UM3019

User manual

Getting started with the STM32Cube function pack for IoT tracker node with Bluetooth® Low Energy, LoRa, NFC connectivity, GNSS and sensors

Introduction

The FP-ATR-ASTRA1 is an STM32Cube function pack that implements a complete asset tracking application, which supports long-range connectivity and short-range connectivity. This application reads the data from the environmental and motion sensors, retrieves the geo-position from GNSS and sends them to the cloud using Bluetooth® Low Energy and LoRaWAN® connectivity.

The FP-ATR-ASTRA1 package supports low-power profiles and the related transitions to ensure long battery autonomy. Moreover, it provides key features such as secure element management, the possibility to add custom algorithms, debugging interfaces, and expansion capability.

The firmware is available as a standard source code .zip file and as an STM32CubeMX pack in the STM32CubeMX library. Thus, to simplify the FP-ATR-ASTRA1 customization, the application firmware has been fully integrated in STM32CubeMX.

The user can select the desired use cases and configure parameters inside STM32CubeMX, generating the final complete application without needing a deep knowledge of the source code.

1 FP-ATR-ASTRA1 software expansion for STM32Cube

1.1 Overview

The FP-ATR-ASTRA1 software package expands STM32Cube functionality.

The key features of the package are:

- Complete asset tracking firmware application to manage long-range connectivity (LoRaWAN) and short-range connectivity (Bluetooth® Low Energy and NFC)
- Environmental and motion sensors management to monitor asset status
- Outdoor localization and geo-fencing based on a GNSS NMEA string available on Teseo-LIV3F
- Secure element personalization and certificate retrieving
- Dynamic NFC/RFID EEPROM memory management which allow easy provisioning, storing configuration parameters and collecting sensors data
- Power/battery management with low-power operating modes
- Flexible state machine to support different use cases
- Predefined customizable use cases:
  - Fleet management
  - Livestock monitoring
  - Goods monitoring
  - Logistics
  - Custom
- Implementation available for the STEVAL-ASTRA1B
- Package compatible with STM32CubeMX. It can be downloaded from and installed directly into STM32CubeMX
- Fully integrated in an end-to-end, proof-of-concept ecosystem, which includes:
  - the DSH-ASSETRACKING web cloud dashboard
  - the STAssetTracking mobile app available on Google Play and App Store

1.2 Architecture

FP-ATR-ASTRA1 is developed to support all the STEVAL-ASTRA1B functionalities.

The FP-ATR-ASTRA1, compliant with the STM32Cube architecture, is structured into a set of layers of increasing abstraction.

The hardware abstraction layer (HAL) interfaces with the hardware. It provides the low-level drivers and the hardware interface methods to interact with the upper layers (application, libraries, and stacks). It also provides the APIs for the communication peripherals (I²C, SPI, UART, etc.) for initialization and configuration, data transfer, and communication errors.

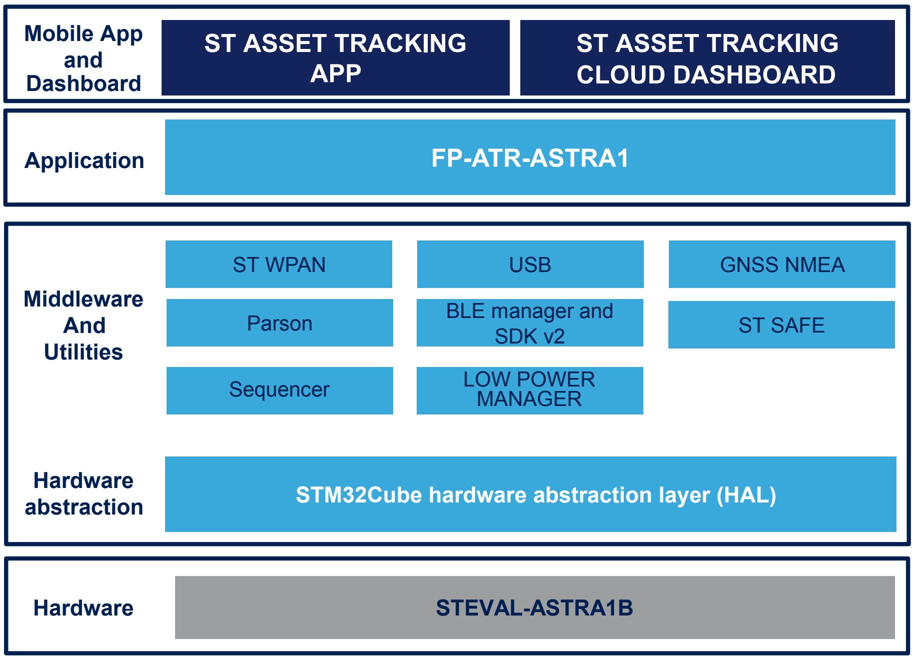
The package provides a board support package (BSP), which deals with the board-specific peripherals and functions (LED, user button, etc.). The BSP structure follows the hardware structure, including a component management layer as well as the specific layers of the board used. The modules included in the BSP are selected according to the used hardware configuration.

Middleware and Utilities provide advanced libraries and protocols for the USB communication, STM32-WPAN, GNSS NMEA, Bluetooth® Low Energy manager, STSAFE, sequencer, and low-power manager.

The horizontal interaction among the layer components is handled directly by calling the feature APIs. The vertical interaction with the low-level drivers is managed through specific callbacks and static macros implemented in the library system call interface.

On top, the application layer contains functions and procedures that characterize the application and can be changed by the end user.

Figure 1. **FP-ATR-ASTRA1 software architecture**

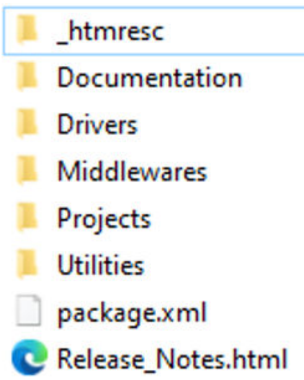


1.3 Folder structure

The software package includes the following folders:

- *Documentation*: contains a compiled HTML file generated from the source code, which details the software components and APIs.
- *Drivers*: contains the HAL drivers and the board-specific drivers for each supported board or hardware platform, including the on-board components and the CMSIS vendor-independent hardware abstraction layer for the Arm Cortex®-M processor series.
- *Middleware*: contains libraries and protocols for the USB communication, STM32-WPAN, GNSS NMEA, Parson library, Bluetooth® Low Energy manager, STSAFE, sequencer, and low-power manager.
- *Projects*: contains sample application used to implement asset tracking examples.
This application is provided for the STEVAL-ASTRA1B evaluation kit with three development environments (IAR Embedded Workbench for Arm, MDK-ARM, and STM32CubeIDE).

Figure 2. **FP-ATR-ASTRA1 folder structure**



1.4 APIs

Detailed technical information with full user API function and parameter description is in a compiled HTML file in the “Documentation” folder.

2 Sample application description

A standard asset tracking routine acquires information about location, sensor and system status. Then, the routine sends the information acquired through the available communication interfaces with the minimum power consumption.

Different use cases require the customization of these routines in terms of:

- acquisition frequency
- sending interval
- algorithms to run in order to add specific features
- localization technologies
- communication interfaces

The FP-ATR-ASTRA1 offers the following preconfigured (and customizable) use cases:

1. fleet management
2. livestock monitoring
3. goods monitoring
4. logistics
5. custom

The characteristics of these use cases are listed in the table below.

Table 1. Use cases

Uses cases	LoRa sending interval	Bluetooth® Low Energy sending interval	Indoor only (GNSS disabled)	Algorithm(1)	Data logging(2)
Fleet management	40 s	100 ms	N	No	Yes
Livestock monitoring	60 s	500 ms	N	Motion	No
Goods monitoring	3600 s	200 ms	Y	Geofence	Yes
Logistics	20 s	1000 ms	N	No	Yes
Custom	30 s	400 ms	N	No	Yes

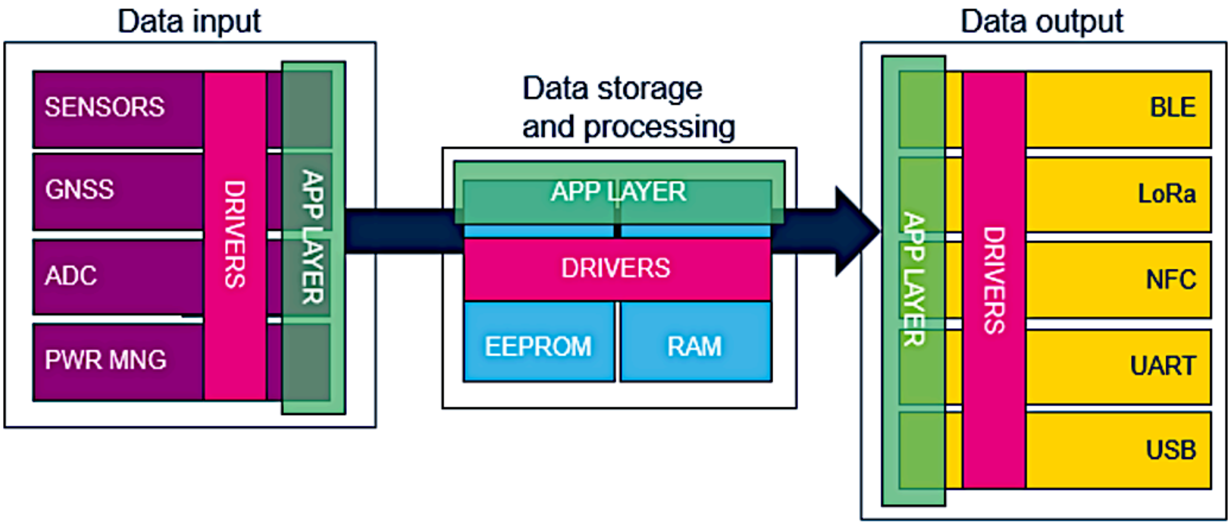
1. *Only an empty callback is available. It has to be filled by the user. See the `manageAlgorithms( )` function in the source code.*
2. *The "LOG" flag is available in the source code. The data logging is going to be available in the coming releases. See `AstraEngineParams_t typedef` in the source code.*

2.1 Firmware library structure

The FP-ATR-ASTRA1 firmware flexibility allows changing the operation mode easily. The state machine can change states and transitions according to the selected use case.

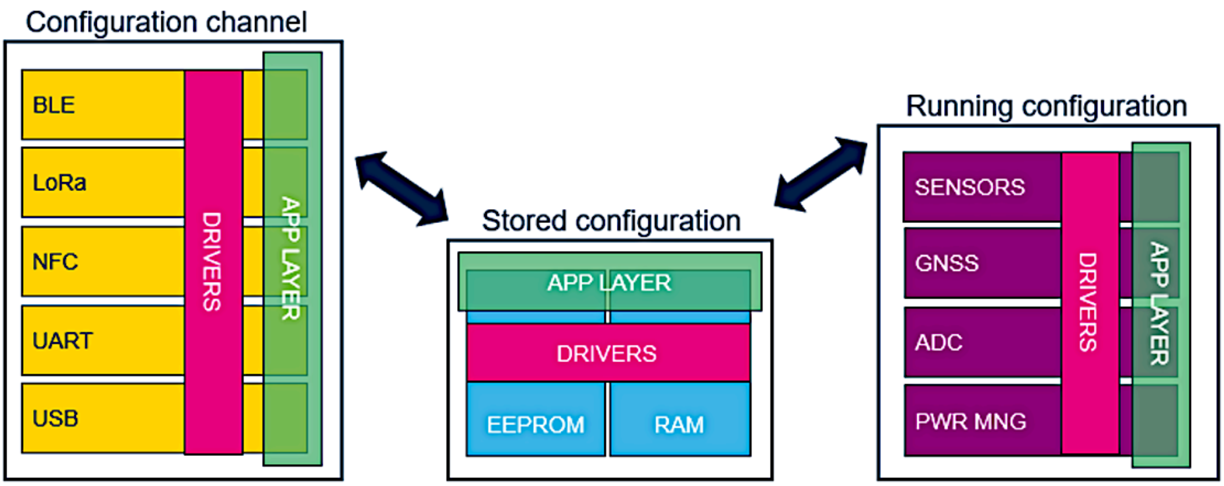
The firmware library structure is modular to meet these requirements. The figure below shows how the data flow from the input to the output.

Figure 3. **Data flow diagram**



The data acquired from the sensors, GNSS, power management circuit, and ADC through the driver interface, are stored in the RAM and/or EEPROM. Then, they are processed and sent. The processing output can be sent as well. The same path is used for the configuration channels, shown in the figure below, to manage the settings at runtime.

Figure 4. **Configuration block diagram**



2.2 State machine

The application flow evolves by using a state machine that triggers a status change depending on events and triggers.

A simple state machine has been implemented featuring two states: full-run state and low-power state. These states are not only related to the MCU power state, but to the whole system condition. A minimum and a maximum performance and, consequently, power consumption characterize these states. In the full-run state, everything is on and all the data are sent to the dashboard. In the low-power state, all the components, are configured in low-power mode.

The side button press event triggers the transition between the two states. Other triggers can be MEMS event output or the result of an algorithm or the BLE connection.

The sample application described in Figure 6. State machine flow features a typical asset tracking application switching between low power and full-run: starting from full-run with sensors and communication interface operating, when the accelerometer reveals no motion the application goes in low power; the system will be woken up from low power by BLE connection or accelerometer motion detection. The side button short press is also able to wake up the system and put it in low power.

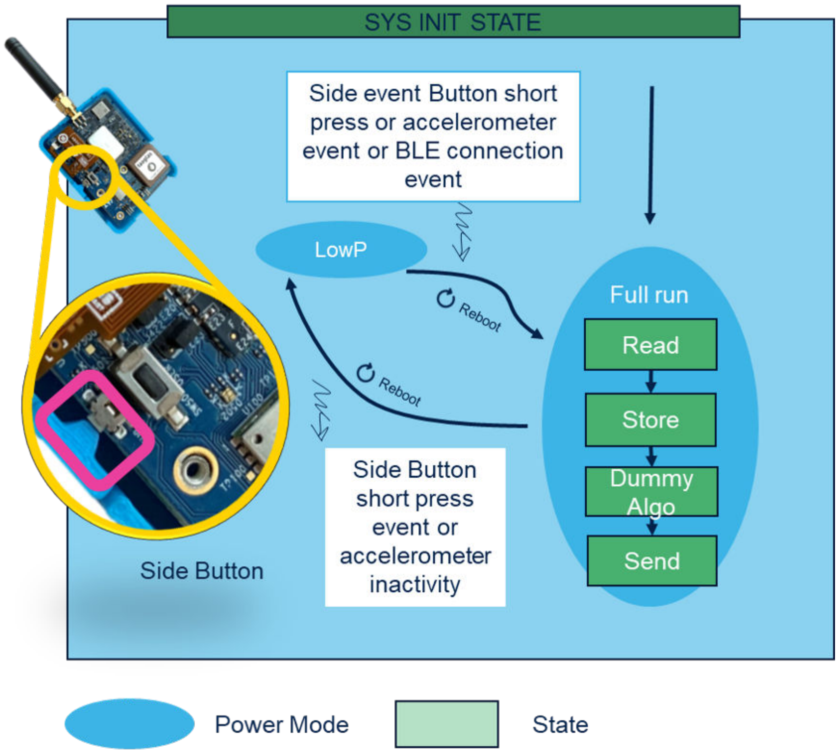
This is just an example of how a state machine can be implemented, changing the behavior of our device.

The power consumption in full-run mode may vary, depending on the transmission rate, GPS signal strength, etc. It is in the order of tenth of milliamperes. The current consumption in low power (see note) is in the order of microamperes.

Several intermediate states can also be implemented by the user, balancing the system responsiveness and the battery life.

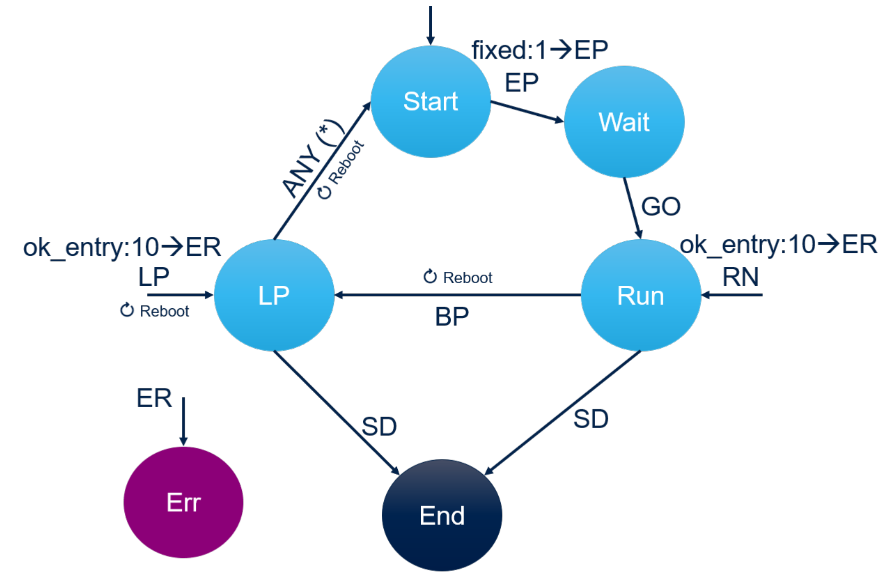
Note: *In the STEVAL-ASTRA1B, the current consumption in low power is optimized by removing SB943.*

Figure 5. State machine-functional diagram



The figure below shows the detailed flow diagram of the process, referring to the code implementation.

Figure 6. **State machine flow**



(*) Wake up sources are configured by application layer

The states are:

- *Start*
- *Run*
- *LP* (low-power)
- *End*
- *Wait*
- *Err*

Figure 7. **State description**

	Start	Run	LP	End	Wait	Err
Power	StartPowerFn	RunPowerFn	null	SD_MD_Func	null	null
USB	StartUsbFn	RunUsbFn	null	null	null	null
Button	StartButtonFn	RunButtonFn	null	null	null	null
Memory	StartMemFn	RunMemFn	null	null	null	null
Sensors	StartSensFn	RunSensFn	null	null	null	null
GNSS	StartGnssFn	RunGnssFn	null	null	null	null
Application	ApplicationFn	ApplicationFn	LpBleFn	ApplicationFn	WaitFn	ApplicationFn
Buzzer	StartBuzzerFn	RunBuzzerFn	null	null	null	null
BLE	StartBleFn	RunBleFn	null	null	null	null
LoRa	StartLoraFn	RunLoraFn	null	null	null	null
Security	StartSecurFn	RunSecurFn	null	null	null	null

The following callbacks describe each state:

- `StartXXXFn` is the function that turns the module on in the start state;
- `RunXXXFn` is the function that executes the module in the run state (all the modules are running);
- `LpXXXFn` is the function that executes the module in the low-power state (GNSS, memory, and sensors are turned off, while the other modules are in low power).

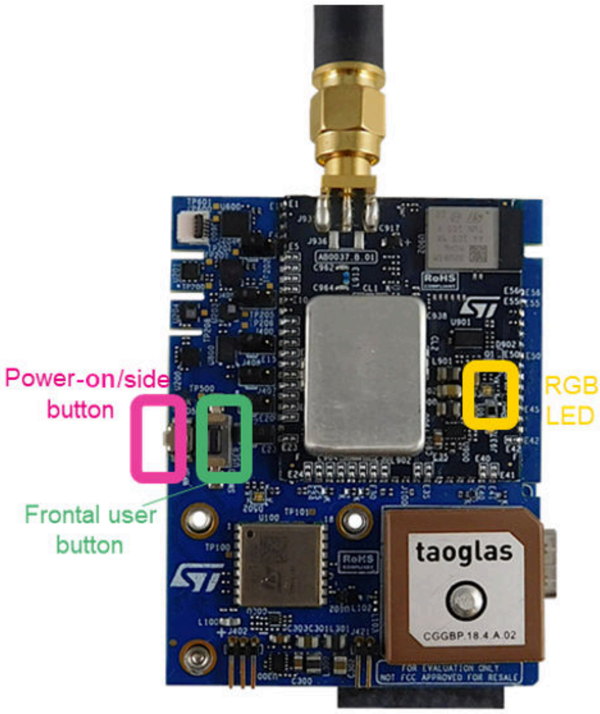
The events are:
- *BP*: button pressed event;
- *SD*: shutdown event;
- *ER*: error event;
- *GO*: to exit from WAIT state;
- *EP*: automatic transition to next step;
- *RN*: go to full-run command
- *LP*: go to low-power command.

See the SM_APP.c file for further details.

2.3 HMI: LEDs and buttons

The STEVAL-ASTRA1B is provided with an RBG user LED and two user buttons:
- power-on side button
- frontal user button

Figure 8. STEVAL-ASTRA1B LEDs and buttons placement



The FP-ATR-ASTRA1B implements simple HMI functionalities using:

- the RGB LED, which turns:
  - blue, if the configuration is ongoing
  - red, if the sending is ongoing
  - green:
    - with slow blinking, if the Bluetooth® Low Energy is not connected
    - with fast blinking, if the Bluetooth® Low Energy is connected
  - yellow, if the Bluetooth® Low Energy is connected and sending
  - magenta, if data are stored into the NFC memory
- the frontal user button:
  - short press: triggers asynchronous data sending
  - long press: system shutdown
- the side button:
  - first press: power-on
  - short press: changes the system status from run to low power and vice versa
  - long press: system reboot

3 FP-ATR-ASTRA1 in STM32CubeMX

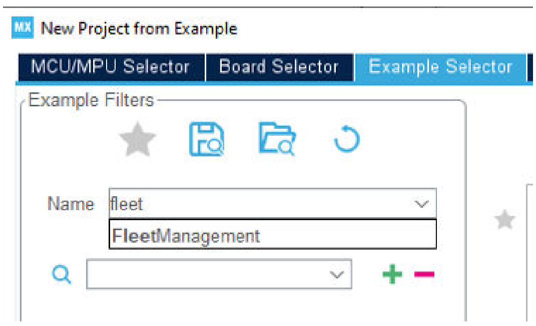
To simplify the customization of the FP-ATR-ASTRA1 function pack, the firmware has been fully integrated in STM32CubeMX. In this way, with a graphical user interface, you can change some parameters or activate some components through the STM32CubeMX interface. There are two options for customization:

1. the source code in a zip file for massive modifications or to dig into the code
2. the STM32CubeMX pack integration that helps to speed up the customization through its graphical interface; it generates a firmware library in source code with the required elements, which reflect the choices done in the GUI for the specific desired use case.

The FP-ATR-ASTRA1 offers several levels of integration in STM32CubeMX:

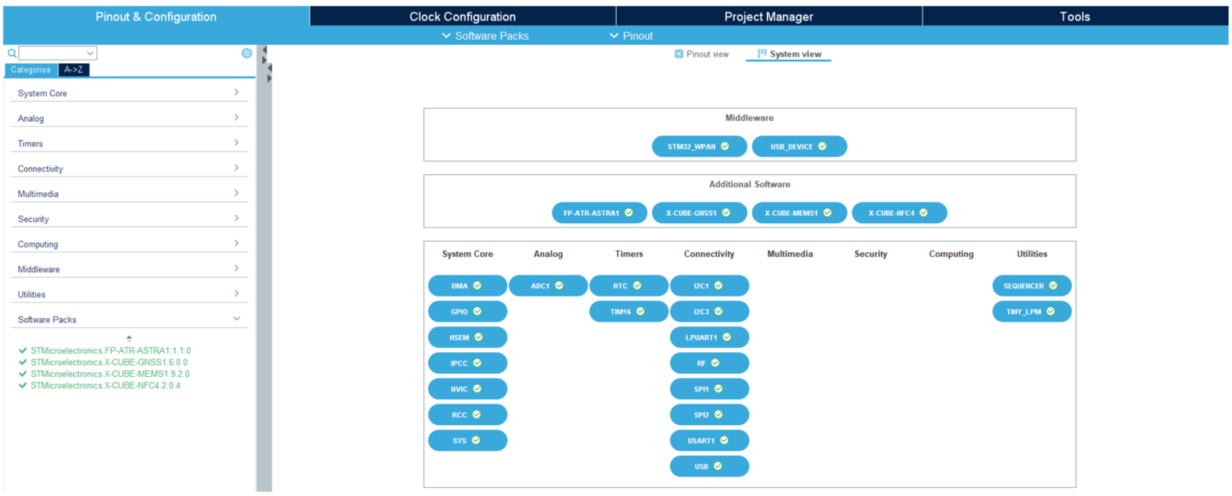
- Hardware level, by accessing the STM32CubeMX board selector: to configure the STM32WB5MMG GPIOs and peripherals, according to their usage on STEVAL-ASTRA1B, and to generate the initialization code to start the user code customization.
- Application level: the STM32CubeMX pack folder contains some preconfigured.ioc files. These files implement four typical use cases and a custom one. The project settings can be customized and a full application is generated. It is ready to be compiled and loaded into the STM32WB5MMG of the STEVAL-ASTRA1B, running a complete asset tracking application.
- Application Level: Access the CubeMX “example selector” to create a project based on asset tracking examples:

Figure 9. **Search result for Fleet management application in the CubeMX example selector**



The FP-ATR-ASTRA1 pack has been designed to generate a complete firmware application. The figure below shows the overall architecture.

