5.13.2 Function documentation

Menu_Delay()

```
void Menu_Delay (
uint32_t duration )
```

Interface function to add latency in the menu.

Parameter

duration	Latency in ms

Menu_DisplayChar()

```
void Menu_DisplayChar (
uint32_t Line,
uint32_t Column,
uint8_t Ascii )
```

Interface function to print a string on the display.

Parameters

Line	Indicates the line number where the string as to be printed
Column	The position of the character in the line
Ascii	The character to be written

Menu_DisplayClearLine()

```
void Menu_DisplayClearLine (
uint16_t Line )
```

Interface function to clear a line on display.

Parameter

Line	Indicates the number of the line to clear

Menu_DisplayPicture()

```
void Menu_DisplayPicture (
uint32_t PosX,
uint32_t PosY,
const char  pict )
```

Interface function to print a picture (bmp or jpeg) on the display.

Parameters

PosX	Position of the top left corner of the picture on X-axis
PosY	Position of the top left corner of the picture on Y-axis
pict	Pointer to the picture address in memory

Menu_DisplayRectangle()

```
void Menu_DisplayRectangle (
```

```
uint32_t PosX,
uint32_t PosY,
uint32_t Height,
uint32_t Width )
```

Interface function to print a rectangle on the display.

Parameters

PosX	Position of the top left corner of the rectangle on the X-axis
PosY	Position of the top left corner of the rectangle on the Y-axis
Height	Rectangle height.
Width	Rectangle width.

Menu_DisplayString()

```
void Menu_DisplayString (
uint32_t Line,
const char  Str )
```

Interface function to print a string on the display.

Parameters

Line	Indicates the line number where the string as to be printed.
Str	String to be printed on the display.

Menu_GetDisplayHeight()

```
uint16_t Menu_GetDisplayHeight (
void )
```

Interface function to retrieve the display height.

Return value: Display height in pixels.

Menu_GetDisplayWidth()

```
uint16_t Menu_GetDisplayWidth (
void )
```

Interface function to retrieve the display width.

Return value: Display width in pixels.

Menu_GetFontHeight()

```
uint32_t Menu_GetFontHeight (
void )
```

Interface function to get the FONT height.

Return value: Font height in pixels.

Menu_GetFontWidth()

```
uint32_t Menu_GetFontWidth (
void )
```

Interface function to get the FONT width.

Return value: Font width in pixels.

Menu_GetPictureDim()

```
void Menu_GetPictureDim (
const char  pict,
uint32_t  width,
uint32_t  height )
```

Interface function to compute the dimensions of a picture.

Parameters

pict	Pointer to the picture address in memory
width	Picture position on the X-axis
height	Picture position on the Y-axis

Menu_ReadDirection()

```
uint8_t Menu_ReadDirection (
Menu_Direction_t  State )
```

Interface function to get the direction and push (select) of a joystick.

Parameter

State	Pointer on a Menu_Direction_t structure, used to return the up, down, left, right direction and selection of the joystick.

Return values

Null: the joystick is in neutral position

Not-null: the joystick is in a non neutral position.

Menu_ReadPosition()

```
uint8_t Menu_ReadPosition (
Menu_Position_t  State )
```

Interface function to get the position of a touch on a touchscreen.

Parameter

State	Pointer on a Menu_Position_t structure, used to return the X,Y coordinates of last detected touch.

Return values

Not-null: a touch has been detected

Null: no touches have been detected

Menu_ReadSelection()

```
uint8_t Menu_ReadSelection (
uint8_t  State )
```

Interface function to get the state of a simple button.

Parameter

State Pointer on a boolean, used to return the state of the button.

Return values

Null: the button is pushed

Not-null: the button is not pushed

Menu_SetStyle()

```
void Menu_SetStyle (
ColorStyles_t style )
```

Interface function to set the display front & back colors.

Parameter

style One of the values defined in enum ColorStyles_t..

5.14 LibJPEG decode wrapper

Wrapper calling the libJPEG STM32Cube middleware to decode JPEG pictures.