Figure 10. **FP-ATR-ASTRA1 pack architecture**

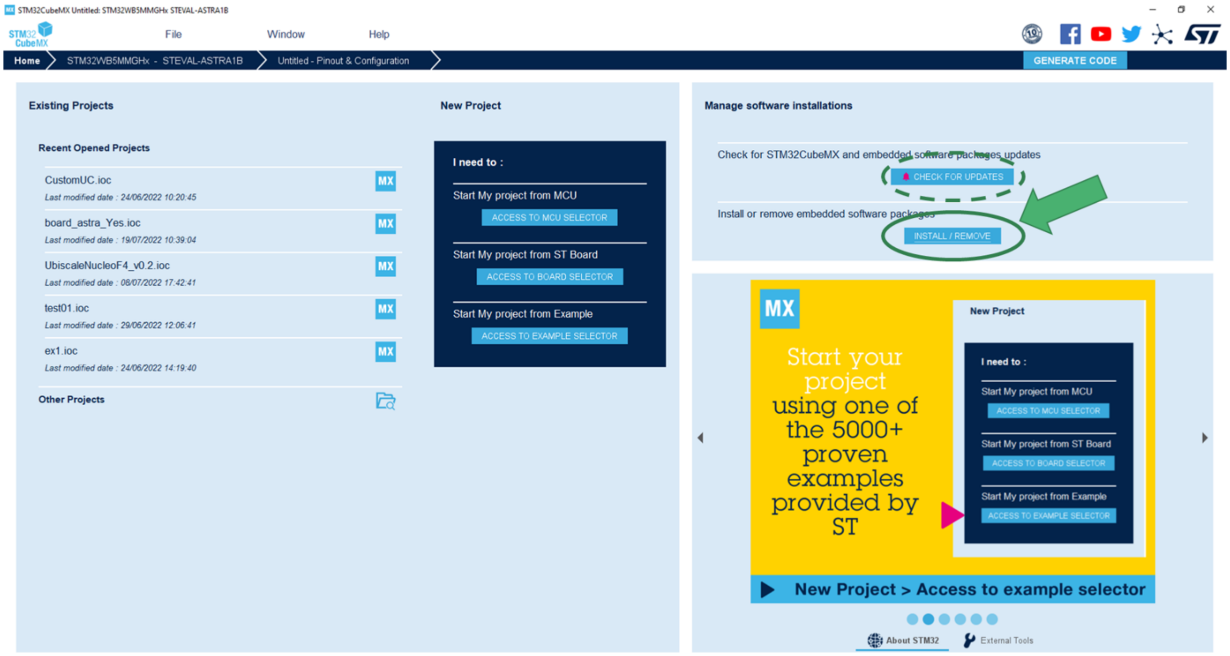


3.1 Prerequisites

The FP-ATR-ASTRA1 function pack is integrated in STM32CubeMX starting from version 6.6.0.

Ensure you have the 6.6.1 version (or above) installed. Then, install the pack by using the [**INSTALL/REMOVE**] button in the STM32CubeMX home view.

Figure 11. **[INSTALL/REMOVE] button**

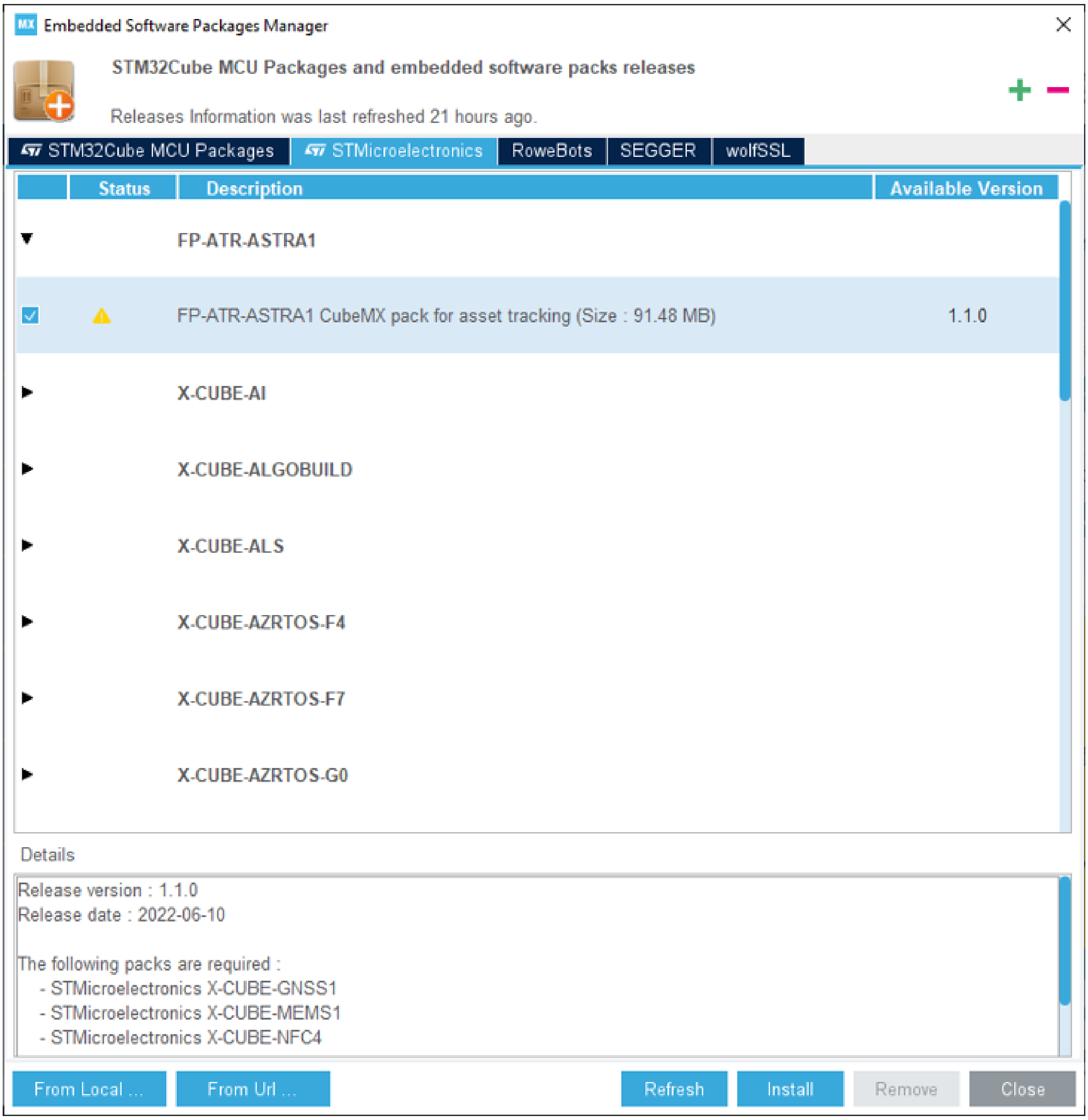


In the [**STMicroelectronics**] tab, select the FP-ATR-ASTRA1 pack. As shown in Figure 12, a yellow triangle signals that some other packs are required for the correct operation of the FP-ATR-ASTRA1. The [**Details**] window shows the list of required packs:

- X-CUBE-GNSS1
- X-CUBE-MEMS1
- X-CUBE-NFC4

These packs are required to manage GNSS, MEMS, and NFC features.

Figure 12. **FP-ATR-ASTRA1 software package installation**



Select the FP-ATR-ASTRA1, X-CUBE-GNSS1, X-CUBE-MEMS1, and X-CUBE-NFC4 packs. Then, click on the [**Install**] button.

3.2 Board selector

This technique allows you to generate the initialization code for GPIOs and used peripherals.

By accessing the STM32CubeMX [**Board selector**] (Figure 13) and choosing the STEVAL-ASTRA1B board (Figure 14), you can choose to use or not use the default mode (Figure 15):

- if you choose the default mode for peripheral initialization, all GPIOs and IPs are configured according to the STEVAL-ASTRA1B schematics and connections to the other devices;
- otherwise, no mode is assigned and only GPIOs are configured.

Figure 13. **Board selector**

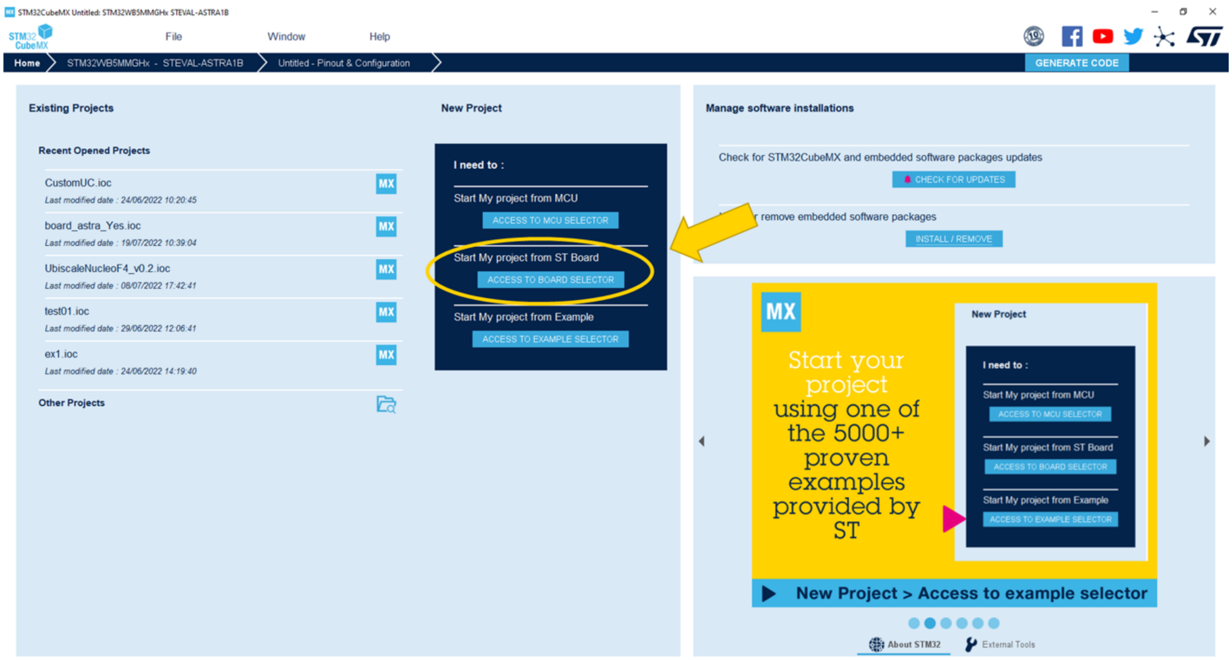
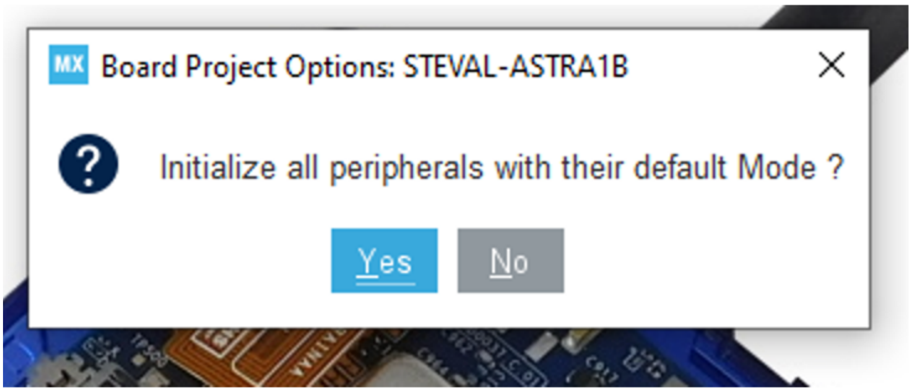


Figure 14. **STEVAL-ASTRA1B board selected**

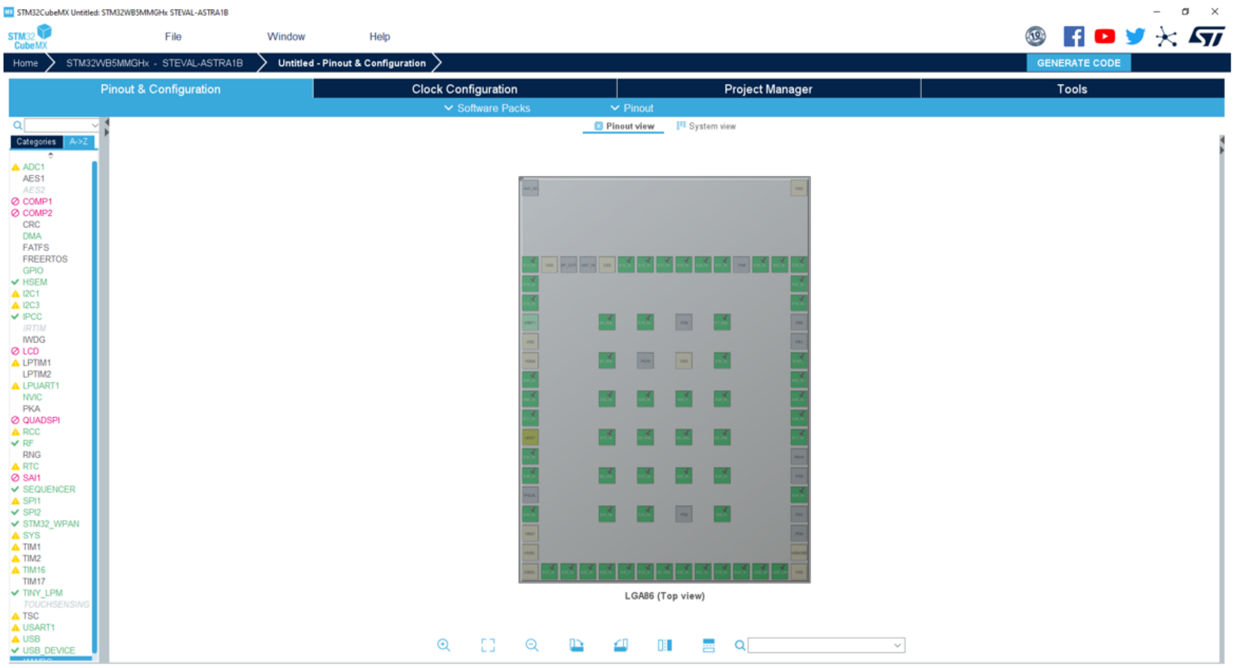
Figure 15. Board selector: peripheral initialization request



3.2.1 Board selector with peripherals in default mode

By answering “Yes” to the question of Figure 15, the following STM32CubeMX project is prepared:

Figure 16. Board selector with peripherals in default mode



The project is ready to be saved, after:

1. adding the [**Project Name**] in the [**Project Manager**] view
2. selecting the desired toolchain/IDE and click on [**GENERATE CODE**]

Starting from the generated code, you can write your own application, but having the initialization code ready (clock three, GPIOs, and peripherals configured according to the hardware design and with the same settings as the function pack), generated by STM32CubeMX.

Refer to UM2966, section 1.4.1.1 "Architecture and pinout", for the pin list and mapping.

3.2.2 Board selector without peripherals mode

By answering “No” to the question of Figure 15, the following STM32CubeMX project is prepared:

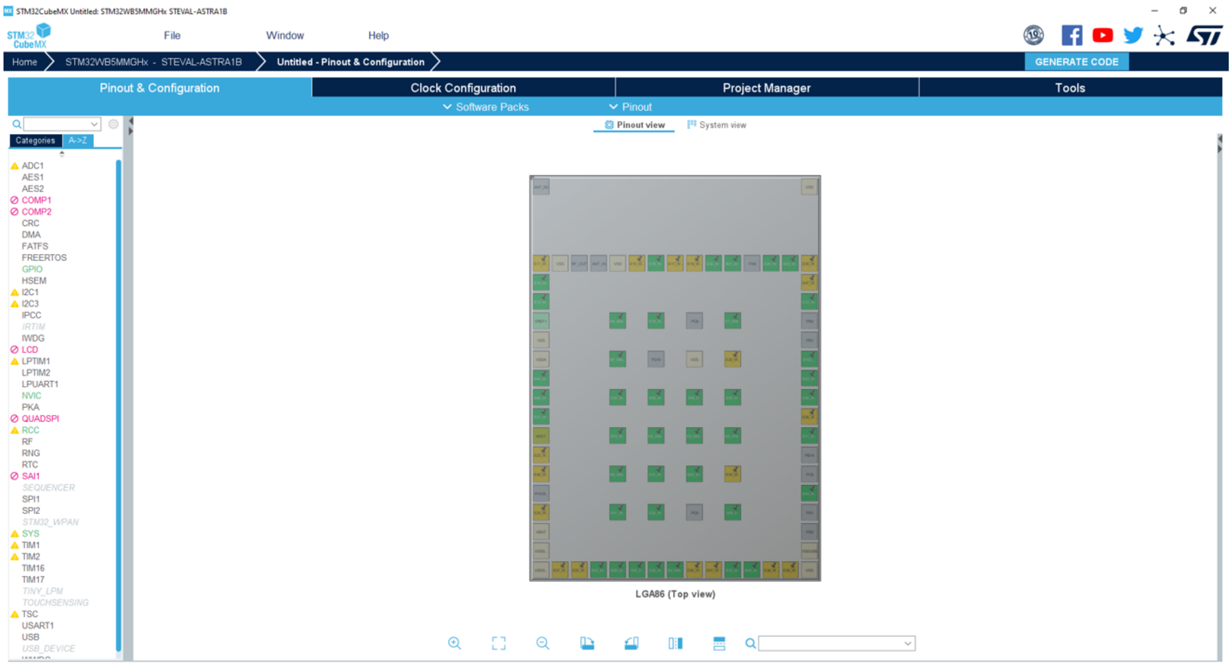


Figure 17. Board selector without peripherals mode

The project is not fully configured, as you should select/customize the peripheral mode according to the your application. After modifications are applied in STM32CubeMX, the project configuration must be completed by:

1. adding the [**Project Name**] in the [**Project Manager**] view
2. selecting the desired toolchain/IDE and click on [**GENERATE CODE**]

Starting from the generated code, you can write your own application, but having the initialization code ready (clock three, GPIOs, and peripherals configured according to the hardware design and with the same settings as the function pack), generated by STM32CubeMX.

Refer to UM2966, section 1.4.1.1 "Architecture and pinout", for the pin list and mapping.

3.3 FP-ATR-ASTRA1 pack dependencies

The FP-ATR-ASTRA1 pack implements the application layer firmware that uses other components already available in STM32CubeMX. These components are required for the correct compilation and operation. They are referenced in the pack definition file as dependencies. The STM32CubeMX requests the user to download the other needed packs (see Figure 12).

The figure below shows the middleware, the embedded utilities, and STM32CubeMX expansion packs required for the FP-ATR-ASTRA1 operation.

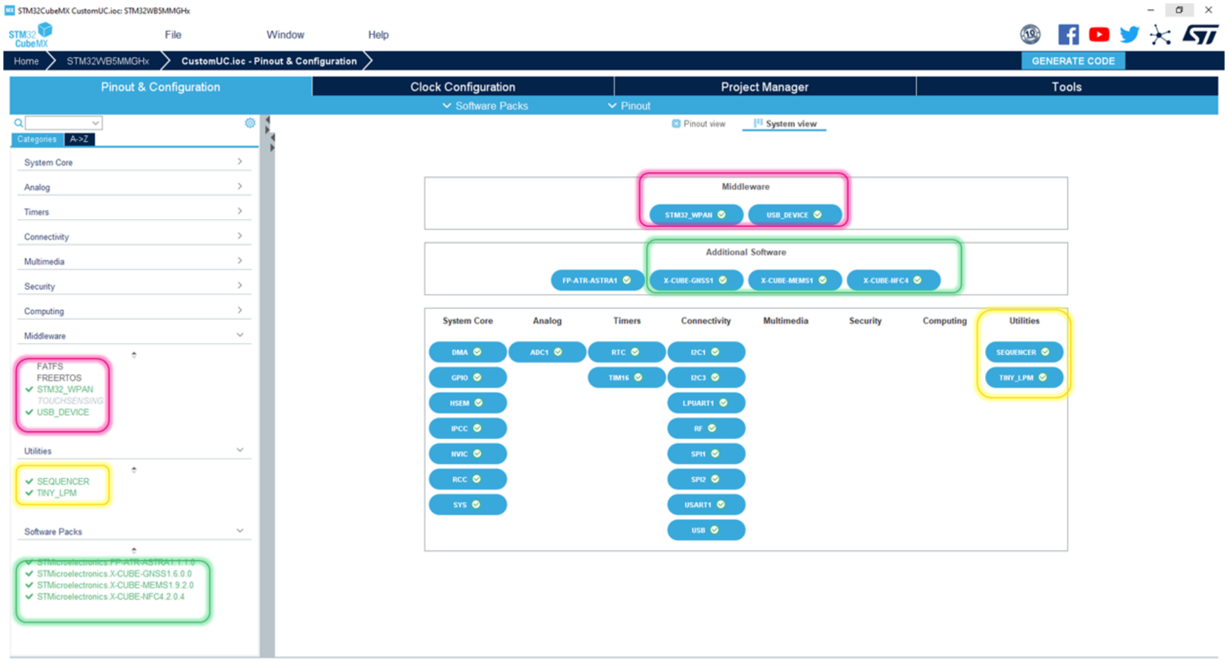


Figure 18. **Middleware, embedded utilities, and STM32CubeMX expansion packs**

The STM32CubeMX helps you to build your own project by automatically selecting the needed components through [**Component dependencies**].

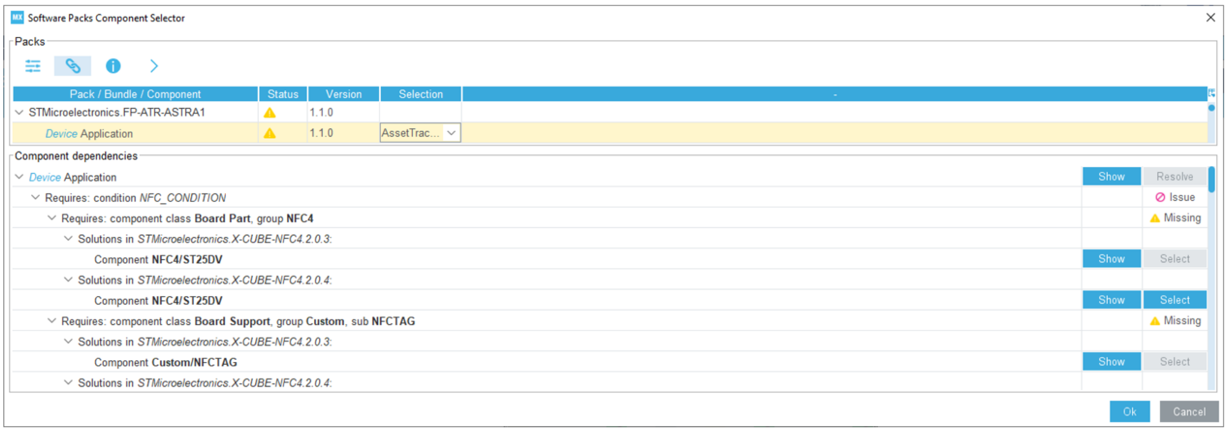
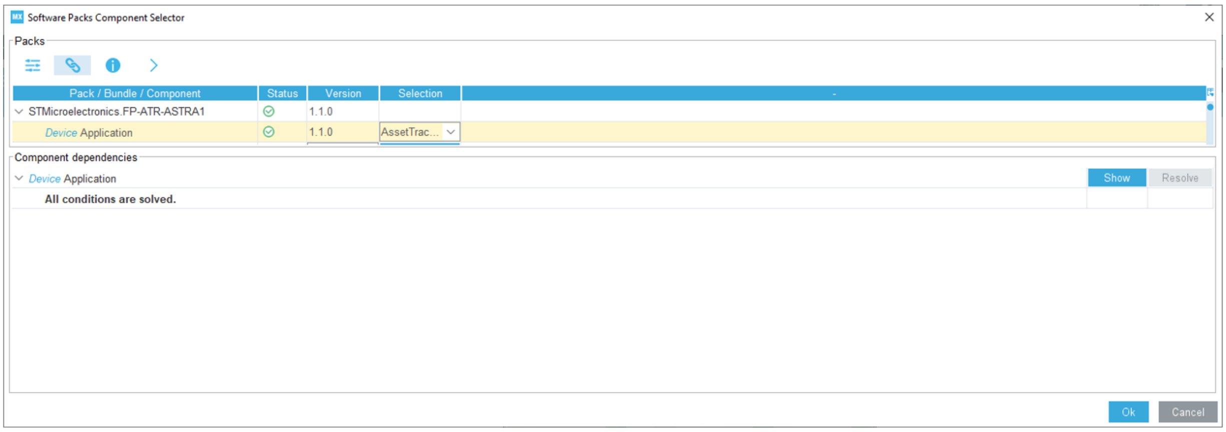


Figure 19. **[Component dependencies]: automatically adding the needed components**

Figure 20. [Component dependencies]: final view with all conditions solved



3.3.1 Middleware

The Astra pack.ioc example includes the following middleware:

- USB: communication device class (virtual COM port)
- STM32_WPAN: Bluetooth® middleware
- GNSS: middleware for the NMEA protocol support
- BLE_MANAGER
- PARSON: json utility
- STSAFE

Note: BLE_MANAGER, PARSON, and STSAFE are included in the application folder, as they do not belong to the STM32CubeMX components.

3.3.2 Embedded utilities

The FP-ATR-ASTRA1 requires the following embedded utilities:

- Sequencer, which allows creating a multitasking task project without a full operating system
- TinyLPM, which is a tiny low-power manager utility to manage the MCU power states.

3.3.3 STM32CubeMX expansion packs

The FP-ATR-ASTRA1 requires the following expansion packs:

- X-CUBE-GNSS1:
  Drivers for the Teseo-LIV3F global navigation satellite system (GNSS) device and middleware for the NMEA protocol support
- X-CUBE-MEMS1:
  Drivers that recognize the sensors and collect temperature, humidity, pressure, and motion data, including advanced motion libraries
- X-CUBE-NFC4:
  Drivers and middleware for STM32 to build applications using a dynamic NFC/RFID tag IC (ST25DV device)

3.4 FP-ATR-ASTRA1 customization

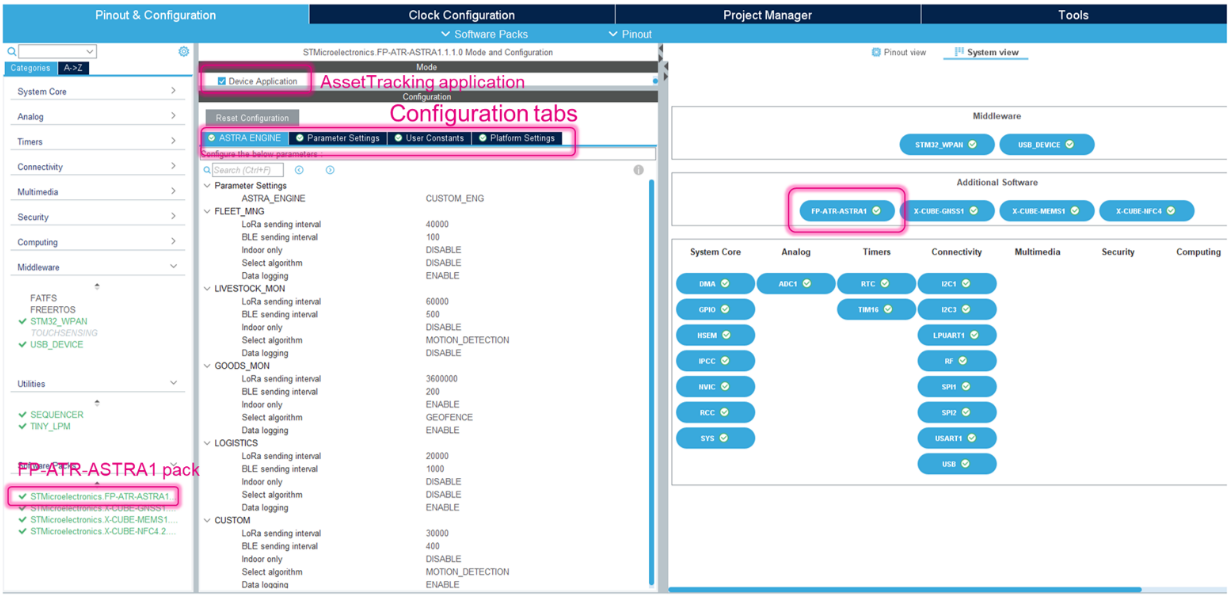


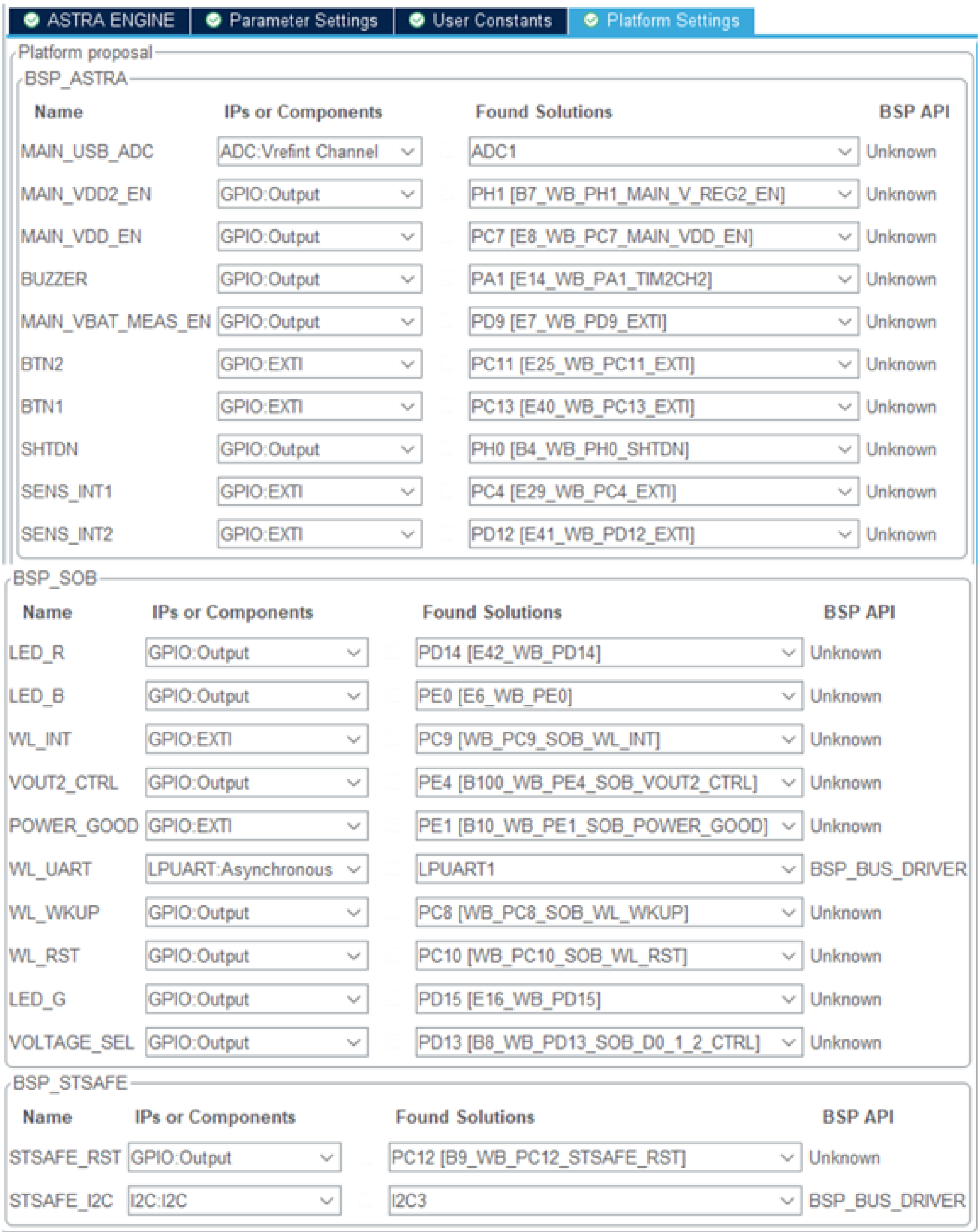
Figure 21. FP-ATR-ASTRA1 overview: asset tracking application component

The FP-ATR-ASTRA1 pack offers three tabs for the customization:
- [**Platform Settings**]
- [**Parameter Settings**]
- [**ASTRA ENGINE**]

3.4.1 [Platform Settings]

This tab allows you to set the configuration for the board specific pins.

Figure 22. **Asset tracking application: [Platform Settings] in the FP-ATR-ASTRA1**



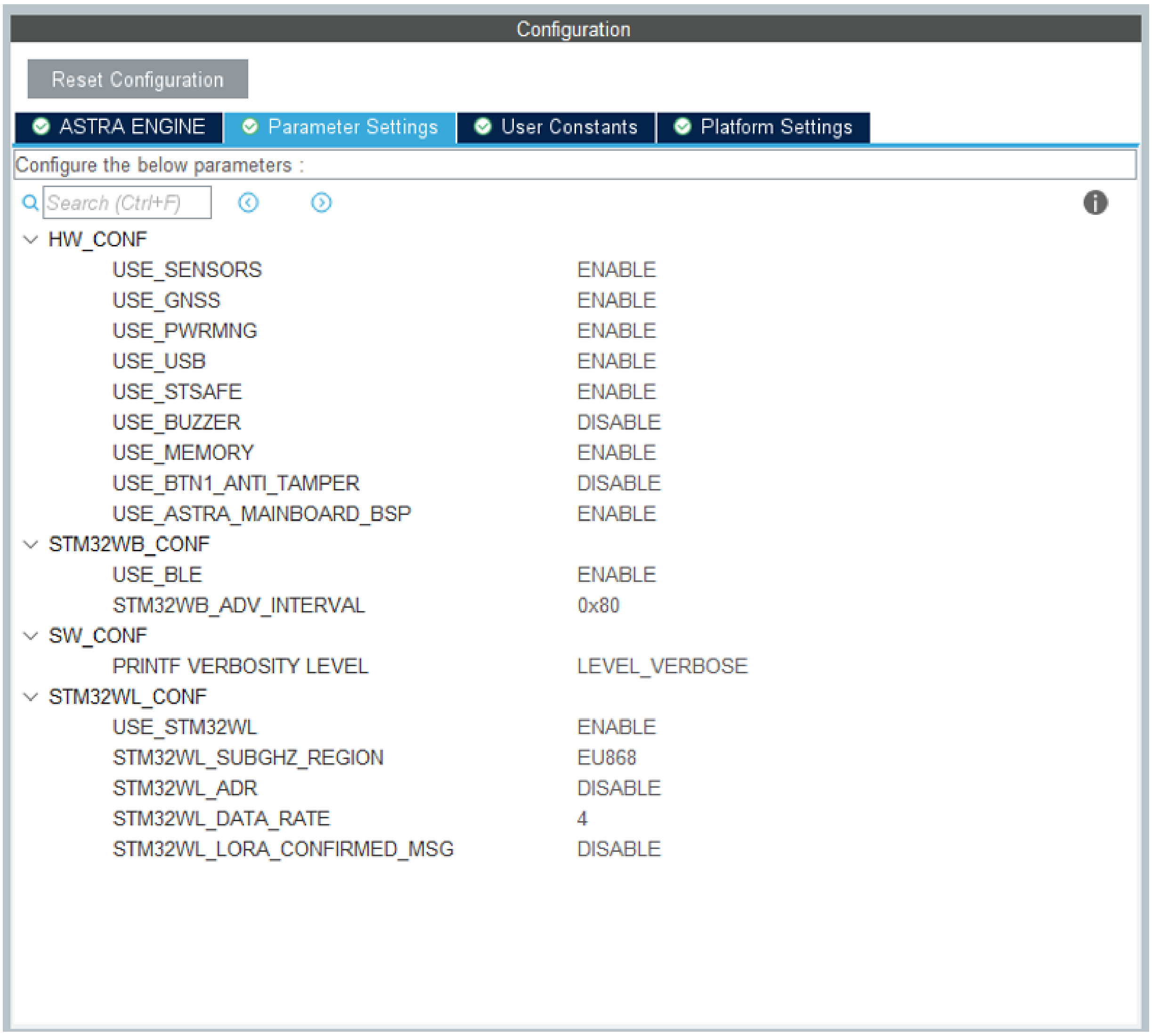
3.4.2 [Parameter Settings]

This tab allows you to set the following parameters:

- HW_CONF: to enable/disable hardware blocks

- SW_Conf:
  - to enable/disable the sequencer
  - to set the print verbosity
- STM32WB_CONF: for Bluetooth® Low Energy configuration
- STM32WL_CONF: for WL connectivity parameters

Figure 23. Asset tracking application: [Parameter Settings] in the FP-ATR-ASTRA1



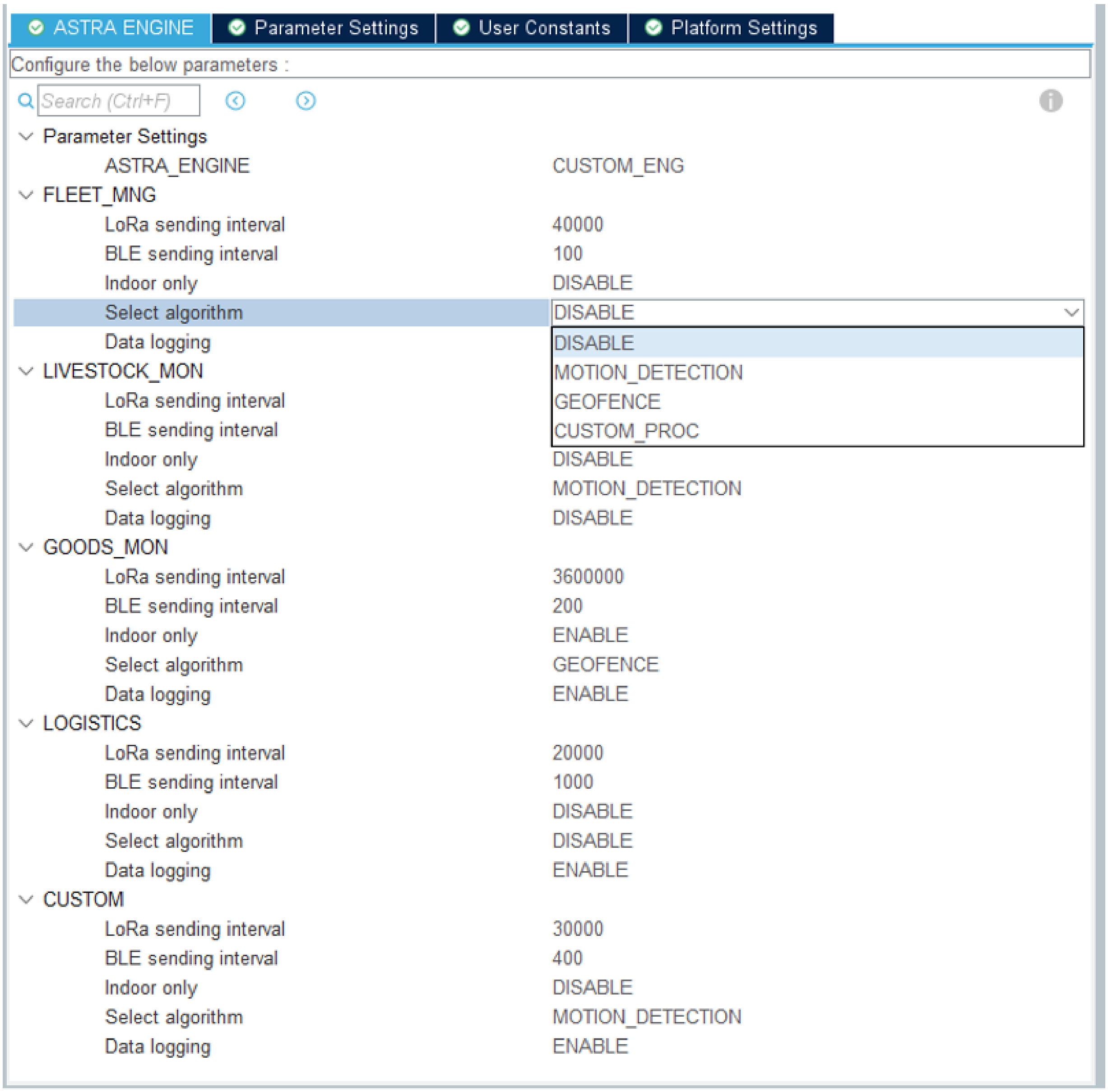
3.4.3 [ASTRA ENGINE] settings

This tab allows you to set:

- the current use case among:
  - Fleet management
  - Livestock monitoring
  - Goods monitoring
  - Logistics
  - Custom

- the profile:
  - Sending interval (msec)
  - Indoor only: enable/disable
  - Enable/select algorithm:
    - Algo1
    - Algo2
    - Etc.
  - Enable/disable datalogging

Figure 24. **Asset tracking application: [ASTRA ENGINE] settings in the FP-ATR-ASTRA1**

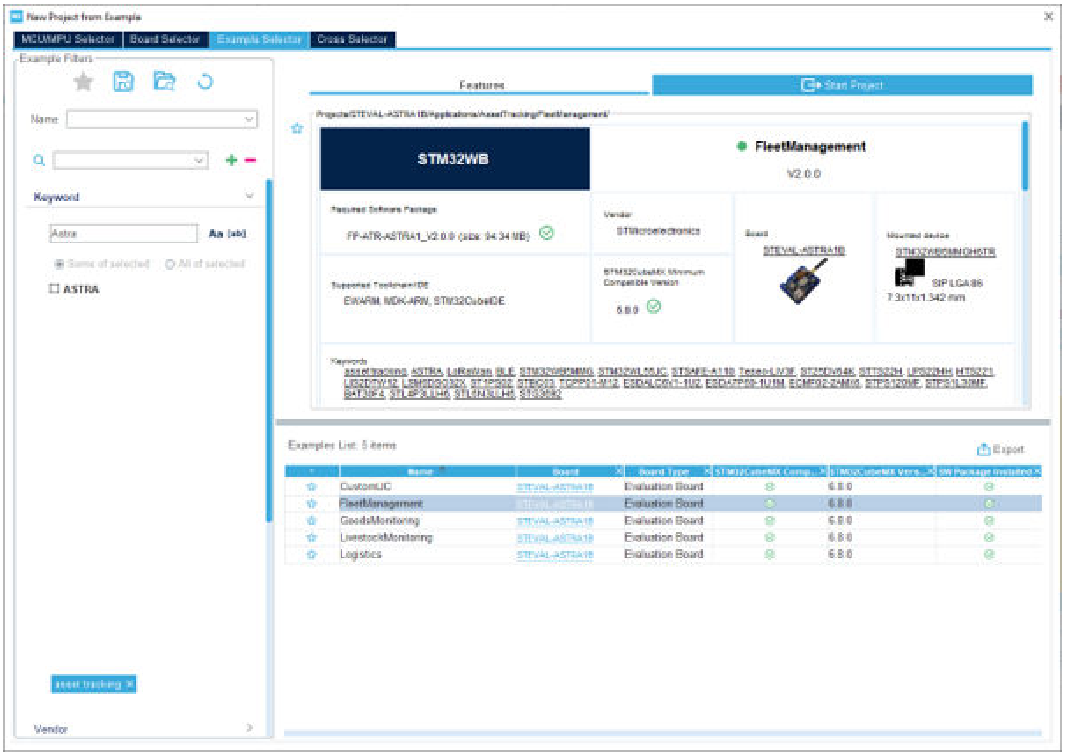


3.5 Create your own project with STM32CubeMX example selector

The example selector allows you to create a copy of the repository example in your working folder to modify it without touching the original one in the STM32CubeMX repository.

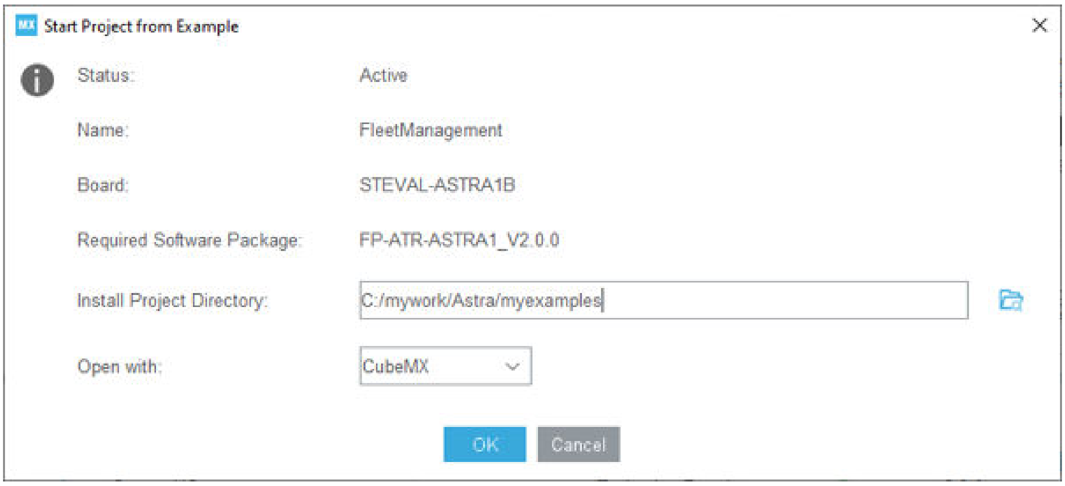
The FP-ATR-ASTRA1 related examples can be found with the “Astra” keyword:

Figure 25. **List of examples included in the FP-ATR-ASTRA1 pack: "Fleet management" selected**



After clicking the "Start project" button, the following screen allows you to select the destination folder

Figure 26. **Select the destination folder for your example**



Then the project folder will be prepared by the STM32CubeMX tool and the *.ioc* project is loaded. Here you can double check the parameters configuration and change them if needed. Once you are finished, select your preferred toolchain and click on "GENERATE CODE" button:

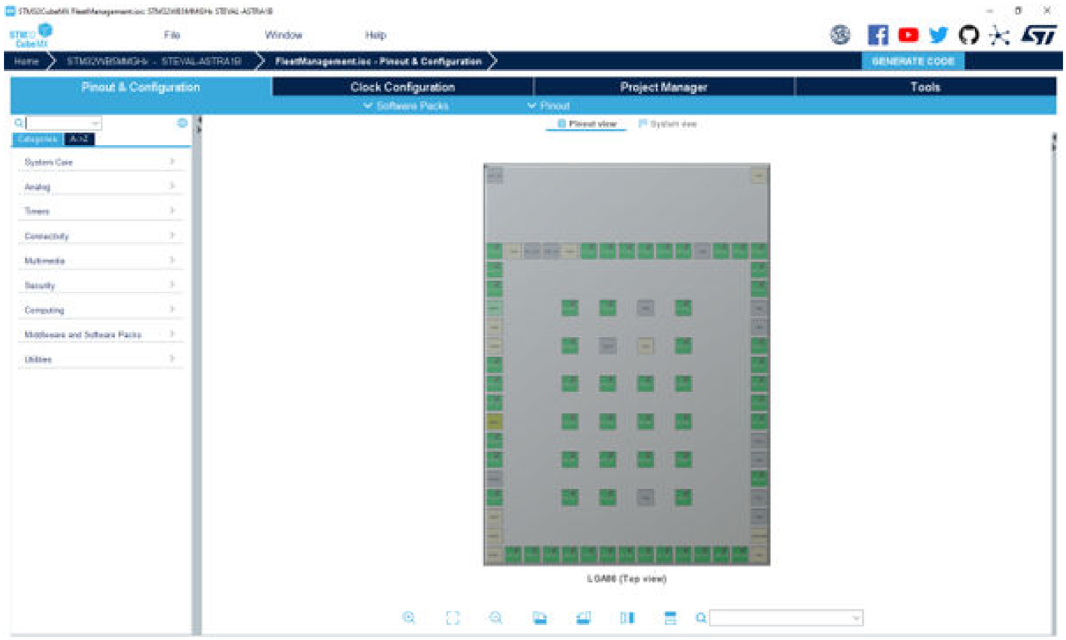
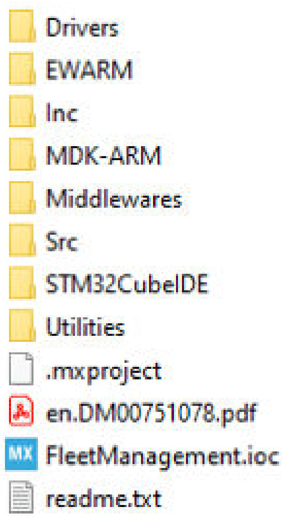


Figure 27. Fleet management example loaded in STM32CubeMX

Open your preferred toolchain (IAR Embedded Workbench for Arm, MDK-ARM or STM32CubeIDE) and load the project from the corresponding subfolder. In the folder you will also find the Inc and Src folders with the project header and source files, the Drivers, Middlewares and Utilities folders, the license file, readme file and STM32CubeMX files.