Table 21. LibJPEG decode wrapper module - Overview

Section	Code	Description
Files	jpeg_decode.c	Contains the JPEG decompressing methods.
Macros	`#define IS_JPEG(ptr) ((ptr[0] == 0xFF) && (ptr[1] == 0xD8))`	Macro checking if a pointer points to the beginning of a JPEG picture.
Functions	`void jpeg_decode (const char jpeg, uint8_t(callback)(uint8_t , uint32_t))`	Decodes a JPEG formatted picture calling the STM32Cube libJPEG middleware.
	`void jpeg_getsize (const char jpeg, uint32_t Width, uint32_t Height)`	Gets the geometry of a JPEG picture.
	`uint32_t jpeg_GetBufferSize (uint8_t jpeg)`	Gets not decompressed JPEG buffer size.
	`void jpeg_decode_exit (j_common_ptr cinfo)`	Callback for the libJPEG, executed when an unrecoverable error occurs.

5.14.1 Detailed description

Wrapper calling the libJPEG STM32Cube middleware to decode JPEG pictures. This module

- implements the function to decode a JPEG picture.
- implements the function to read picture geometry (height and width).
- displays errors when decoding fails.

It uses the decoding part of the libJPEG STM32Cube middleware, and the fs_api (to mimic file system functions).

5.14.2 Function documentation

jpeg_decode()

```
void jpeg_decode (
const char  jpeg,
uint8_t()(uint8_t , uint32_t) callback )
```

Decodes a jpeg formatted picture calling the STM32Cube libJPEG middleware.

Parameters

jpeg	Pointer to the data array with jpeg format picture
callback	Callback function to call after a line of the picture has been decoded

Return value: None

jpeg_decode_exit()

```
void jpeg_decode_exit (
j_common_ptr cinfo )
```

Callback for the libJPEG, executed when an unrecoverable error occured. Displays an error message with the details of the error.

Parameter

cinfo	Pointer to the jpeg_common_struct with the current libJPEG state

Return value: None

jpeg_GetBufferSize()

```
uint32_t jpeg_GetBufferSize (
uint8_t  jpeg )
```

Gets not decompressed Jpeg buffer size.

Parameter

jpeg	Pointer to the data array with jpeg format picture

Return value: Size of buffer in bytes.

jpeg_getsize()

```
void jpeg_getsize (
const char  jpeg,
uint32_t  Width,
uint32_t  Height )
```

Gets the geometry of a JPEG picture. Calls the STM32Cube libJPEG middleware to read the jpeg header and extract the geometry info.

Parameters

jpeg	Pointer to the data array with jpeg format picture
Width	Pointer used to return the width of the JPEG picture
Height	Pointer used to return the height of the JPEG picture

Return value: None

5.15 Simple file system API

Table 22. Simple file system API module - Overview

Section	Code	Description
Files	ff.h	Simple file system API header.
	fs_api.c	Simple file system API implementation.
Data structures	struct FIL	File object structure.
Macros	`#define FA_READ 0x01`	File access control: Read-Only (no other value supported).
Enumerations	`enum FRESULT { FR_OK = 0, FR_DENIED = 7, FR_INVALID_OBJECT = 9 }`	Function return codes.
Functions	`FRESULT f_open (FIL fp, const char start, uint8_t mode)`	Opens a stream of data from an address in memory.
	`FRESULT f_close (FIL fp)`	Closes a stream of data from an address in memory.
	`FRESULT f_read (FIL fp, void buff, uint32_t btr, uint32_t br)`	Reads bytes from a stream of data.
	`FRESULT f_write (FIL fp, const void buff, uint32_t btw, uint32_t bw)`	Does nothing.

5.15.1 Detailed description

Simple implementation of a file system API.

This module implements the functions to open, read and close a file. It does not implement a real file system. A file is considered being a starting address in memory, from where the API reads a stream of data. The purpose of this module is to provide a standard file system API to feed data to the libJPEG.

5.15.2 Data structure documentation

struct FIL

File object structure

Table 23. Data fields

Type	Field name	Description
`uint8_t`	open	Boolean, true if the file is currently opened.
`const char`	ptr	Pointer to the next data to be read.

5.15.3 Enumeration type

FRESULT

`enum FRESULT`

Function return codes

Enumerator

FR_OK	(0) Succeeded
FR_DENIED	(7) Access denied due to prohibited access
FR_INVALID_OBJECT	(9) The file object is invalid

5.15.4 Function documentation

f_close()

```
FRESULT f_close (
FIL  fp )
```

Closes a stream of data from an address in memory. This function mimics the close standard function from a file system. The usage of such function in the ST25DV demonstration is to feed data to the jpeg decoder module, without having a real file system.

Parameter

fp	Pointer to FIL structure - mimics a filehandle

Return values:

FR_OK: Data stream has been successfully closed.

FR_INVALID_OBJECT: fp is not allocated or not open.

f_open()

```
FRESULT f_open (
FIL  fp,
const char  start,
uint8_t mode )
```

Opens a stream of data from an address in memory. This function mimics the open standard function from a file system. The usage of such function in the ST25DV demonstration is to feed data to the jpeg decoder module, without having a real file system.

Parameters

fp	Pointer to FIL structure - mimics a filehandle
start	Pointer to an address in memory, that will be used as source of the data
mode	File access control, only FA_READ is supported (read-only)

Return values:

FR_OK: Data stream is open

FR_INVALID_OBJECT: fp is not allocated/initialized.

FR_DENIED: mode is different from FA_READ.

f_read()

```
FRESULT f_read (
FIL  fp,
```

```
void  buff,
uint32_t btr,
uint32_t  br )
```

Reads bytes from a stream of data. This function mimic the read standard function from a file system. The usage of such function in the ST25DV demonstration is to feed data to the jpeg decoder module, without having a real file system.

Parameters

fp	Pointer to FIL structure - mimics a filehandle
buff	Memory buffer to store read data
btr	Number of bytes to read
br	Number of bytes actually read

Return values

FR_OK: Data stream has been successfully read.

FR_INVALID_OBJECT: fp is not allocated or not open.

f_write()

```
FRESULT f_write (
FIL  fp,
const void  buff,
uint32_t btw,
uint32_t  bw )
```

Does nothing. This function mimics the write standard function from a file system. The usage of such function in the ST25DV demonstration is to feed data to the jpeg decoder module, without having a real file system.

Parameters

fp	Pointer to FIL structure - mimics a filehandle
buff	Memory buffer with source of data
btr	Number of bytes to read
br	Number of bytes actually read

Return value

FR_OK: Always.

5.16 ST25 Discovery interrupt routines

This module defines all the required interrupt routines for the ST25DV demo.

Table 24. ST25 Discovery interrupt routines module - Overview

Section	Code	Description
File	stm32f4xx_it.c	Main Interrupt Service Routines.
Modules	Cortex®-M4 exceptions handlers	Defines handlers for the Cortex®-M4 hardware exceptions
Functions	`void SysTick_Handler (void)`	Handles System Tick Handler.
	`void TIMx_IRQHandler (void)`	Handles TIM interrupt request.
	`void TIMp_IRQHandler (void)`	Handles TIM interrupt request.
	`void HAL_TIM_PeriodElapsedCallback (TIM_HandleTypeDef htim)`	Period elapsed callback in non blocking mode This timer is used for calling back User registered functions with information.
	`void BNRG_SPI_EXTI_IRQHandler (void)`	Handles External line interrupt request for BlueNRG Bluetooth module.
	`void USART3_IRQHandler (void)`	Handles USART3 interrupts for the Wi-Fi module.
	`void NFCMEM_GPO_EXTIHandler (void)`	Handles External line 0 interrupt request.
	`void HAL_GPIO_EXTI_Callback (uint16_t GPIO_Pin)`	Handles callback from external line interrupt request
Variables	`volatile uint32_t ms_counter = 0`	Current millisecond count.
	`volatile uint8_t GPO_Activated = 0`	Polling variable for the ST25DV GPO interrupt, updated from GPO interrupt callback.
	`TIM_HandleTypeDef TimHandle`	Timer handle from st25_spwf_wifi.c.
	`TIM_HandleTypeDef PushTimHandle`	Timer handle from st25_spwf_wifi.c.
	`UART_HandleTypeDef UartWiFiHandle`	UART handle from wifi_module.c.