Figure 28. Project folder created by STM32CubeMX example selector

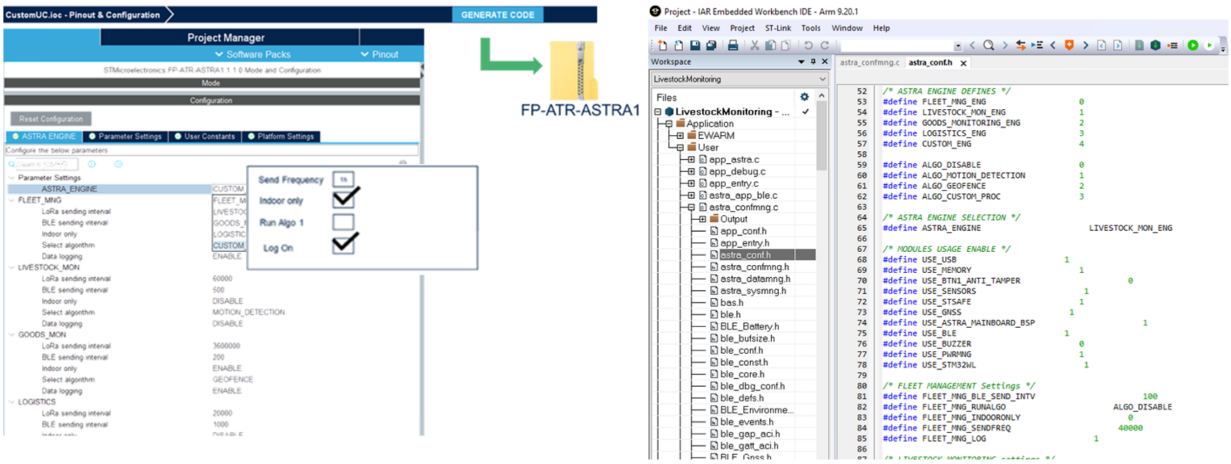


4 FP-ATR-ASTRA1 source code

4.1 Overview

Once all the settings are configured, the code can be generated. By opening the desired toolchain, the firmware is ready to be flashed on the board with the applied customizations.

Figure 29. Source code generation



The generated source code has a modular architecture in terms of hardware blocks and functionalities.

The hardware block management is identified through specific defines (`USE_GNSS`).

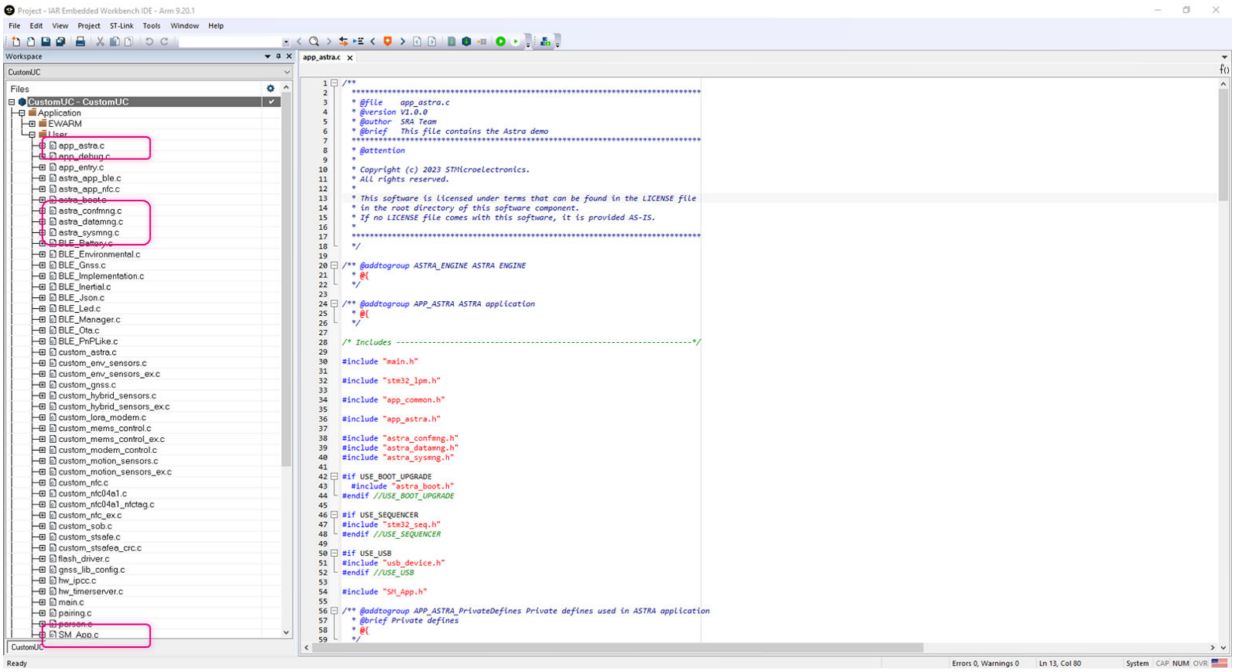
Functionalities are managed in different files, such as system initialization, state machine configuration, or data management.

4.2 Relevant files

Despite the file tree complexity, only few files are involved in the application configuration of the use cases:

- *app_astra.c/.h*
- *astra_confmng.c/.h*
- *astra_datamng.c/.h*
- *astra_sysmng.c/.h*
- *SM_APP.c/.h*

Figure 30. Application files



4.2.1 app_astra.c/.h

The *app_astra.c* file is the main file, that is, the entry point. It calls the initialization functions inside `MX_Astra_Init()`.

The `MX_Astra_Init()` function is used for the system initialization.

Figure 31. MX_Astra_Init() function

```
void MX_Astra_Init(void)
{
  custom_astra_init();

  HandleBoot();

  MX_IPCC_Init();

  PeriphClock_Config();

// __SM_CONF_H is used to switch to SM_Engine
#ifdef __SM_CONF_H
  SM_App_Init();
#endif //__SM_CONF_H
  custom_sob_power_init();
  custom_sob_v_reg1_out_high_value();
  custom_sob_led_init();
  AstraLedColor(ASTRA_LED_COLOR_GREEN);

  debug_init();

  PRINTF_INFO("SYSTEM initialization\r\n");
```

After some calls to handle the boot and clock configuration, the `SM_App_Init()` is in charge of the state machine initialization. The basic power configuration and debug initialization functions are also called here.

The `MX_Astra_Process()` function is used for the main loop.

Here, the `AstraProcess()` function is called either directly or by the sequencer. Inside the `void AstraProcess(void)` function, the `SM_App_Process()` manages the entire system according to the state machine described in SM_App.c.

The `Init` functions are implemented in the *astra_confmng.c/.h* file and the process functions are implemented in the *astra_datamng.c/.h* file.

4.2.2 astra_confmng.c/.h

This is the board configuration manager with the variables selected by the user to enable/disable each hardware block and use case implementation and configuration.

Figure 32. How to select and configure use cases in the code

```
/* ASTRA ENGINE DEFINES */
#define FLEET_MNG_ENG                   0
#define LIVESTOCK_MON_ENG               1
#define GOODS_MONITORING_ENG            2
#define LOGISTICS_ENG                   3
#define CUSTOM_ENG                      4

#define ALGO_DISABLE                    0
#define ALGO_MOTION_DETECTION           1
#define ALGO_GEOFENCE                   2
```

```
/* FLEET MANAGEMENT Settings */
#define FLEET_MNG_BLE_SEND_INTV                 100
#define FLEET_MNG_RUNALGO                 ALGO_DISABLE
#define FLEET_MNG_INDOORONLY                  0
#define FLEET_MNG_SENDFREQ                  40000
#define FLEET_MNG_LOG                 1

/* LIVESTOCK MONITORING settings */
#define LIVESTOCK_MON_INDOORONLY                 0
#define LIVESTOCK_MON_RUNALGO                 ALGO_MOTION_DETECTION
#define LIVESTOCK_MON_LOG                 0
#define LIVESTOCK_MON_SENDFREQ                 10000
#define LIVESTOCK_BLE_SEND_INTV                100

/* GOODS MONITORING Settings */
#define GOODS_MON_INDOORONLY                  1
#define GOODS_MON_LOG                 1
#define GOODS_MON_SENDFREQ                  3600000
#define GOODS_MON_RUNALGO                 ALGO_GEOFENCE
#define GOODS_MON_BLE_SEND_INTV                 200

/* LOGISTICS settings */
#define LOGISTICS_RUNALGO                 ALGO_DISABLE
#define LOGISTICS_INDOORONLY                  0
#define LOGISTICS_SENDFREQ                  20000
#define LOGISTICS_LOG                 1
#define LOGISTICS_BLE_SEND_INTV                 1000
```

Refer to *astra_conf.h*, included in the astra_confmng.h, for:

- *ASTRA ENGINE* parameters;
- module usage parameters (`USE_XXXX`);
- generic info: MCU name, board name, firmware version string, etc.

4.2.3 astra_datamng.c/.h

In this file, the data gathered from the sensors and other inputs are stored in the RAM. They are ready to be manipulated, for example, to run a specific algorithm on the data.

There is a globally accessible *C* structure, the `AstraEngData`, where the data are available anytime.

The data are acquired from each module:

- for example, `stm32wl_process()` for input/output on LoRa
- `ReadSensors(); ManageSensorsEvents();` for sensor input
- saved in the RAM and/or EEPROM memory
- `AstraEngData_t` to store the latest acquired data

The `AstraEngData_t` structure is globally accessible.

The following code is used for the `AstraEngData_t` event data.

```
typedef struct
{
  struct
  {
    uint8_t bDebuggerEnabled:1;
    uint8_t GNSS_NEW_DATA:1;
    uint8_t ENVIRONMENTAL_NEW_DATA:1;
    uint8_t POW_MAN_NEW_DATA:1;
    uint8_t MEMS_EVENT_NEW_DATA:1;
    uint8_t INERTIAL_NEW_DATA:1;
    uint8_t bReserved7:1;
    uint8_t bReserved8:1;
    uint8_t bReserved8_16;
    uint8_t bReserved116_24;
    uint8_t bReserved124_32;
  }c;
```

The following code is used for the `AstraEngData_t` sensor data.

```
  struct
  {
    /* gnss */
    float gnss_latitude;
    float gnss_longitude;
    float gnss_altitude;
    int32_t gnss_sats;
    uint8_t gnss_fix_validity;

    /* sensors */
    MOTION_SENSOR_Axes_t AccValue;
    MOTION_SENSOR_Axes_t LPAccValue;
    MOTION_SENSOR_Axes_t GyrValue;

    float LPTempValue;

    float PressValue;
    float TempValue;
    float HumValue;

    MOTION_SENSOR_Event_Status_t LPAccInt;
    MOTION_SENSOR_Event_Status_t AccInt;
```

The following code is used for the `AstraEngData_t` ADC data.

```
/* ADC */
    uint16_t batteryVoltage;
    uint16_t USBVoltage;

  }d;

  uint32_t nFwVersion;
} AstraEngData_t;
```

For the LoRa communication, the LoRa packet is formatted and sent inside the `LoraSendPacket()` function.

For the Bluetooth® Low Energy communication, the data are sent to the Bluetooth® Low Energy client in the `ManageBle()` function in *astra_sysmng.c*.

4.2.4 astra_sysmng.c/.h

In this file, the system-related functions are implemented.

The main functionalities are the command line interface, button callbacks, algorithms, LEDs, asset tracking use case management, and the timer management. Most of these are managed in the `AstraSysSmManager()` function:

- `manageDecimationTasks` for the timer management (for LoRa joining or data sending, Bluetooth® Low Energy advertising flag update, GNSS data read)
- `manageLeds` for LED blinking
- `manageAlgorithms()`, which is a placeholder for the data processing
- `manageData()` for the data-related flag management and the Bluetooth® Low Energy data sending
- `UpdateSystemStatus()` for the system status update in the Bluetooth® Low Energy advertising

4.2.5 SM_APP.c/.h

These files contain the configuration structures of the state machine.

See Section 2.2 for more details.

The main functions involved in the state machine management are `SM_App_Init` and `SM_App_Process`.

The `SM_App_Init` configures the state machine data for the correct startup and operation.

The `SM_App_Process` calls the `SM_Cycle()` function to let the state machine evolve and adds some commodities (trace messages) for the system monitoring.

5 STEVAL-ASTRA1B BlueST-SDK and Bluetooth® Low Energy manager

The STEVAL-ASTRA1B acts as a Bluetooth® Low Energy peripheral. You can connect it via a Bluetooth® Low Energy central entity, such as a smartphone, tablet, or industrial gateway.

The Bluetooth® Low Energy firmware implementation is simplified thanks to the BlueST-SDK. This SDK allows the board to be recognized also in advertising mode, as the advertising packet has a specific structure and some Bluetooth® Low Energy characteristics are preconfigured.

The main functionalities are:

- the data exchange;
- the debug console characteristic for the trace messages and textual human interaction;
- the extended configuration characteristic for the configuration data exchange in the json format.

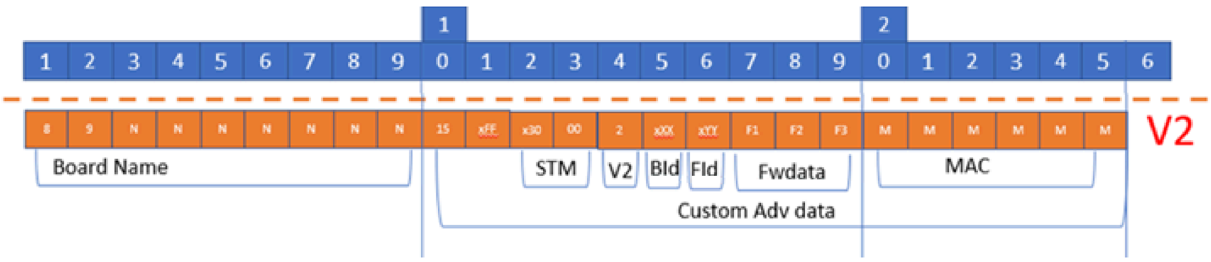
These functionalities are available in the middleware through the APIs.

The BlueST-SDK enables the microcontroller firmware to advertise its capabilities and exchange data with mobile applications as well as industrial gateways via Bluetooth® Low Energy. It is shared as an open-source code on multiple Github repositories for Android, iOS, and Python.

The following figure shows the advertise frame format.

Figure 33. Advertise format

Byte	Value	Description
0	0x09 or 0x0F	Length of STM specific advertisement data
1	**0xFF**	**Manufacturer specific advertisement data flag**
2	0x30	
3	0x00	0x0030 = STMicroelectronics
4	0x02	BlueSTSDK protocol version
5	0x00-0xFF	Device ID (identify the board)
6	0x01-0xFF	Firmware ID
7	0x00-0xFF	Custom option byte 1
8	0x00-0xFF	Custom option byte 2
9	0x00-0xFF	Custom option byte 3
10-15		6 byte device MAC (optional)



We assume that the mobile application (or the gateway), which receives the advertisement and needs to interpret it, is able to translate the device ID and the firmware ID into a set of related properties. This can be achieved by relying on a database.

The mapping among the device ID, the firmware ID, and the actual description of the firmware capabilities relies on a json document.

There are maximum three option bytes available. The firmware developer can define their meaning to convey different types of information. The developer also translates each byte and its meaning (for example, the mobile application according to the mapping table defined in the json file).

5.1 STEVAL-ASTRA1B advertising packet

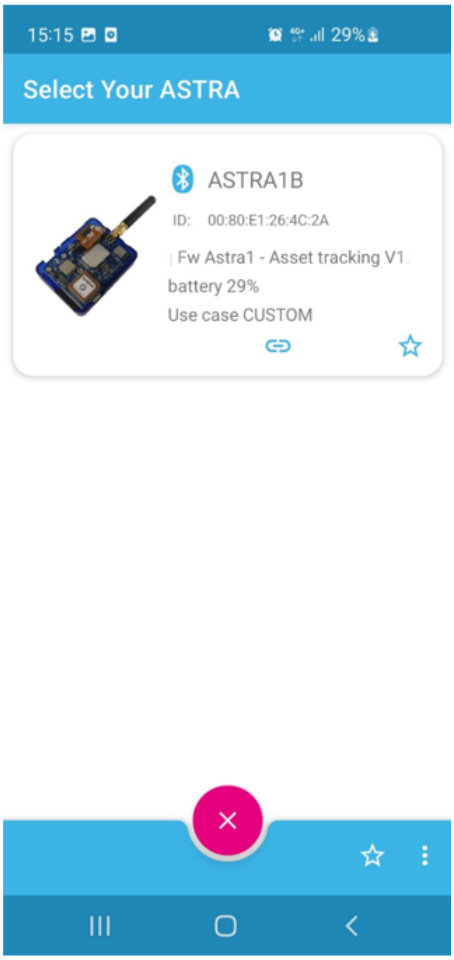
The STEVAL-ASTRA1B advertising packet includes the following options:

- *ble_dev_id = 0x0C*, where 0x0C = STEVAL-ASTRA1B
- *ble_fw_id = 0x01* for the public firmware (*Astra1 - Asset tracking*)
- *ble_fw_id = 0xFE* for the development firmware

There are three option bytes:

- battery level percentage
- *Enum String = Use case* (for example, *LOGISTICS*, *GOODS_MONITORING*, etc.)
- *Enum Icon = System Info* (for example, *Lora Joined*, *GNSS fix ok*, etc.)

Figure 34. STEVAL-ASTRA1B discovery in the STAssetTracking app



The STAssetTracking app adds other information, as soon as the board is recognized. The related information is gathered from a json file downloaded from the web server (for example, firmware version, firmware name, implemented characteristics, etc.).

5.2 Data exchange

The Bluetooth® Low Energy manager configures the device characteristics according to the data exchanged with the Bluetooth® Low Energy master.

The STEVAL-ASTRA1B configured characteristics are:

- data:
  - battery: battery data
  - environmental: environmental data
  - motion: motion data
- firmware upgrade:
  - FUOTA characteristic to manage the board restart for the firmware upgrade
- debug console:
  - output trace messages from the application
  - output errors from the application (error messages are highlighted among the traces)
  - receive user textual input for human interaction (refer to Section 9 for USB commands and debug)
- extended configuration characteristic:
  - configuration commands, exchanged in json format (for further details, see Section 8)

6 NFC dynamic memory

The STEVAL-ASTRA1B evaluation board is equipped with a ST25DV64KC device which is a NFC/RFID dynamic tag fitted with a 64-kbit electrically erasable programmable memory (EEPROM). It features two interfaces: an I²C serial link that can be operated from a DC power line as well as a RF link which is activated when the device acts as a contactless memory powered by the received carrier electromagnetic wave.

The ST25DV64KC device offers a fast transfer mode between the wireless and the wired worlds, thanks to a 256 bytes volatile buffer (also called mailbox). In addition, it provides a GPO interruption pin which can be configured on multiple RF events (field change, memory write, activity, fast transfer end, user set/reset/pulse) and I^2C events (memory write completed, RF switch off).

The ST25DV64KC device is compliant with Type 5 Tag (NFC-V) defined by the NFC forum and it can handle the NFC data exchange format (NDEF), which NFC standard adds upon the generic RFID.

NDEF is a light-weight binary message format designed to encapsulate application-defined payload bearing one or more NDEF records into a single message. The NDEF records concatenation defines the NDEF message.

NDEF message defined by the FP-ATR-ASTRA1 is composed of:

- a 8-bytes Capability Container (CC) file which manages the information compliant with the NFC Forum Type 5 Tag
- the TLV fields

The TLV format is a generic data structure used to embed information and to store NDEF messages which is composed of three fields:

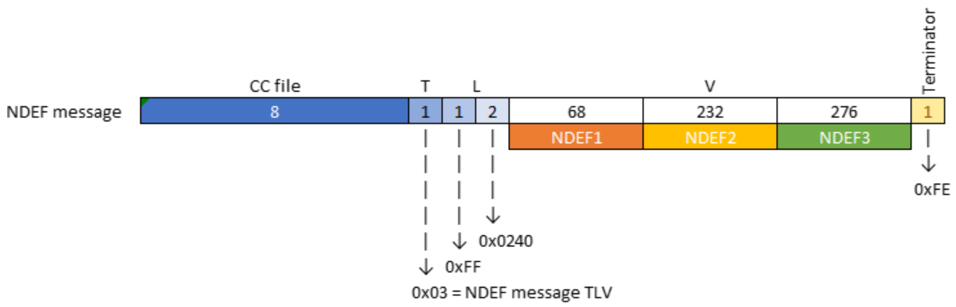
- Type field (T)
- Length field (L)
- Value field (V) which includes the NDEF messages.

Finally, the terminator byte 0xFE is the last TLV block in the data area.

As defined by the NFC forum, the T field value 0x03 specifies that the value field V contains NDEF messages.

The L field is coded by 3 bytes. It means that the first byte value 0xFF is followed by two bytes which declare the V field length.