5.16.1 Detailed description

This module defines all the required interrupt routines for the ST25DV demonstration.

Interrupts routines used in the ST25DVdemo:

- System tick (ms timer)
- SPI3 for the BlueNRG Bluetooth module
- USART3 for the Wi-Fi module
- GPO interrupt from ST25DV

Note: The hardware exception handlers are defined in the same file, but are documented in a sub-module to improve readability.

5.16.2 Function documentation

HAL_GPIO_EXTI_Callback()

```
void HAL_GPIO_EXTI_Callback (
```

```
uint16_t GPIO_Pin )
```

This function handles callback from external line interrupt request.

Parameter

GPIO_Pin

HAL_TIM_PeriodElapsedCallback()

```
void HAL_TIM_PeriodElapsedCallback (
TIM_HandleTypeDef  htim )
```

Period elapsed callback in non blocking mode This timer is used for calling back User registered functions with information.

Parameter

htim TIM handler

SysTick_Handler()

```
void SysTick_Handler (
void )
```

This function handles System Tick Handler.

Parameters: None

Return values: None

TIMp_IRQHandler()

```
void TIMp_IRQHandler (
void )
```

This function handles TIM interrupt request.

Parameters: None

Return values: None

TIMx_IRQHandler()

```
void TIMx_IRQHandler (
void )
```

This function handles TIM interrupt request.

Parameters: None

Return values: None

5.17 Cortex®-M4 exceptions handlers

This module defines handlers for the Cortex®-M4 hardware exceptions.

Table 25. Cortex®-M4 exceptions handlers module - Overview

Section	Code	Description
Files	stm32f4xx_it.c	Main Interrupt Service Routines.
Functions	`void NMI_Handler (void)`	Handles NMI exception.
	`void HardFault_Handler (void)`	Handles Hard Fault exception.
	`void MemManage_Handler (void)`	Handles Memory Manage exception.
	`void BusFault_Handler (void)`	Handles Bus Fault exception.
	`void UsageFault_Handler (void)`	Handles Usage Fault exception.
	`void SVC_Handler (void)`	Handles SVCall exception.
	`void DebugMon_Handler (void)`	Handles Debug Monitor exception.
	`void PendSV_Handler (void)`	Handles PendSVC exception.

5.17.1 Detailed description

This module defines handlers for the Cortex®-M4 hardware exceptions. Hardware exception handlers are called when a specific situation occurs on the HW, such as unknown instruction or bus error.

5.17.2 Function documentation

BusFault_Handler()

```
void BusFault_Handler (
void )
```

This function handles Bus Fault exception.

Parameters: None

Return values: None

DebugMon_Handler()

```
void DebugMon_Handler (
void )
```

This function handles Debug Monitor exception.

Parameters: None

Return values: None

HardFault_Handler()

```
void HardFault_Handler (
void )
```

This function handles Hard Fault exception.

Parameters: None

Return values: None

MemManage_Handler()

```
void MemManage_Handler (
void )
```

This function handles Memory Manage exception.

Parameters: None

Return values: None

NMI_Handler()

```
void NMI_Handler (
void )
```

This function handles NMI exception.

Parameters: None

Return values: None

PendSV_Handler()

```
void PendSV_Handler (
void )
```

This function handles PendSVC exception.

Parameters: None

Return values: None

SVC_Handler()

```
void SVC_Handler (
void )
```

This function handles SVCall exception.