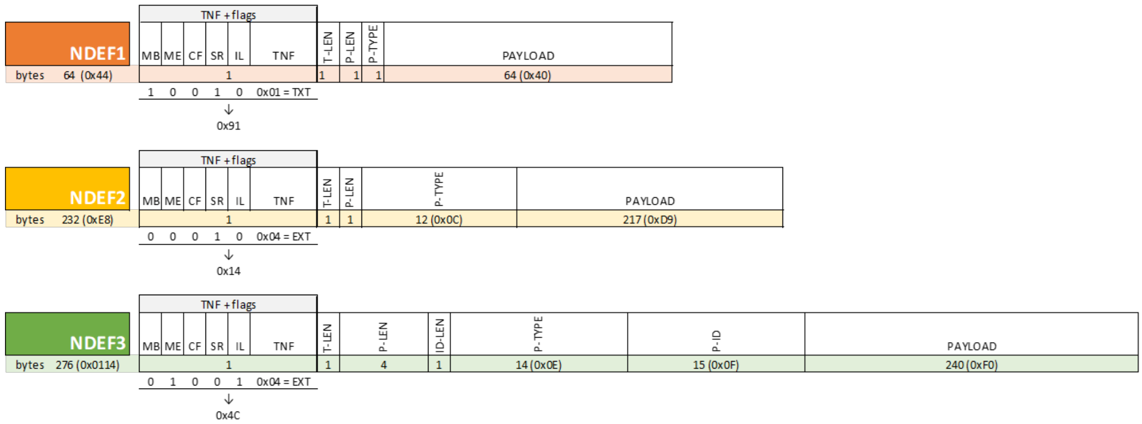
Figure 35. FP-ATR-ASTRA1 NDEF message



The FP-ATR-ASTRA1 organizes the NFC memory by means of three NDEF records, as shown in the picture below:

- NDEF record 1: NFC forum well known type (URI record). It is used to support provisioning via NFC
- NDEF record 2: EXTERNAL record. It is a reserved area to store: LoRa keys, configuration parameters and FW information.
- NDEF record 3: EXTERNAL record. It stores the datalog configuration parameters and sensors data according to the SmarTag2 protocol

Figure 36. FP-ATR-ASTRA1 NDEF records



Each NDEF record is made up of a header and a payload. The header describes the payload.

The first five bits in the NDEF record header are flags. They are used to describe how to process the record and some information about the record's location in the message.

The bit flags in the first byte of the record header are as follows:

- MB (Message Begin). If it is set, the NDEF record is the first in the NDEF message.
- ME (Message End). If it is set, the NDEF record is the last in the NDEF message.
- CF (Chunk Flag). If it is set, the NDEF record is chunked.
- SR (Short Record). If it is set, the short record format is used for payload length.
- IL (ID Length is present). If it is set, the "ID Length" and "ID payload" fields are present.

The Message Begin and Message End flags are used for processing records within a message. The first record in a message will have its MB flag set to true. The middle records will have both flags set to false. The end record in a message will have the ME flag set to true. A message with one record will have both the Message Begin and Message End bits set true.

The last three bits of the NDEF record header identify the Type Name Format (TNF) which provides information on how to interpret the "type" field. Among all the available TNF values listed in the table below, the FP-ATR-ASTRA1 implements the well-known type in the first NDEF message and the external type in the others.

Table 2. Type Name Format (TNF)

Type name format	Value
Empy	0x00
NFC forum well known type	0x01
Media type	0x02
Absolute URI	0x03
NFC forum external type	0x04
Unknown	0x05
Unchanged	0x06
Reserved	0x07

The header also identifies: the payload type length, the payload length, payload type and, optionally, payload identifier.

The “payload type” describes more specifically the content of the payload declared by the TNF bits. Since the description of the payload type can require more than one byte, the number of used bytes is defined in the “payload type length field” which is followed by an adjustable number of bytes containing the “payload length”.

The “payload identifier” is an optional field. It’s used to let the application identifying the payload within the record by ID or to allow other payloads to reference it. If the “payload identifier” is used it should be a valid URI. In this case a “ID length” field follows the “payload length”. It contains information on the number of bytes making up the “identifier payload” placed between the “payload type” and the “payload” fields.

The header is followed by the payload, which is the content of the NDEF record.

6.1 The datalog

The third NDEF record manages the datalog. The memory allocated for this purpose is compliant with the SmarTag2 protocol and it is structured as follows:

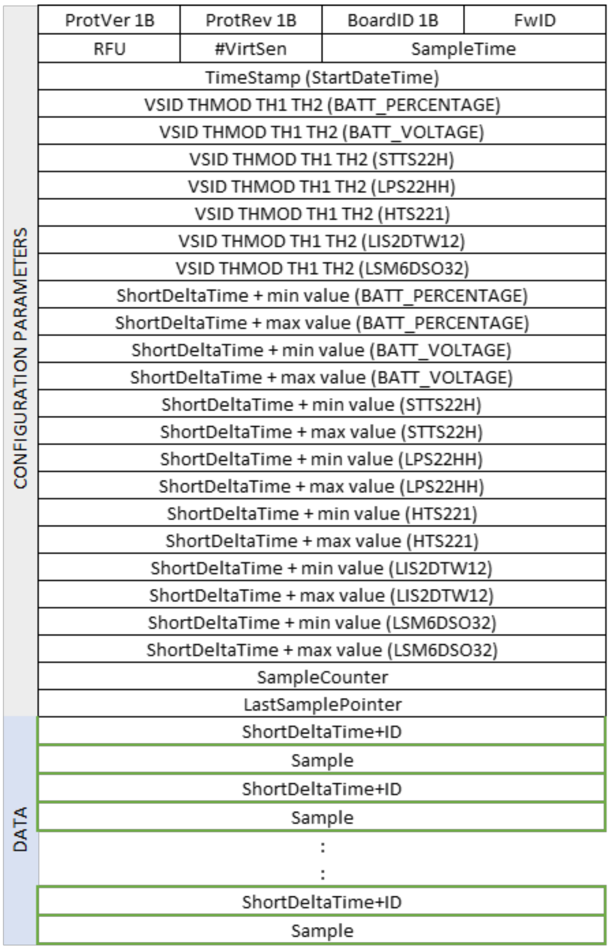
- The first 12 bytes contain information relating to: protocol version, protocol revision, Board ID, Firmware ID, RFU, number of virtual sensors, sample time and the TimeStamp.
- A memory area which contains information about the thresholds. The amount of memory allocated to this section depends on the number of sensors
- A memory area which contains information about the boundary values for each sensor according to the thresholds set. Also the memory allocated to this section depends on the number of sensors
- 8 bytes to store the sample counter
- 8 bytes to store the last sample pointer
- the remaining memory is filled with the measurements of each sensor. Each measure needs 16 bytes:
  - The first 8 bytes are used to store the ID associated with the sensor and the delta time referred to the Time Stamp described above
  - The remaining 8 bytes contain the measurement

Sensors are labelled as “virtual” because they could also contain information coming from real sensors or from indirect or composite measures such as the percentage of battery charge.

The number of virtual sensors actually monitored, as well as the Sample Time (which is the NFC memory data saving rate) can be changed acting on the “settings” view as described in paragraph 6.1.2.

The default Time Stamp which is set when datalog configuration is refreshed, is updated when the sensor configuration is modified by the ST Asset Tracking app.

Figure 37. **Datalog structure**



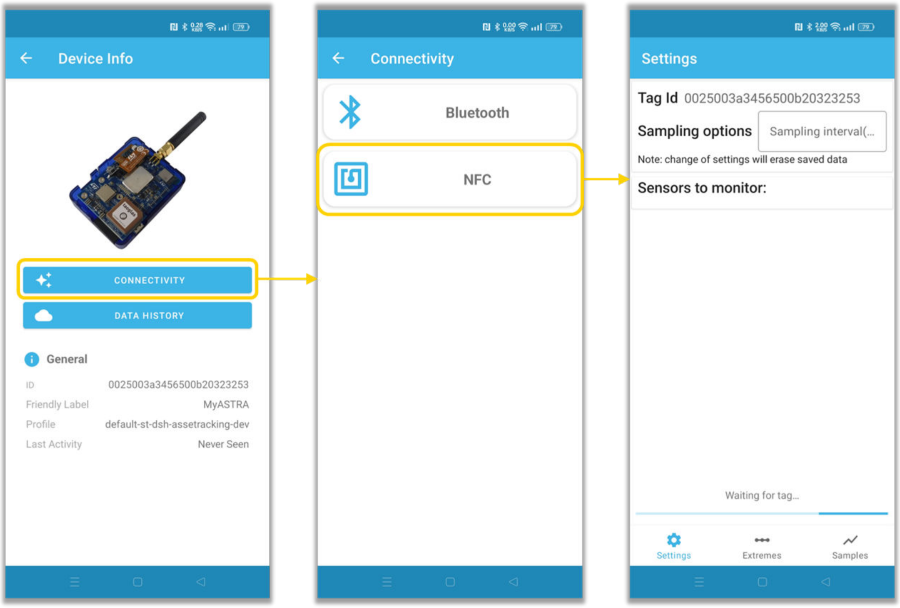
The datalog configuration and memory allocation depends on the number of virtual sensor and it is refreshed as well as it is updated by acting on the ST Asset Tracking app.

The ST Asset Tracking app supports NFC data upload to ST Asset Tracking dashboard.

6.1.1 How to access the NFC views

To access the NFC view, navigate through “connectivity” and then “NFC”. Thus, the smartphone waits to read the NFC memory. This operation is accomplished when the smartphone and the STEVAL-ASTRA1B board NFC antennas are aligned and matched.

Figure 38. How to move towards the NFC view



6.1.2 The “settings” view

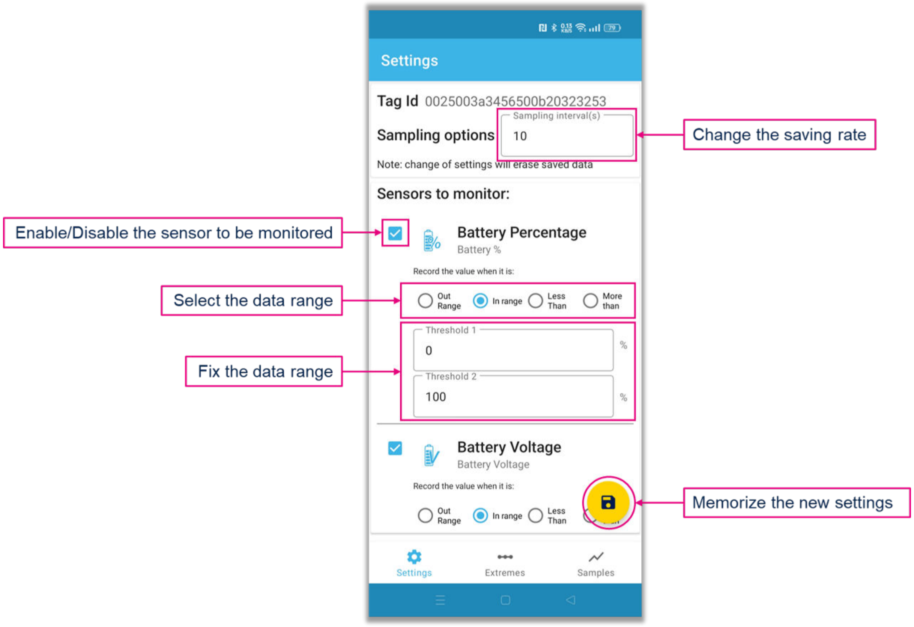
The “settings” view (see picture below) allows the user to select the sensor data to be logged. The FP-ATR-ASTRA1 supports the following “virtual sensors”: battery percentage, battery voltage, temperature, pressure, humidity, IMU accelerometer, 6-axis accelerometer. The data saving related to each sensor can be enabled or disabled by acting on the related checkbox and the maximum and minimum thresholds can be defined as well.

At the top of the “settings” view, the sampling field allows defining the NFC memory data saving rate.

Finally, to make the settings changes running, it is necessary to press the button marked with the save icon.

At the same time as updating the settings, the app changes the time reference stored into the memory, therefore the new data will be stored according to this.

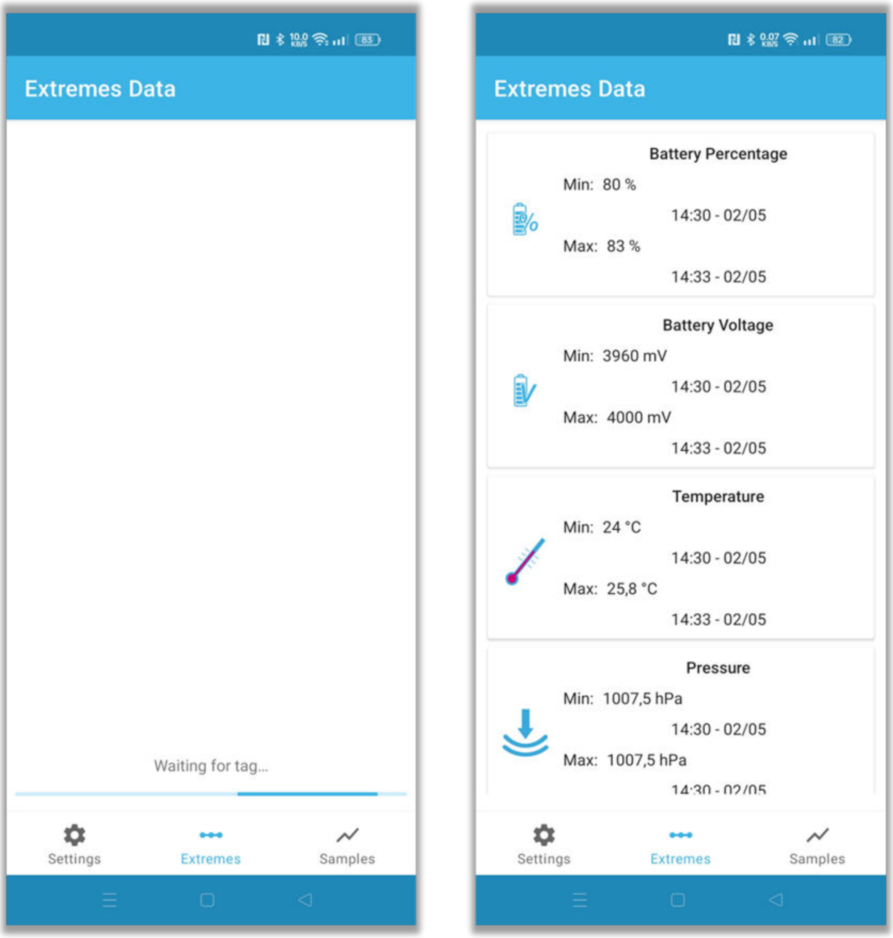
Figure 39. NFC settings view



6.1.3 The “extremes” view

The “extreme” view is enabled by clicking on the bottom center button of the NFC view. This page shows the maximum and minimum values obtained during the logging of the selected sensors.

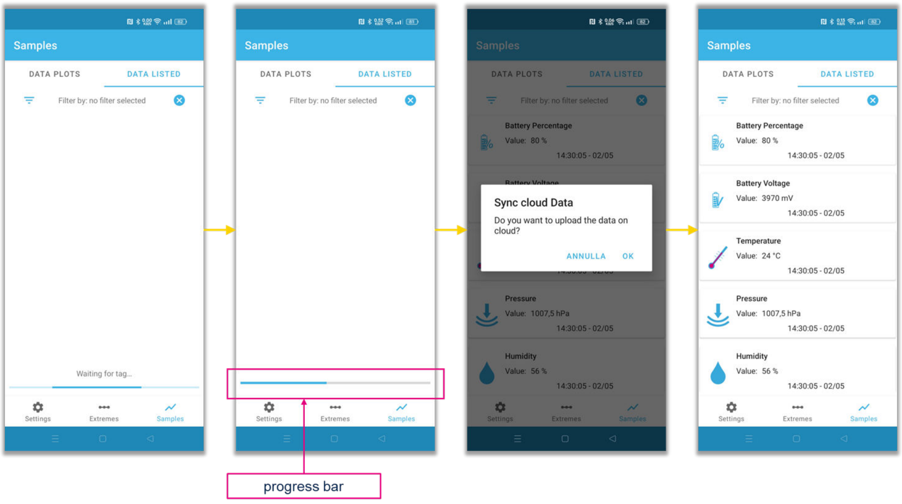
Figure 40. Data extremes



6.1.4 The “samples” view

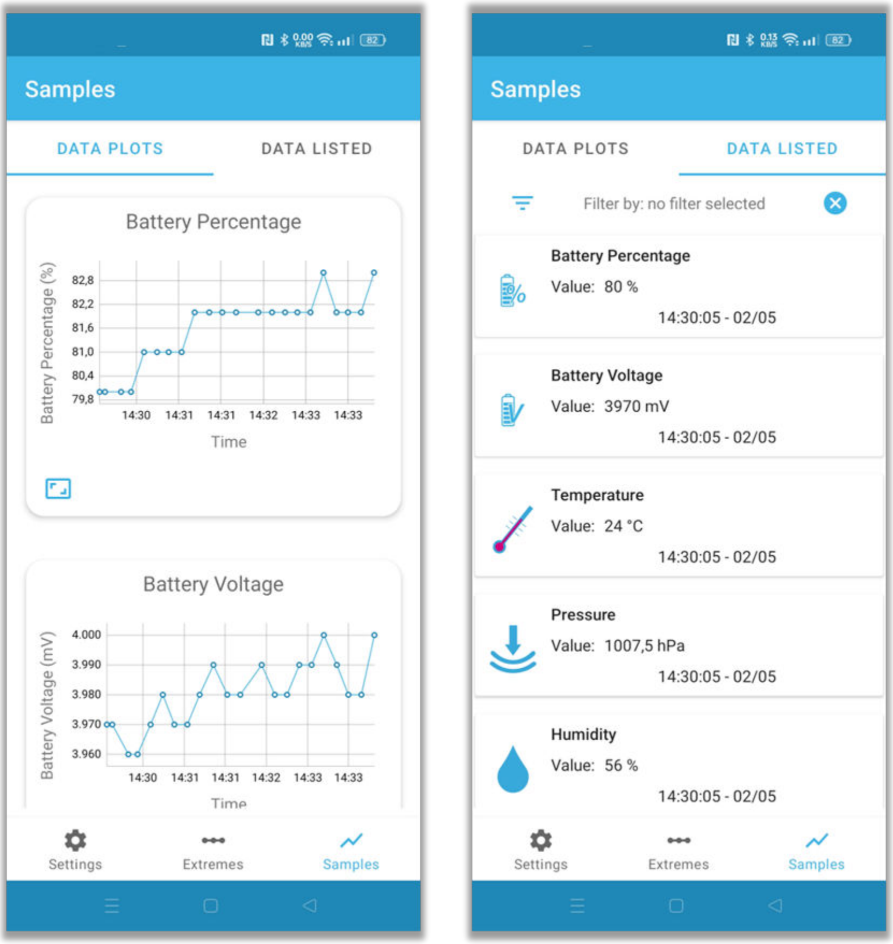
The “sample” view is enabled by clicking on the bottom right button of the NFC view. This page collects all the data logged for the selected data with the related data range. Due to the large amount of data to be transferred, the synchronization could take a few seconds. This operation is shown by a progress bar at the bottom of the screen. At the end of the synchronization phase between the smartphone and the STEVAL-ASTRA1B the ST Asset Tracking app allows uploading data on the cloud.

Figure 41. Data synchronization



Samples can be displayed as a list or plots, selecting the choice at the top of the view.

Figure 42. Samples views



7 LoRaWAN® stack

The LoRaWAN® stack runs on the STM32WL55JC microcontroller. The STM32WB5MMG drives the stack thanks to the AT command master. The STM32WL55JC, instead, runs the *AT-SLAVE* firmware preloaded in the default configuration of the STEVAL-ASTRA1B.

The STAssetTracking app can set the LoRaWAN® keys through specific commands. The keys are stored in the EEPROM (without KMS).

A simplified provisioning procedure through the STAssetTracking app automatically sets the needed keys to register them on TTN V3 and on the DSH-ASSETRACKING dashboard.

The LoRaWAN® frame payload is in Cayenne LPP format and consists of:

- pressure, temperature, and humidity
- accelerometer
- GNSS location
- analog input for the battery voltage and digital input for the tamper status

LoRa data packet is prepared in the `LoraSendPacket()` function in the *astra_datamng.c* file.

8 Using the STEVAL-ASTRA1B with the STAssetTracking app and the DSH-ASSETRACKING dashboard

The STAssetTracking app supports all the features and Bluetooth® Low Energy custom commands implemented in the FP-ATR-ASTRA1.

The app is available at Google Play and App store.

To allow the STEVAL-ASTRA1B board to join the LoRaWAN network, you need to use the STAssetTracking together with the DSH-ASSETRACKING dashboard.

8.1 How to connect the STEVAL-ASTRA1B to a LoRaWAN® network and to ST-ASSET tracking dashboard

Before using the STEVAL-ASTRA1B and FP-ATR-ASTRA1, you have to register the device on a network server and an application server. STMicroelectronics provides an asset tracking dashboard that allows the user to discover asset tracking STMicroelectronics solution.

You need a my.st.com account to install the STAssetTracking app. After the installation, follow the procedure below to log on to the app and the DSH-ASSETRACKING dashboard.

By using the STAssetTracking app you, will be automatically registered on the dashboard with TTN network server connection.

Important: *The system works with only high bands frequencies, thus the device and network server region has to be set accordingly (see* *Figure 43. ASTRA registration methods**).*

8.1.1 Device registration using the DSH-ASSETRACKING

Log in into the DSH-ASSETRACKING using your myst.com account, to register the device the user has to follow the following the instructions in this page https://dsh-assetracking.st.com/#/howto.

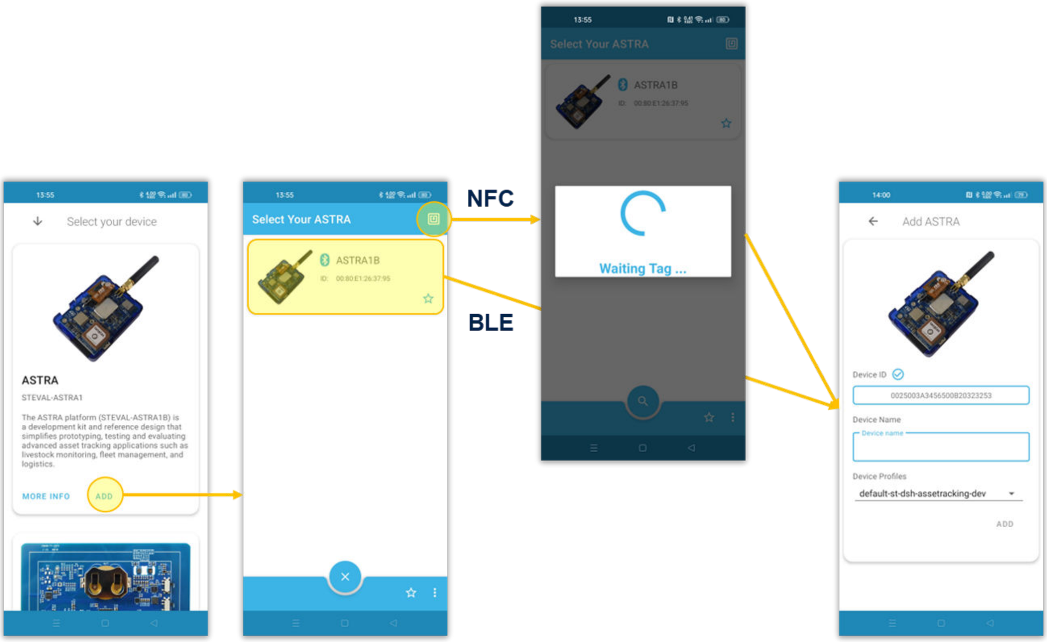
8.1.2 Device registration using mobile app

The user has two ways to start devices registration.

- NFC pairing
- BLE discovery

The user can provision the board by BLE manual connection tapping on ASTRA board icon or via NFC approaching the smartphone near the ASTRA to be registered and waiting for NFC tag reading (this second choice is very useful in case of multiple board in advertising mode).

Figure 43. **ASTRA registration methods**



One the ASTRA board is discovered the registration page appears.