Parameters: None

Return values: None

UsageFault_Handler()

```
void UsageFault_Handler (
void )
```

This function handles Usage Fault exception.

Parameters: None

Return values: None

5.18 ST25 Discovery MCU support package

This module defines the MCU init routines for some of the ST25 Discovery peripherals.

Table 26. ST25DV Discovery MSP module - Overview

Section	Code	Description
Files	stm32f4xx_hal_msp.c	Provides the code for the MSP initialization and de-initialization.
Functions	`void HAL_MspInit (void)`	De-initializes the Global MSP (do nothing).
	`void HAL_ADC_MspInit (ADC_HandleTypeDef hadc)`	Initializes the low level hardware: GPIO, CLOCK, NVIC for ADC.
	`void HAL_ADC_MspDeInit (ADC_HandleTypeDef hadc)`	De-initializes the low level hardware: GPIO, CLOCK, NVIC for ADC.
	`void HAL_SPI_MspInit (SPI_HandleTypeDef hspi)`	Initializes the low level hardware: GPIO, CLOCK, NVIC for SPI.
	`void HAL_SPI_MspDeInit (SPI_HandleTypeDef hspi)`	De-initializes the low level hardware: GPIO, CLOCK, NVIC for SPI.
	`void HAL_UART_MspInit (UART_HandleTypeDef huart)`	Initializes the low level hardware: GPIO, CLOCK, NVIC for UART.
	`void HAL_UART_MspDeInit (UART_HandleTypeDef huart)`	De-initializes the low level hardware: GPIO, CLOCK, NVIC for UART.
	`void HAL_CRC_MspInit (CRC_HandleTypeDef hcrc)`	Initializes the low level hardware: GPIO, CLOCK, NVIC for CRC.
	`void HAL_CRC_MspDeInit (CRC_HandleTypeDef hcrc)`	De-initializes the low level hardware: GPIO, CLOCK, NVIC for CRC.

5.18.1 Detailed description

This module defines the MCU init routines for some of the ST25 Discovery peripherals.

MCU Support Package initialization consists in a system configuration (GPIO, Clocks, NVIC and DMA) for the peripherals.

These routines are callbacks called from the peripheral initialization (as described by the STM32Cube methodology).

This module initializes the system for the following peripherals:

- Analog to Digital Converter
- SPI3 for BlueNRG Bluetooth module communication
- USART3 for Wi-Fi module communication

Note: The MSP init for the peripherals used by the STM32Cube drivers (known as IOBus in STM32Cube) are statically defined in the Board Support Package, as recommended by STM32Cube guidelines.

5.18.2 Function documentation

HAL_ADC_MspDeInit()

void HAL_ADC_MspDeInit (

ADC_HandleTypeDef hadc)

De-initializes the low level hardware: GPIO, CLOCK, NVIC for ADC.

Parameter

hadc pointer to an ADC_HandleTypeDef structure that contains the configuration information for ADC module

ADC1 GPIO Configuration PC5 → ADC1_IN15 PB0 → ADC1_IN8 PB1 → ADC1_IN9

HAL_ADC_MspInit()

```
void HAL_ADC_MspInit (
ADC_HandleTypeDef  hadc )
```

Initializes the low level hardware: GPIO, CLOCK, NVIC for ADC.

Parameter

hadc pointer to an ADC_HandleTypeDef structure that contains the configuration information for ADC module

ADC1 GPIO Configuration PC5 → ADC1_IN15 PB0 → ADC1_IN8 PB1 → ADC1_IN9

HAL_CRC_MspDeInit()

```
void HAL_CRC_MspDeInit (
CRC_HandleTypeDef  hcrc )
```

DeInitializes the low level hardware: GPIO, CLOCK, NVIC for CRC.

Parameter

hcrc pointer to a CRC_HandleTypeDef structure that contains the configuration information for CRC module

Return values: None

HAL_CRC_MspInit()

```
void HAL_CRC_MspInit (
CRC_HandleTypeDef  hcrc )
```

Initializes the low level hardware: GPIO, CLOCK, NVIC for CRC.

Parameter

hcrc pointer to a CRC_HandleTypeDef structure that contains the configuration information for CRC module

Return values: None

HAL_MspInit()

```
void HAL_MspInit (
void )
```

Initializes the Global MSP (do nothing...).

Empty function.

HAL_SPI_MspDeInit()

```
void HAL_SPI_MspDeInit (
SPI_HandleTypeDef  hspi )
```

De-initializes the low level hardware: GPIO, CLOCK, NVIC for SPI.

Parameter

hspi — pointer to an SPI_HandleTypeDef structure that contains the configuration information for SPI module

SPI1 GPIO Configuration PA4 → SPI1_NSS PA5 → SPI1_SCK PA6 → SPI1_MISO PA7 → SP I1_MOSI

SPI2 GPIO Configuration PC2 → SPI2_MISO PC3 → SPI2_MOSI PB12 → SPI2_NSS PB13 → SPI2_SCK

SPI3 GPIO Configuration PC10 → SPI3_SCK PC11 → SPI3_MISO PC12 → SPI3_MOSI

HAL_SPI_MspInit()

```
void HAL_SPI_MspInit (
SPI_HandleTypeDef  hspi )
```

Initializes the low level hardware: GPIO, CLOCK, NVIC for SPI.

Parameter

hspi — pointer to an SPI_HandleTypeDef structure that contains the configuration information for SPI module

SPI3 GPIO Configuration PC10 → SPI3_SCK PC11 → SPI3_MISO PC12 → SPI3_MOSI

HAL_UART_MspDeInit()

```
void HAL_UART_MspDeInit (
UART_HandleTypeDef  huart )
```

De-initializes the low level hardware: GPIO, CLOCK, NVIC for UART.

Parameter

huart — pointer to an UART_HandleTypeDef structure that contains the configuration information for UART module

USART2 GPIO Configuration PA0-WKUP → USART2_CTS PA1 → USART2_RTS PA2 → USART2_TX PA3 → USART2_RX

USART3 GPIO Configuration PB14 → USART3_RTS PD8 → USART3_TX PD9 → USART3_RX PD11 → USART3_CTS

USART6 GPIO Configuration PC6 → USART6_TX PC7 → USART6_RX

HAL_UART_MspInit()

```
void HAL_UART_MspInit (
UART_HandleTypeDef  huart )
```

Initializes the low level hardware: GPIO, CLOCK, NVIC for UART.

Parameter

huart pointer to an UART_HandleTypeDef structure that contains the configuration information for UART module

USART2 GPIO Configuration PA0-WKUP → USART2_CTS PA1 → USART2_RTS PA2 → USART2_TX PA3 → USART2_RX

USART3 GPIO Configuration PB14 → USART3_RTS PD8 → USART3_TX PD9 → USART3_RX PD11 → USART3_CTS

USART6 GPIO Configuration PC6 → USART6_TX PC7 → USART6_RX

6 Revision history

Table 27. Document revision history

Date	Revision	Changes
21-Feb-2017	1	Initial release.

IMPORTANT NOTICE – PLEASE READ CAREFULLY

STMicroelectronics NV and its subsidiaries (“ST”) reserve the right to make changes, corrections, enhancements, modifications, and improvements to ST products and/or to this document at any time without notice. Purchasers should obtain the latest relevant information on ST products before placing orders. ST products are sold pursuant to ST’s terms and conditions of sale in place at the time of order acknowledgement.

Purchasers are solely responsible for the choice, selection, and use of ST products and ST assumes no liability for application assistance or the design of Purchasers’ products.

No license, express or implied, to any intellectual property right is granted by ST herein.

Resale of ST products with provisions different from the information set forth herein shall void any warranty granted by ST for such product.

ST and the ST logo are trademarks of ST. All other product or service names are the property of their respective owners.

Information in this document supersedes and replaces information previously supplied in any prior versions of this document.

© 2017 STMicroelectronics – All rights reserved