Figure 44. **Registration parameters**

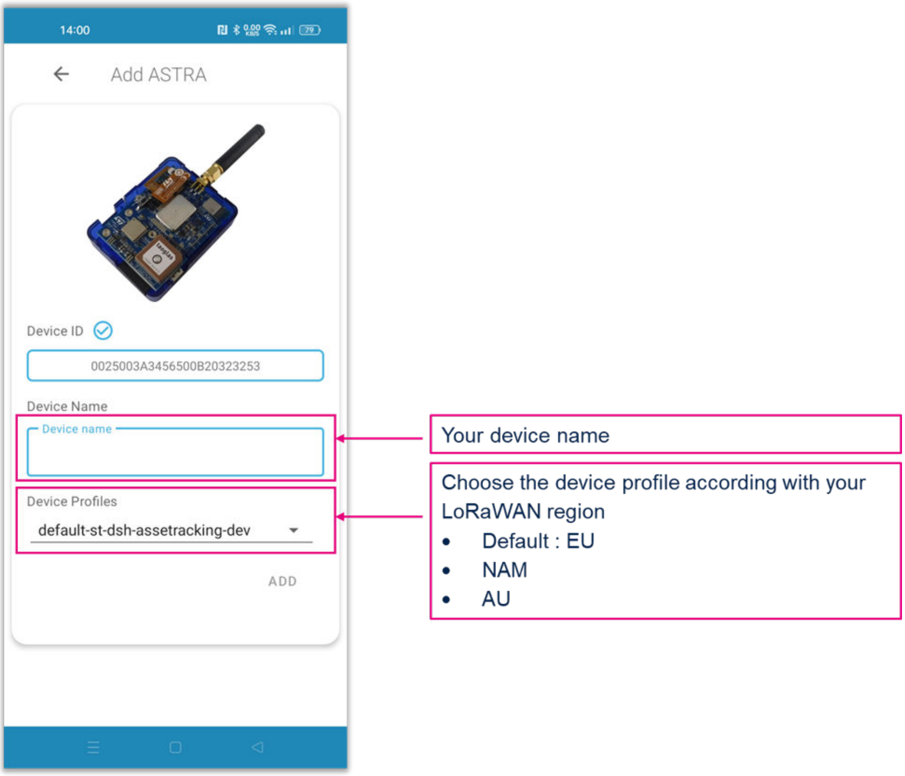
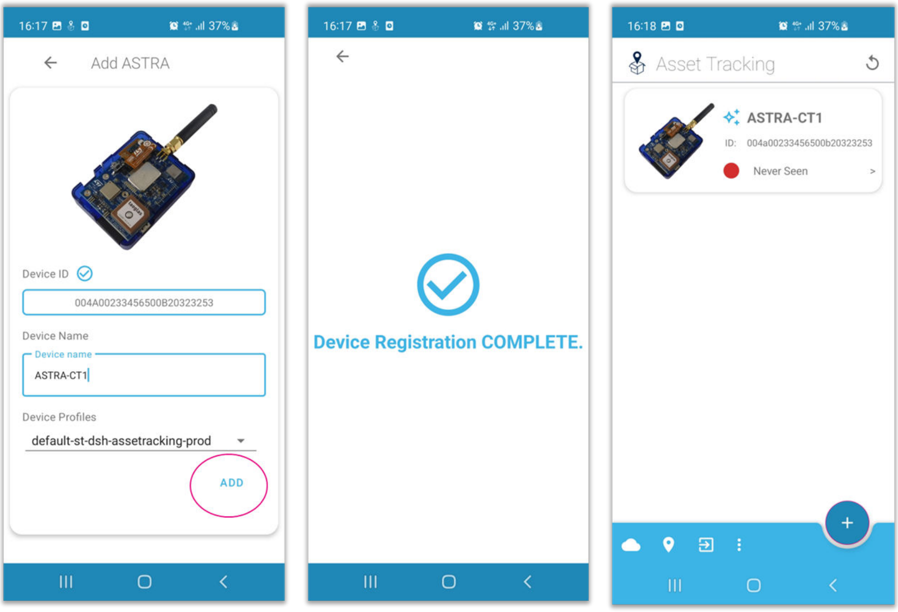


Figure 45. Registration procedure



As outcome of the registration procedure, the STEVAL-ASTRA1B board is provisioned on the LoraWAN network according to the JoinEUI, APP and Network keys.

This operation already provision the ASTRA board with JoinEUI, APP and Network keys. The STEVAL-ASTRA1B board can join the LoRaWAN network as soon as it is available. After joining, it can start sending data to the dashboard.

8.1.3 Gateway settings

To use STAssetTracking automatic registration you need to register (https://www.thethingsindustries.com/docs/gateways/concepts/adding-gateways/) and use the gateway with the frequency plan channel and in the cloud listed in the table below:

Table 3. Cloud

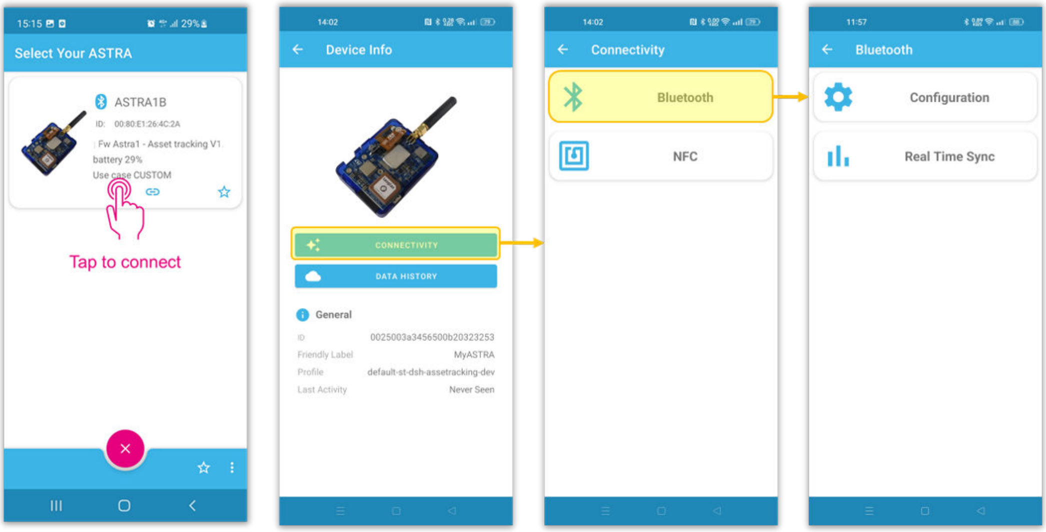
Region	Cloud	Frequency plan
AME	nam1.cloud.thethings.network	US_902_928_FSB_1
EU	eu1.cloud.thethings.network	EU_863_870_TTN
AU	au1.cloud.thethings.network	AU_915_928_FSB_1

8.2 How to use STASSETTRACKING APP with STEVAL-ASTRA1B

8.2.1 BLE Connection

Once the board is registered, it is visible in “Your Astra view”, tapping on it the user has access to the other views:

Figure 46. BLE connection

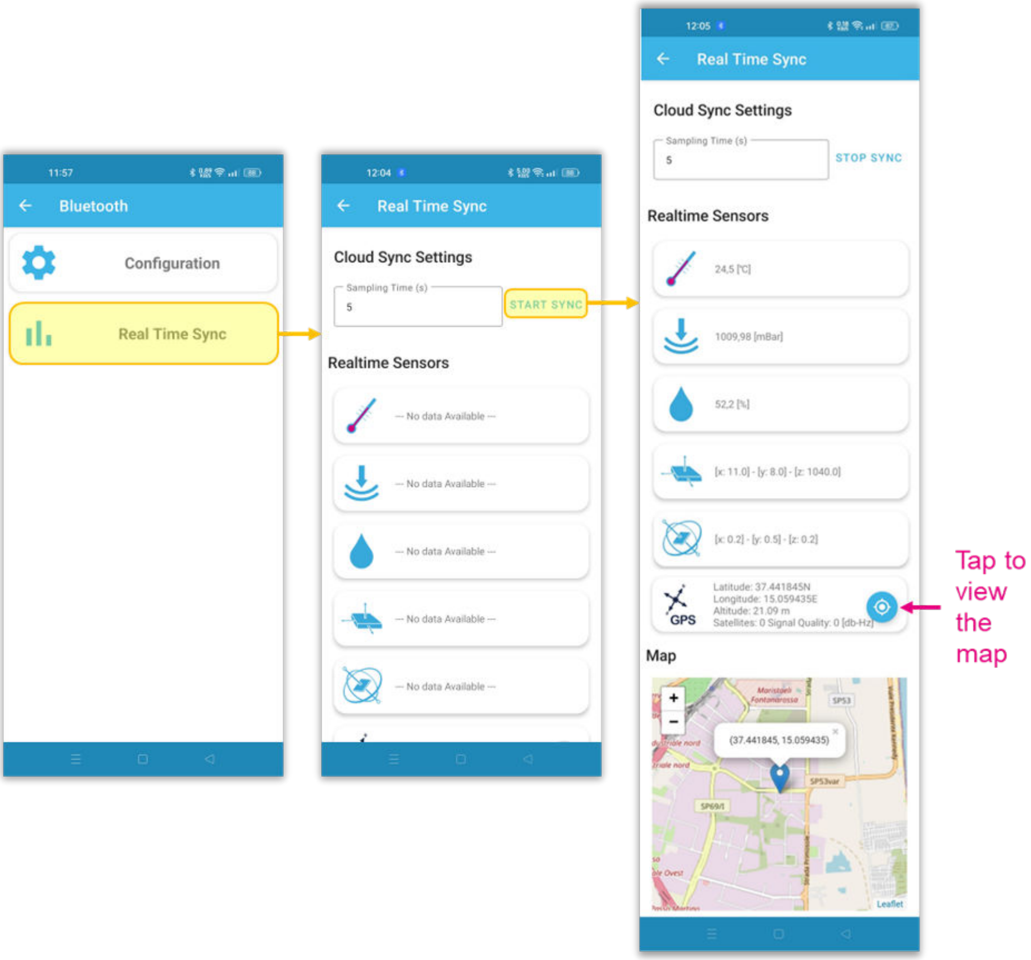


8.2.2 Data Acquisition

The mobile app allows the used to acquire data using BLE connectivity or NFC, for NFC refer to paragraph Section 6.1.1 How to access the NFC views.

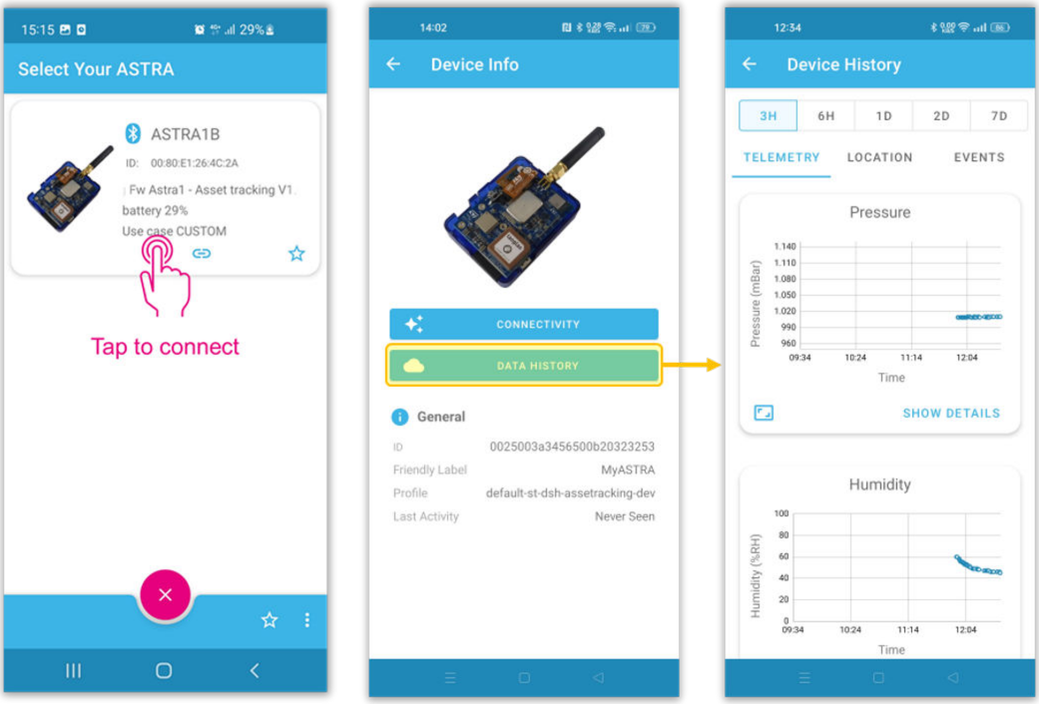
As soon as the user tap the “Start Sync”, the smartphone starts storing BLE data for a period of time equal to the "sampling time" value. Once this time is expired the data are forwarded to the dashboard (the real BLE frequency acquisition is set by FW).

Figure 47. BLE reat time data



Select [**DATA HISTORY**] to get graphs about the sensor data history of the device.

Figure 48. **Data history**

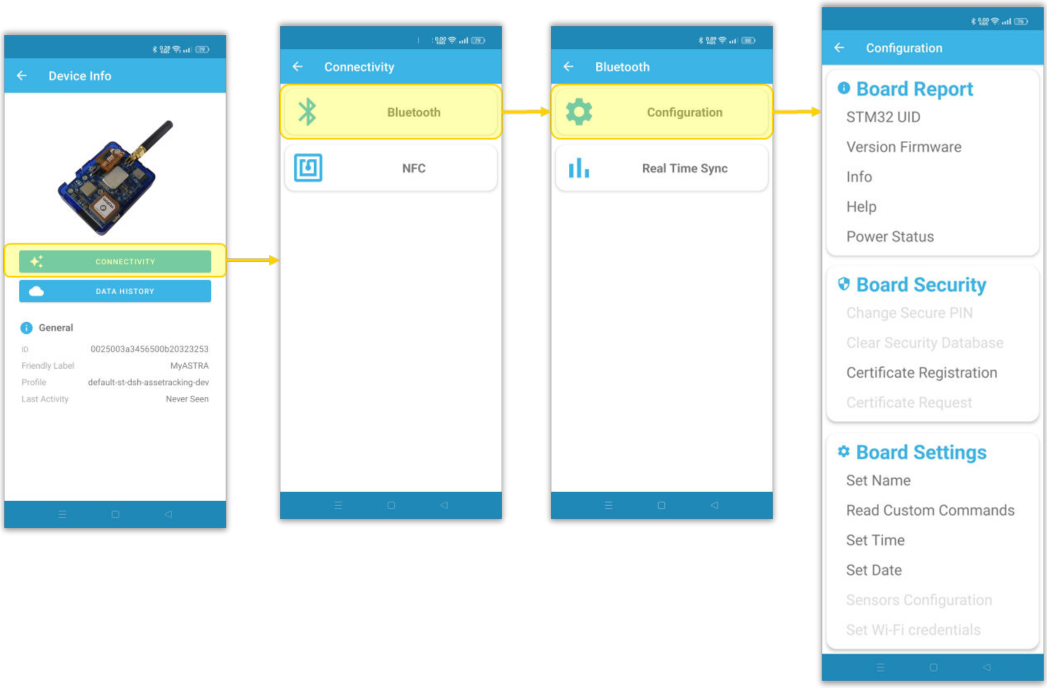


8.3 Board configuration

Configuration using NFC has already been discussed in paragraph Section 8.1.2 Device registration using mobile app.

To access BLE configuration Tab follow the path which is shown in the picture below:

Figure 49. **Board configuration menu**



From [**Board Report**], you can access the board and the firmware information.

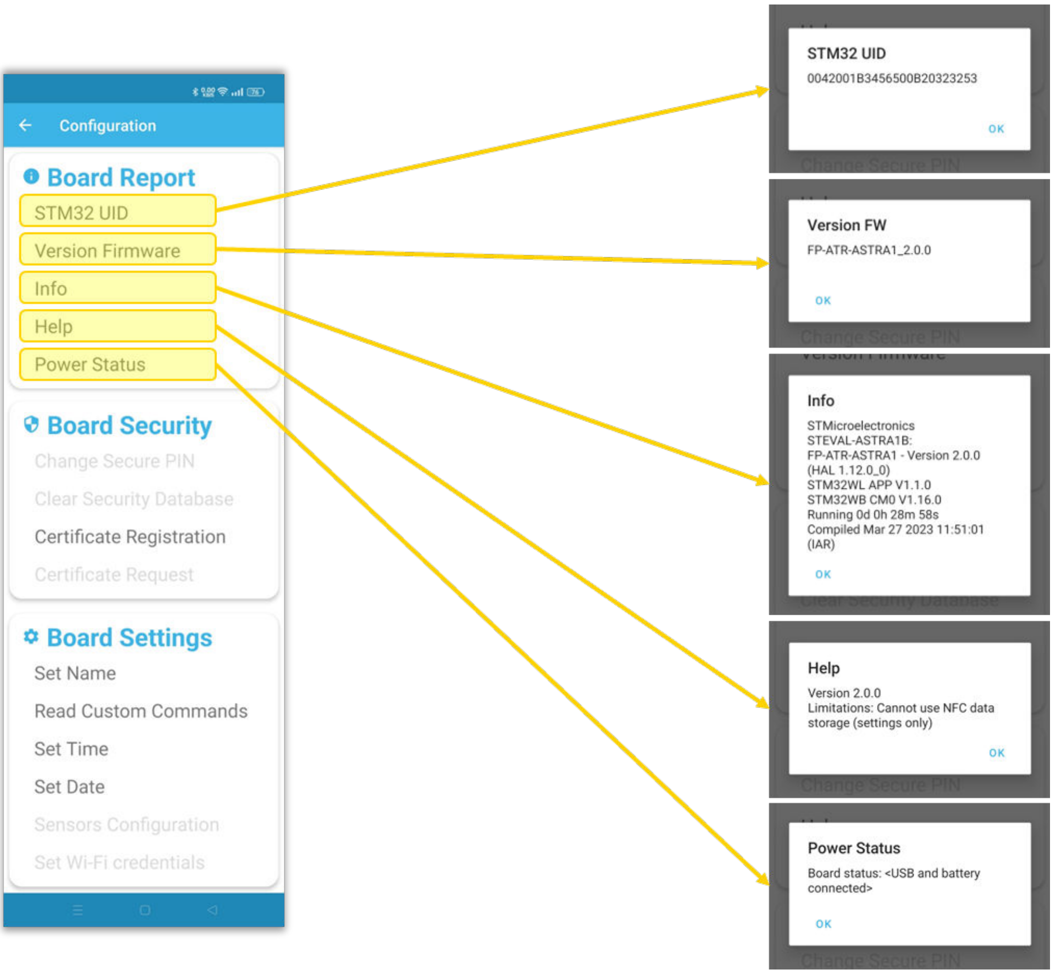


Figure 50. Board report command

From [**Board Settings**], you can access the settings.

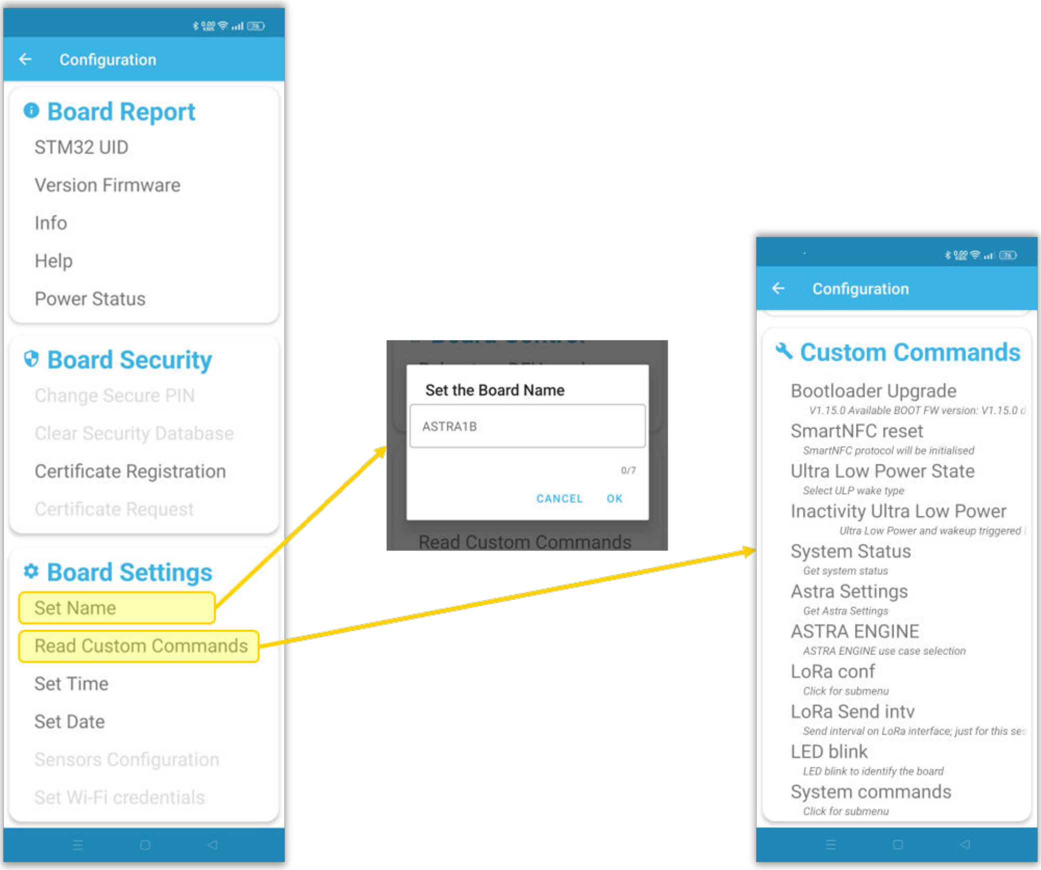
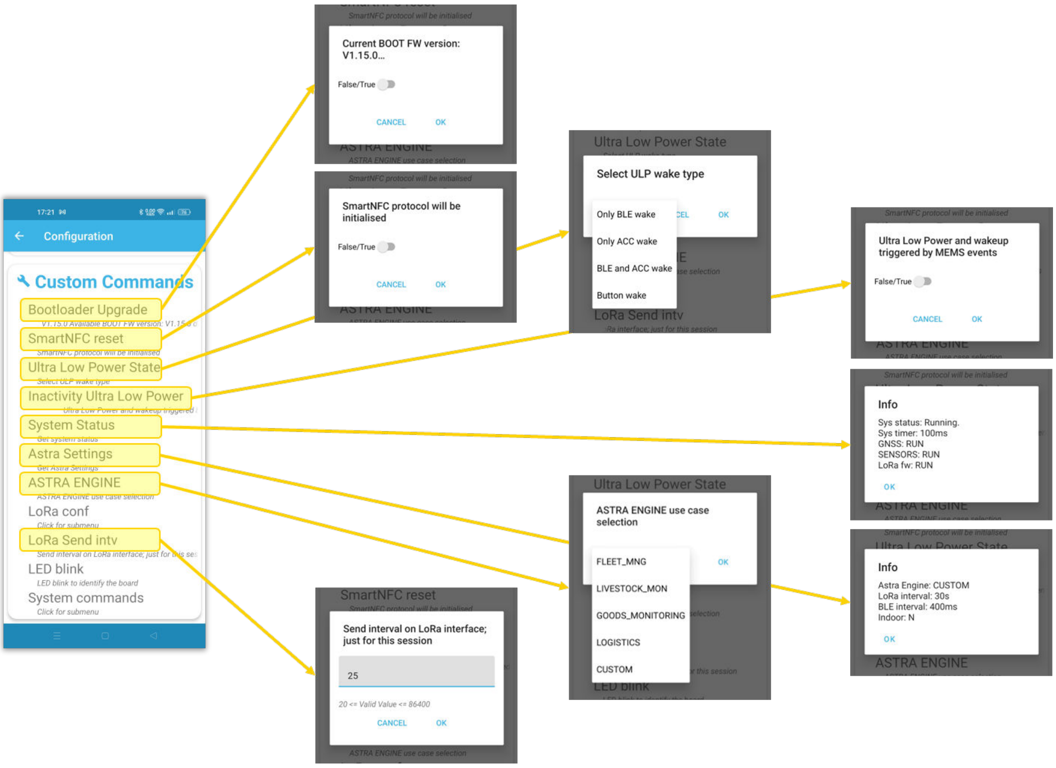


Figure 51. Board settings

8.3.1 Custom commands

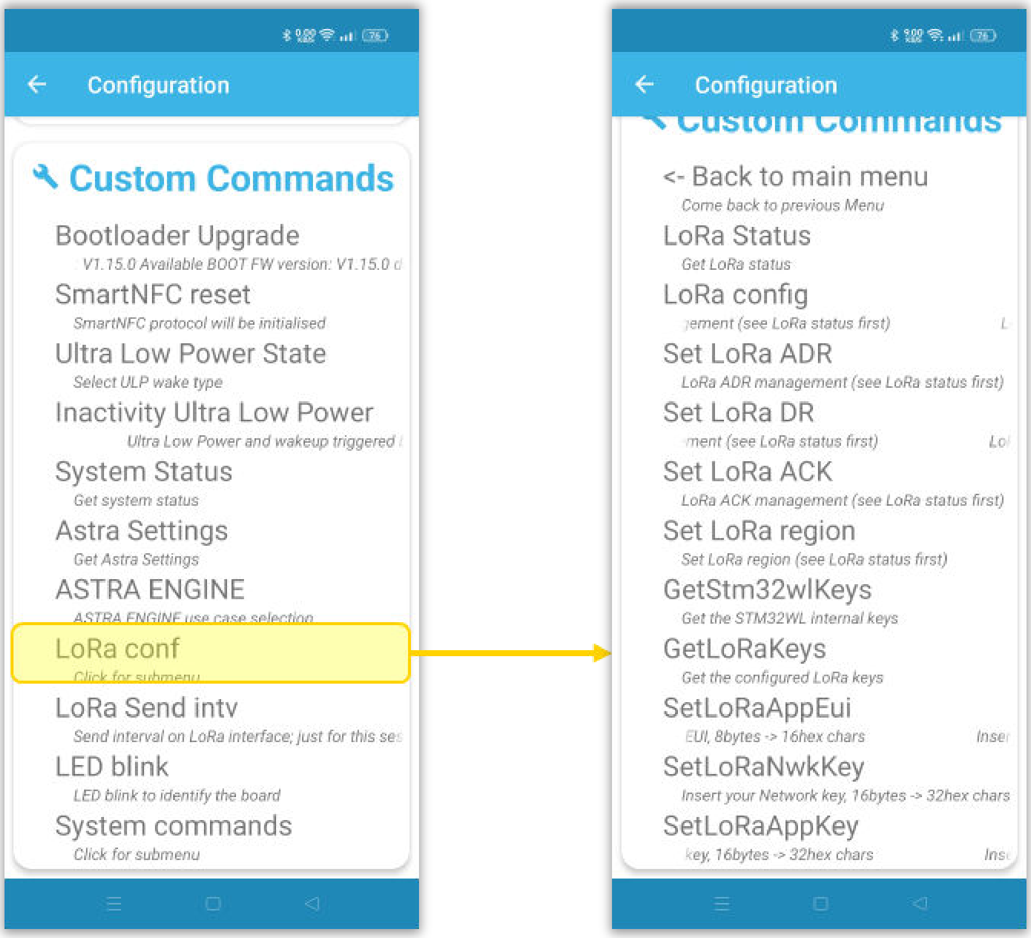
To get more specific information, access the [**Custom Commands**] view from [**Board Settings**] section.

Figure 52. **Custom commands**



The LoraWAN status and information are also available on the custom command tab. Access the [**LoRa conf**] view from [**Custom commands**] section.

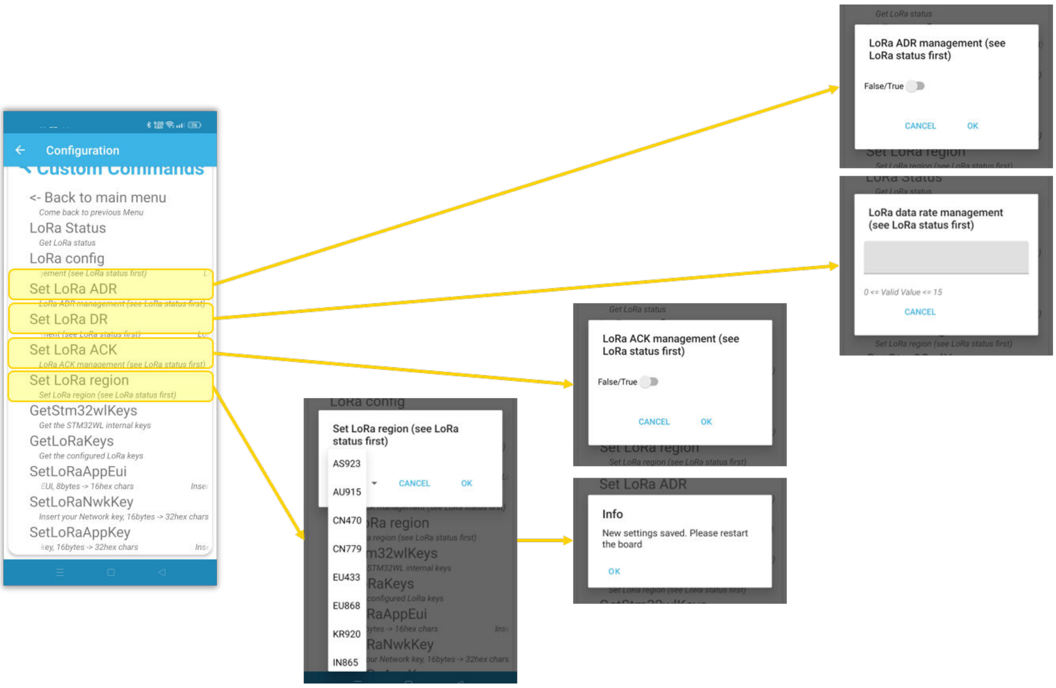
Figure 53. **LoRa configurations**



From [**LoRa conf**] submenu you can access the LoRa configuration parameters

Figure 54. **System Command - LoRA configurations**

Figure 55. System Command - LoRA configurations (cont.)



In the [**Custom Commands**], the [**System commands**] option allows viewing/configuring the system parameter, GNSS, the sensor configuration, etc.

Figure 56. **System commands**

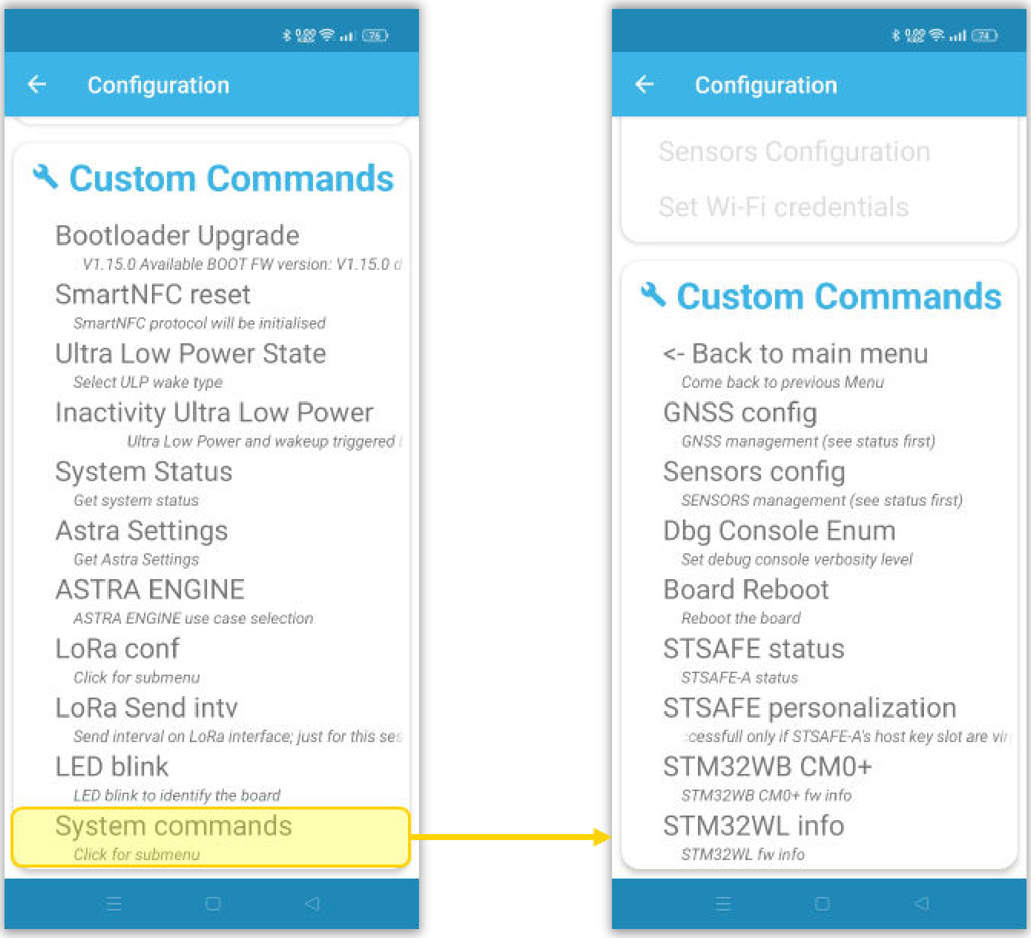


Figure 57. **System commands - details**

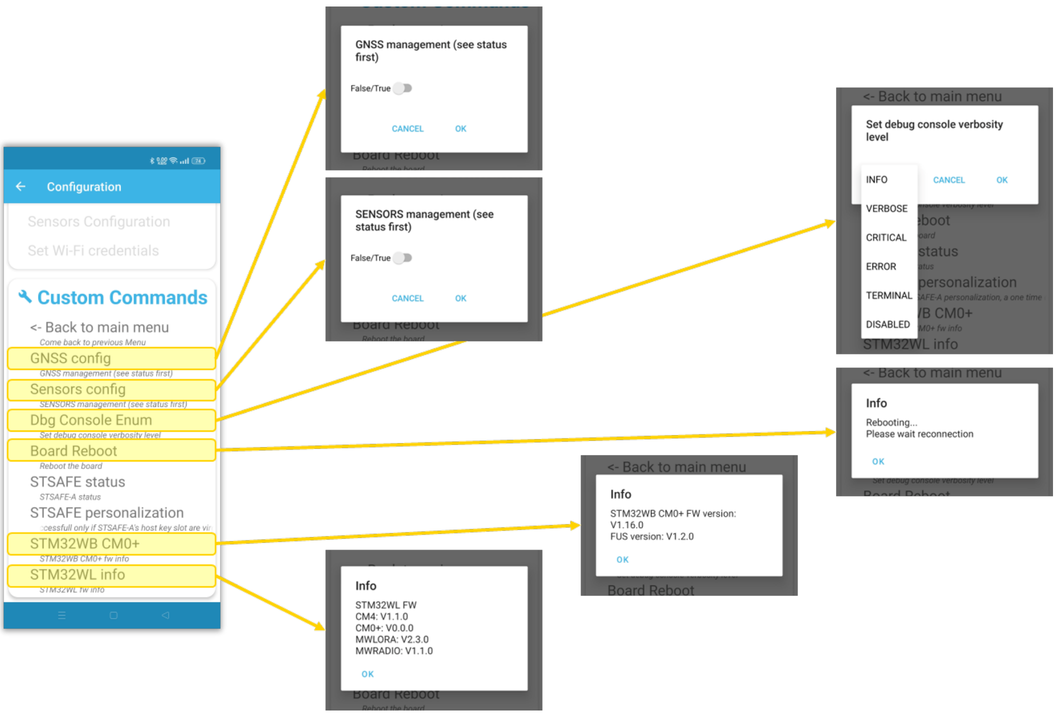
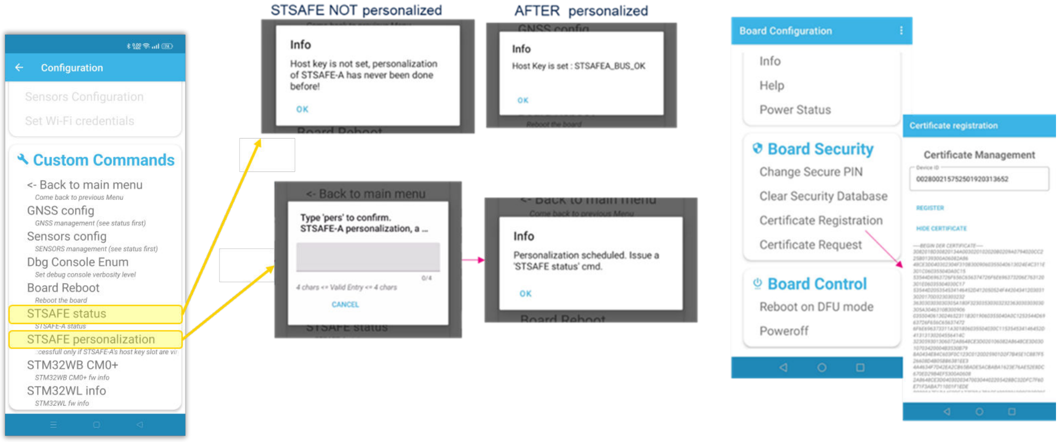
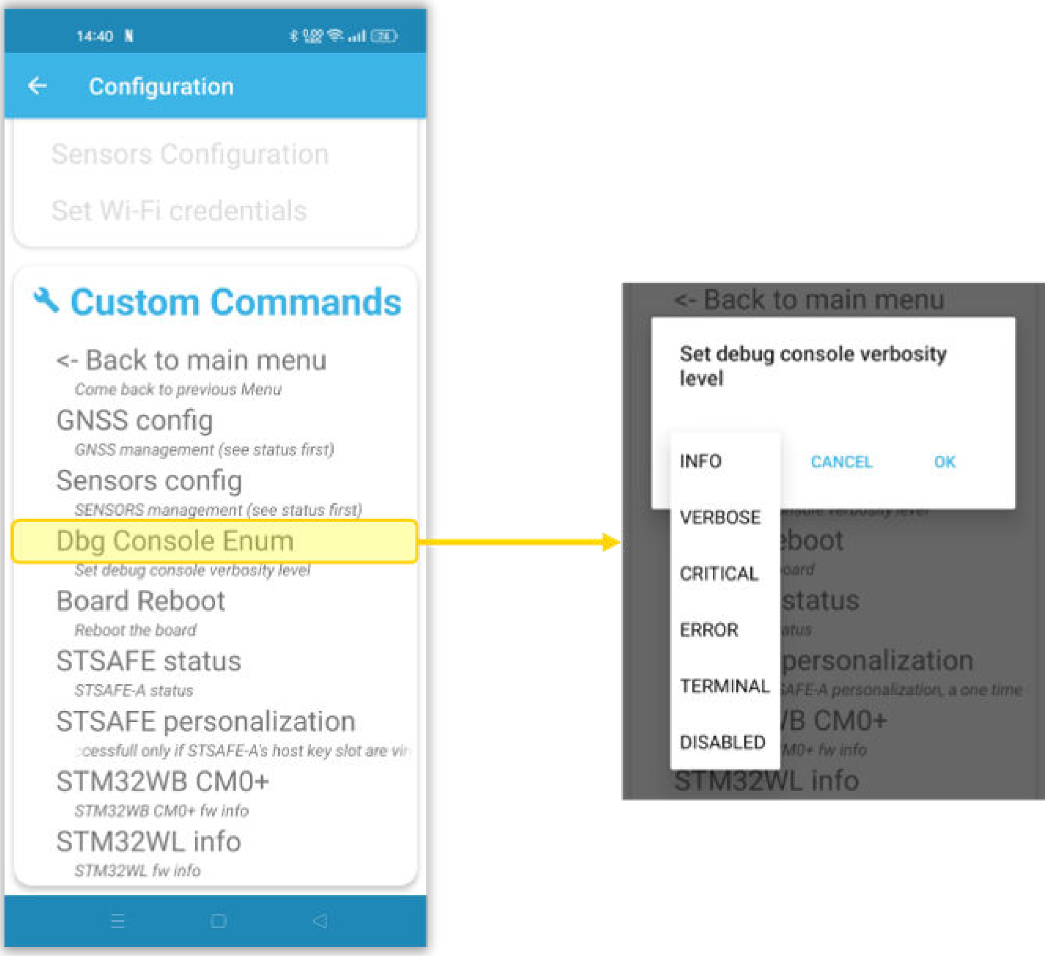


Figure 58. **System Command - STSafe personalization and show certificate : DER certificate is sent from**



Above all, the [**Dbg Console Enum**] command allows the user to set up the verbosity of the debug console. Refer to the figure below about changing it and refer to Section 9. to discover how to access it.

Figure 59. **How to change the debug console verbosity**



8.4 Asset tracking dashboard access and views

The following figures show an overview of the dashboard.

For further details, go to the relevant web page (DSH-ASSETRACKING).

Figure 60. DSH-ASSETRACKING login

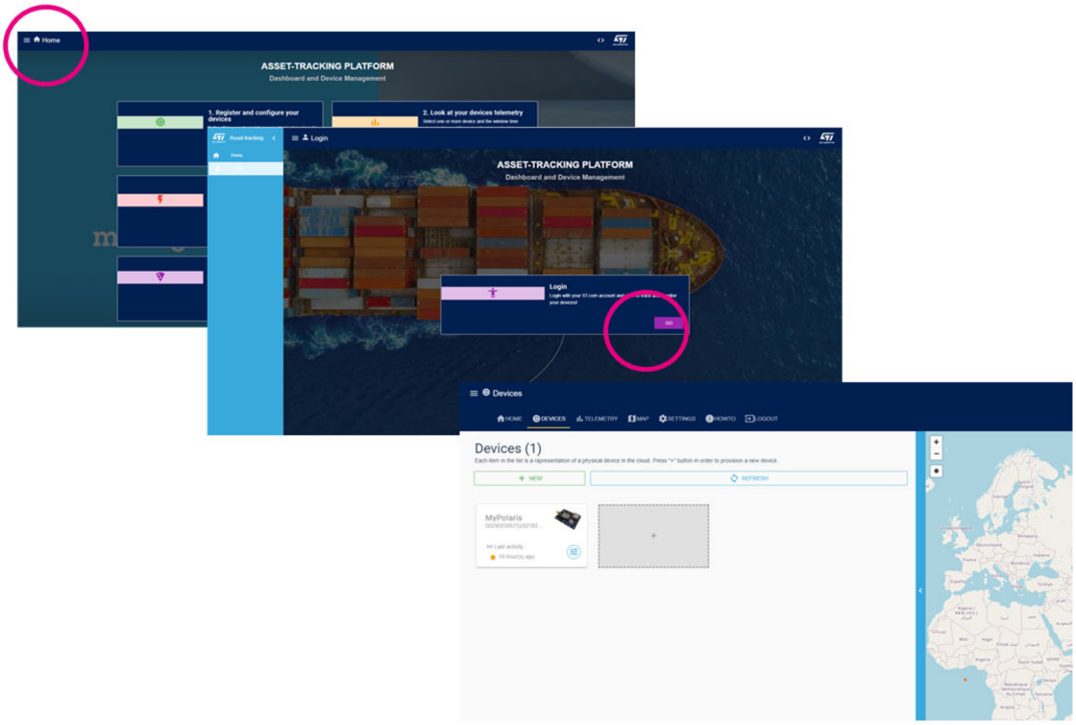
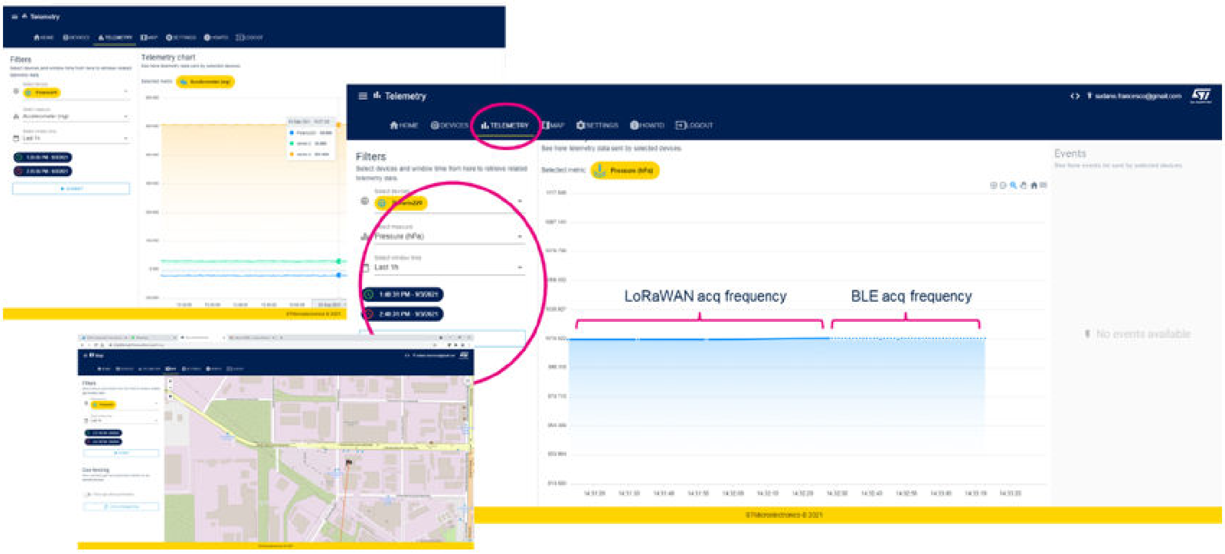


Figure 61. DSH-ASSETRACKING views



9 CLI, command line interface, and debug console on Bluetooth® Low Energy and USB

The FP-ATR-ASTRA1 firmware package implements a textual human interface via USB or Bluetooth® Low Energy.

For the USB interface, the communication device class (CDC) class, virtual COM port is used. It provides an input/output communication channel for reading and writing via a serial terminal on a PC and/or a smartphone. For further details, see UM2966.

Figure 62. Debug console - PC view (1 of 3)

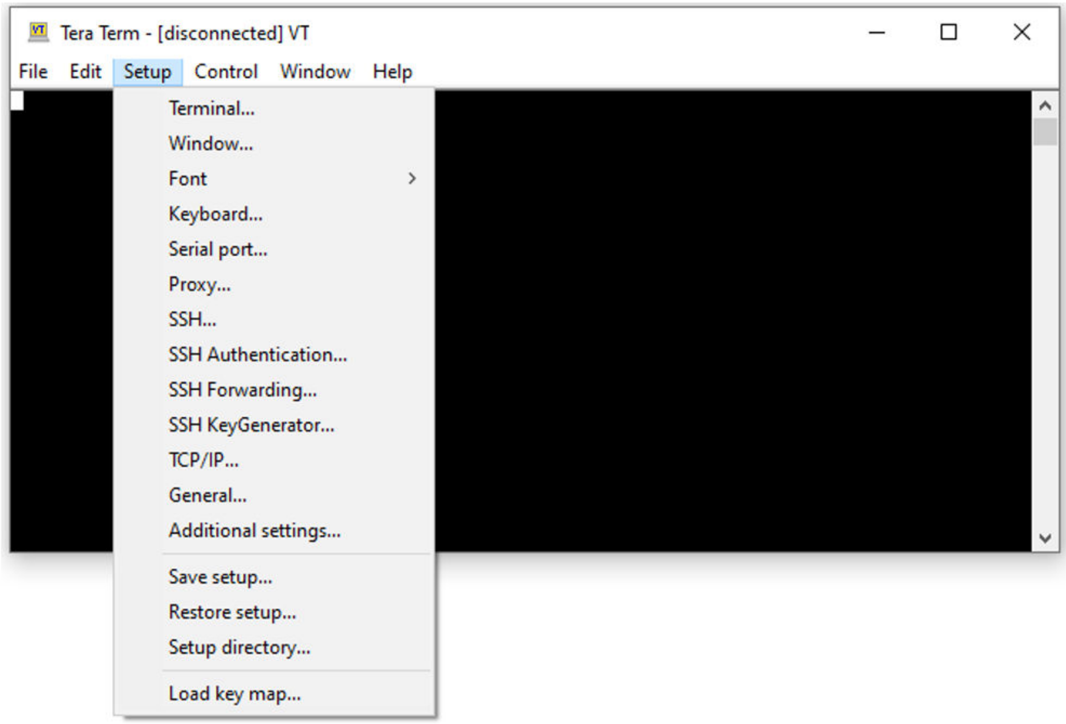


Figure 63. Debug console - PC view (2 of 3)

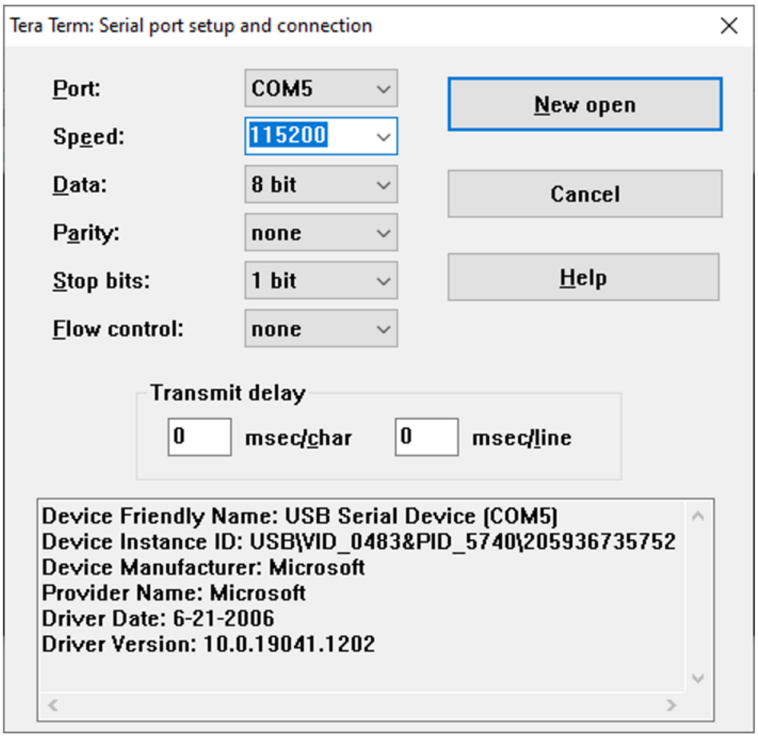


Figure 64. Debug console - PC view (3 of 3)

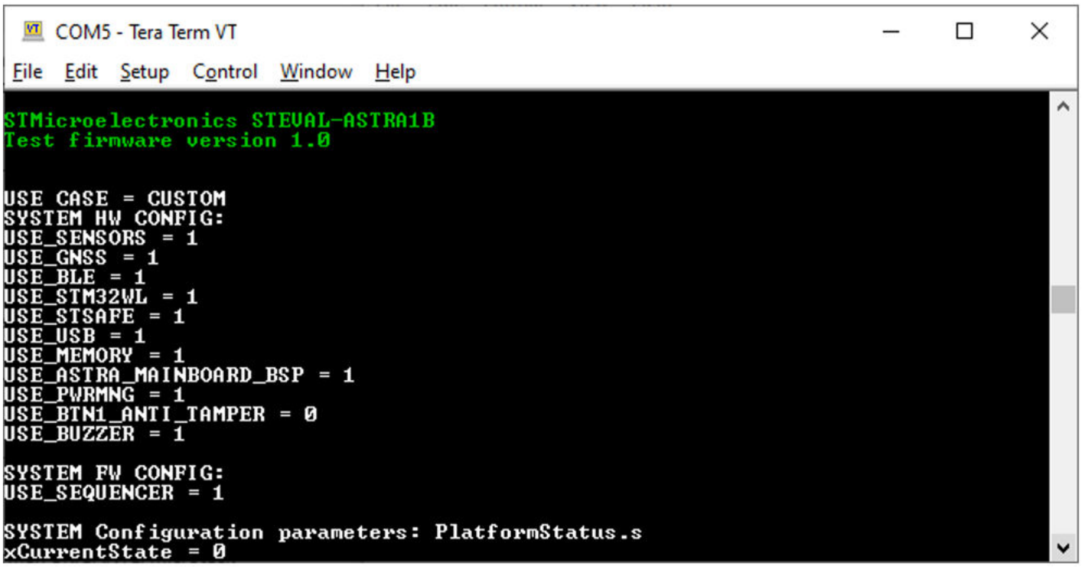
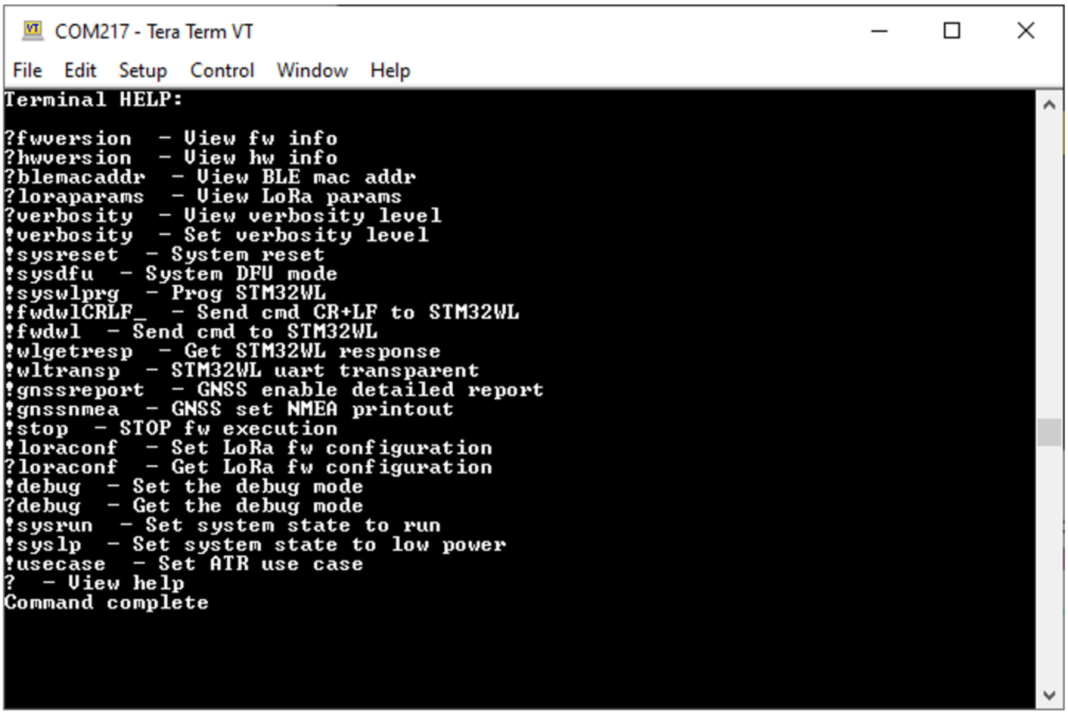


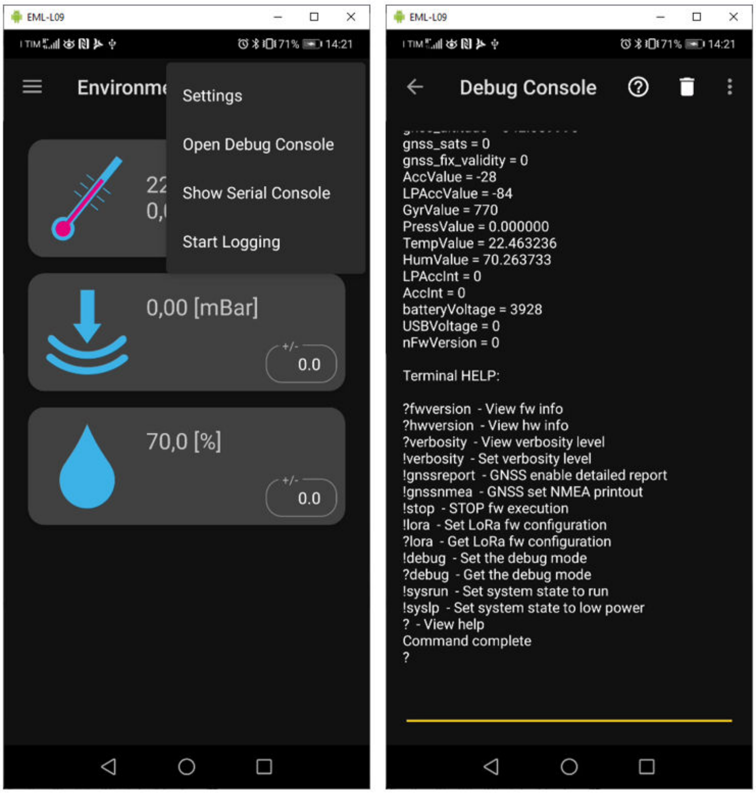
Figure 65. **Debug console - command list**



For the Bluetooth® Low Energy input/output textual communication, the debug console is used. The Bluetooth® Low Energy manager implements this characteristic and is able to exchange the user textual data.

The debug console is also available over Bluetooth® Low Energy using the STBLESensor app (the STAssettracking app does not support this feature).

Figure 66. **Debug console - app view**



The textual console can receive the user input and call the appropriate handler function. The Bluetooth® Low Energy and USB interfaces are used at the same time for the input. The responses are outputs on both interfaces, regardless of the one that received the command.

The handler functions are defined in the *Astra_sysmng.c* as follows:

```
/**
* @brief Command handler descriptor
*/
typedef struct s_cmd_struct {
  char cCmdString[20];                          //!< Command string
  char cCmdHelp[50];                            //!< Command help string
  pCmdHandlerFunc_t pCmdHandlerFunc;            //!< Command function pointer
  uint8_t bShowInHelp;                          //!< To show the function with '?' command
  uint8_t bPrintConfig;                         //!< To disable message printing after comman
d execution
} xCmdStruct_t;
```

Each command is identified by:

- a command string that identifies the string sent by the user to trigger the command; it can be followed by parameters
- a command help string that gives a brief description of the command when the command list is shown (see `?` command)
- the command function that is called to process the command
- the `bShowInHelp` flag to hide the function when the command list is shown (see `?` command)
- the `bPrintConfig` flag to suspend the printing (or reduce verbosity) after a command execution; this ensures that the command response is not overtaken by other traces

As a general rule, the command behavior is identified by the first character. `?` is the starting character for the *get* commands. `!` is the starting character for the *set* commands.

The following table lists the implemented commands with some additional information with respect to the brief description available with the `?` command.

Table 4. CLI command list

ASCII command	Short help	Set/Get	Description
`?`	Help	G	It shows this command list
`?fwversion`	View firmware information	G	It gets the firmware version
`?hwversion`	View hardware information	G	It gets the hardware version
`?blemacaddr`	View Bluetooth® Low Energy MAC address	G	It gets the Bluetooth® Low Energy MAC address
`?loraparams`	View LoRa parameters	G	It gets the LoRa connection parameters
`?verbosity`	View verbosity level	G	It gets the log messages verbosity level
`!verbosity`	Set verbosity level	S	It sets the log messages verbosity level. Usage: `!verbosity-X` where X is the verbosity level. The allowed values are: 1. `LEVEL_TERMINAL` 2. `LEVEL_ERROR` 3. `LEVEL_CRITICAL` 4. `LEVEL_VERBOSE` 5. `LEVEL_INFO`
`!sysreset`	System reset	S	It forces a system reset
`!sysdfu`	System DFU mode	S	It forces a system reboot in USB DFU mode

ASCII command	Short help	Set/Get	Description
`?sysblver`	System bootloader version	G	It returns the FOTA bootloader version
`!syswlprg`	Program STM32WL	S	It forces the STM32WL reboot in DFU mode
`!fwdwlCRLF_`	Send cmd CR+LF to STM32WL	S	It sends cmd CR+LF to the STM32WL microcontroller
`!fwdwl`	Send cmd to STM32WL	S	It forwards commands to the STM32WL microcontroller
`!wlgetresp`	Get STM32WL response	S	It gets responses from the STM32WL microcontroller
`!wltransp`	STM32WL UART transparent	S	This command makes the USB virtually connected to the STM32WL UART port. It means that the STM32WB forwards the AT commands received from the USB to the UART, which is used for the communication between the two microcontrollers
`?gnssstatus`	GNSS status report	G	It gets the GNSS status report
`!gnssreport`	GNSS enable detailed report	S	It enables/disables the GNSS detailed report. Usage: `!gnssreport-X` where X is the on/off word. Thus, the allowed values are: on, off
`!gnssnmea`	GNSS set NMEA printout	S	It enables/disables the NMEA printout. Usage: `!gnssnmea-X` where X is the on/off word. Thus, the allowed values are: on, off
`!stop`	Stop firmware execution	S	It stops the firmware execution
`!loraconf`	Set LoRa firmware configuration	S	It sets the LoRa mode. Usage: `!loraconf-X` where X is the LoRa mode. The allowed values are: • 0 that means disabled • 1 that means enabled
`?loraconf`	Get LoRa firmware configuration	G	It gest the LoRa mode
`!debug`	Set the debug mode	S	It sets the debug level. Usage: `!debug-X` where X is the debug level. The allowed values are: 0 LEVEL_NOTHING 1. LEVEL_TERMINAL 2. LEVEL_ERROR 3. LEVEL_CRITICAL 4. LEVEL_VERBOSE 5. LEVEL_INFO
`?debug`	Get the debug mode	G	It gets the debug level.
`!sysrun`	Set system state to run	S	It moves the system state to run
`!syslp`	Set system state to low power	S	It moves the system state to low power

ASCII command	Short help	Set/Get	Description
`!usecase`	Set ATR use case	S	It sets the use case. Usage: `!usecase-X` where X is the use case number. The allowed values are: 1. 0 FLEET_MNG 2. LIVESTOCK_MON 3. GOODS_MONITORING 4. LOGISTICS 5. CUSTOM

10 Firmware upgrade

10.1 STM32WB5MMG firmware upgrade

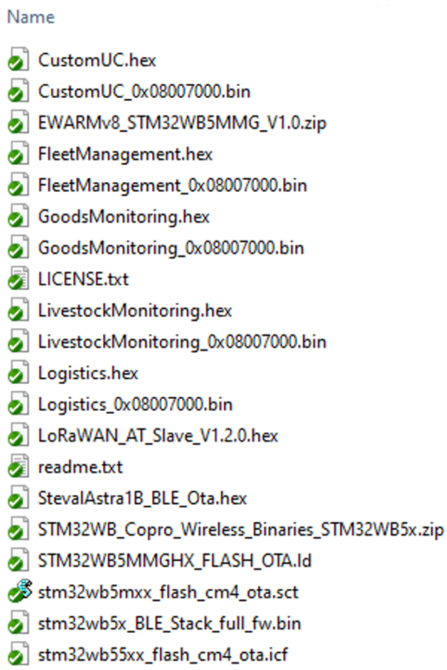
STM32WB5MMG is an ultra-low-power module with a dual core Arm Cortex®-M4 MCU and Cortex®-M0+.

In the STEVAL-ASTRA1B, the Cortex®-M4 hosts the FP-ATR-ASTRA1 application firmware, while the Cortex®-M0+ hosts the Bluetooth® Low Energy stack.

The firmware of the cores can be upgraded in different ways, as described in the following sections.

The *Projects\STEVAL-ASTRA1B\Applications\AssetTracking\Binaries* folder contains all the compiled codes.

Figure 67. Binaries folder



10.1.1 How to update the STM32WB5MMG M4 core via STLINK-V3MINI

To download the application code into the STM32WB5MMG M4 core via STLINK-V3MINI, follow the procedure below.

Step 1. Check the programming jumper position according to UM2966, table 4.

Step 2. Connect the STLINK-V3MINI to the STEVAL-ASTRA1B programming connector J300 (see UM2966, section 2.2, for further information).

Step 3. As the STEVAL-ASTRA1B loads a preprogrammed bootloader, to download your customized code without overwriting the bootloader, use the following file:
- *stm32wb55xx_flash_cm4_ota.icf* linker file to set the correct flash memory address space in IAR.

Step 4. Use STM32CubeProgrammer to update the firmware by choosing one binary file, which is in the "Binaries" folder or using your own. Follow the readme.txt file inside that folder for further details.

Step 5. To restore the bootloader with the *StevalAstra1B_BLE_Ota.hex* compiled file that is available in the binary folder, use STM32CubeProgrammer to program the bootloader at the 0x08000000 address.

10.1.2 How to update the STM32WB5MMG M0+ core via STLINK-V3MINI

To download the application code into the STM32WB5MMG M0+ core via STLINK-V3MINI, follow the procedure below.

Step 1. Check the programming jumper position according to UM2966, table 4.

Step 2. Connect the STLINK-V3MINI to the STEVAL-ASTRA1B programming connector J300 (see UM2966, section 2.2, for further information).

Step 3. Refer to the *Projects\STEVAL-ASTRA1B\Applications\AssetTracking\Binaries\STM32WB_Copro_Wireless_Binaries_STM32WB5x.zip* folder, which contains the *stm32wb5x_BLE_Stack_full_fw.bin* and the instructions to program the firmware.

For further details, see AN5185.

Note: *During the CM0+ upgrade procedure, the CM4 bootloader is overwritten. It is strongly recommended to upgrade the CM4 after this procedure.*

10.1.3 How to update the STM32WB5MMG M4 core via USB

To download the application code into the STM32WB5MMG M4 core via USB, follow the procedure below.

Step 1. Put STM32WB5MMG in the boot mode by sending the `!sysdfu` command.

Step 2. Use STM32CubeProgrammer to update the firmware by choosing one binary file, which is in the "Binaries" folder or using your own. Follow the readme.txt file inside that folder for further details.

10.1.4 How to update the STM32WB5MMG M0+ core via USB

To download the application code into the STM32WB5MMG M0+ core via USB, follow the procedure below.

Step 1. Put STM32WB5MMG in the boot mode by sending the `!sysdfu` command.

Step 2. Set the nBOOT0 option byte to "0" and nBOOT1 option byte to "1" in order to select "system flash" for the next reboot. Power cycle the device.

Step 3. Use STM32CubeProgrammer to program the *stm32wb5x_BLE_Stack_full_fw.bin* firmware.

Refer to the Projects\STEVALASTRA1B\Applications\AssetTracking\Binaries\STM32WB_Copro_Wireless_Binaries_STM32WB5x.zip folder, which contains the *stm32wb5x_BLE_Stack_full_fw.bin* and the instructions to program the firmware.
For further details, see AN5185.

Step 4. Set the nBOOT0 option byte to "1" and nBOOT1 option byte to "1" in order to select "main flash" for the next reboot. Power cycle the device.

10.1.5 Firmware upgrade over-the-air (FUOTA) for the STM32WB5MMG M4 core

To download the application code into the STM32WB5MMG M4 core via FUOTA, follow the procedure below.

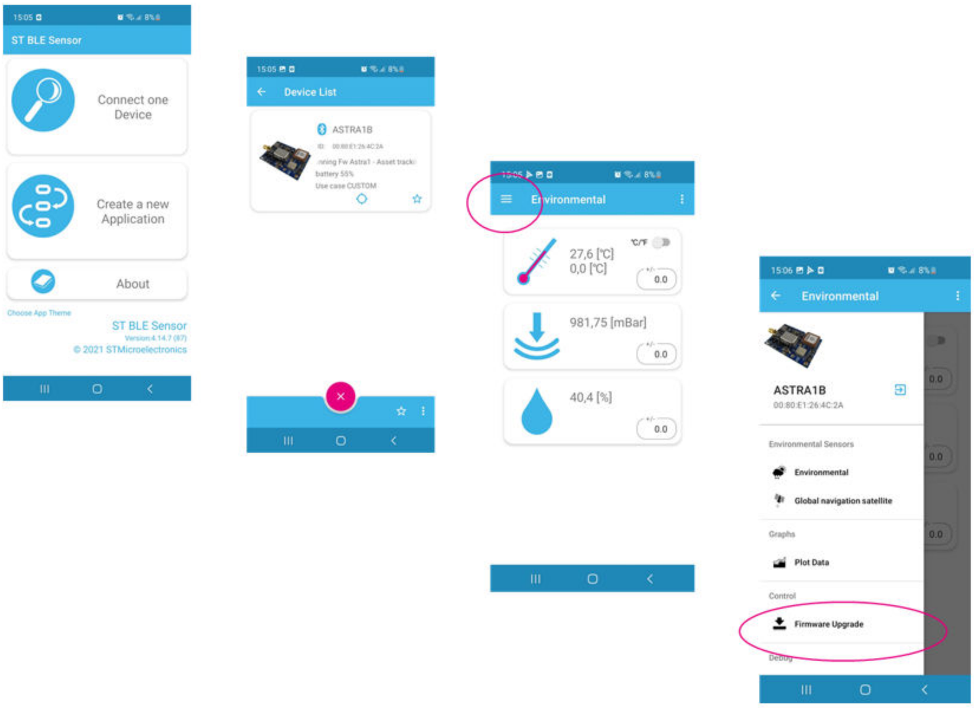
Before performing the firmware upgrade via the app, ensure that the bootloader version is “BLE_OtaFwVer1.13.0_HSE”, using the `?sysblver` command line.

If the bootloader version is different or you get the “No bootloader version found” message, or the command is not recognized, perform an update of the bootloader using firmware upgrade over USB or firmware updgrade via STLINK-V3MINI. The bootloader binary file is “StevalAstra1B_BLE_Ota.hex”, which is inside the "Binaries" folder.

Step 1. Open the STBLESensor app and connect your board.

Step 2. Go to the [**Firmware Upgrade**] section.

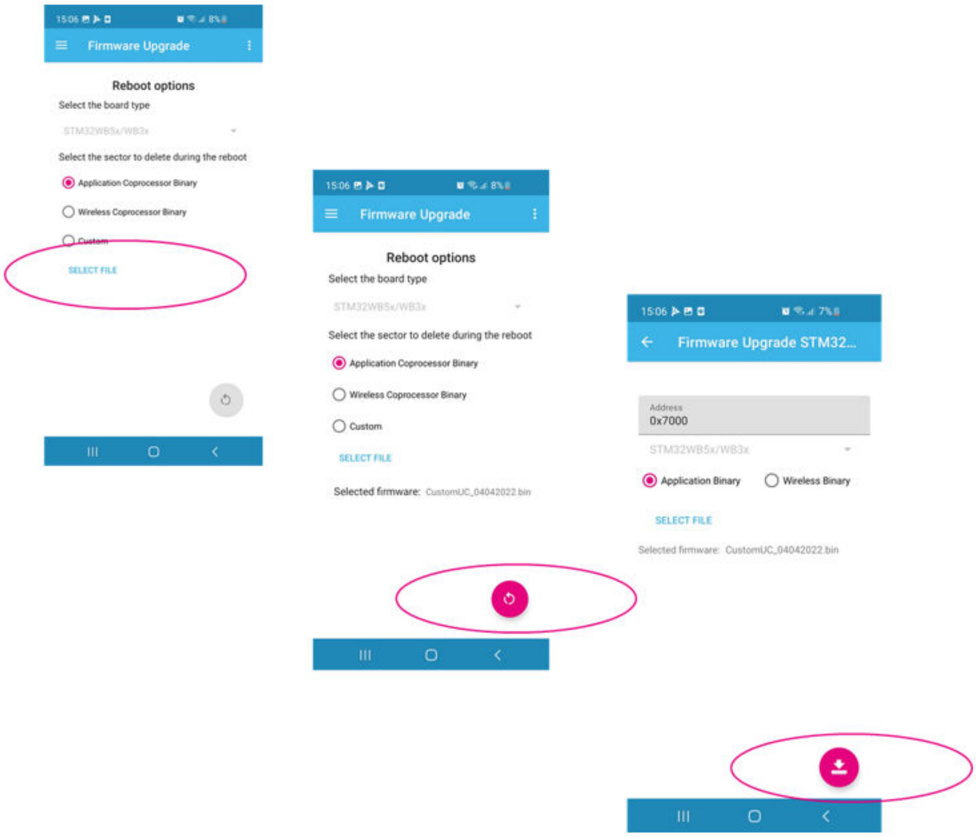
Figure 68. FUOTA procedure



Step 3. Select the application coprocessor binary (M4 core).

Step 4. Select the file (.bin) compiled with the BLE_OTA flag (see the selected project readme file, that is, *Projects\STEVAL-ASTRA1B\Applications\AssetTracking\CustomUC\readme.txt*).

Figure 69. **FUOTA for the M4 core**



10.1.6 Firmware upgrade over-the-air (FUOTA) for the STM32WB5MMG M0+ core

To download the application code into the STM32WB5MMG M0+ core via FUOTA, follow the procedure below.

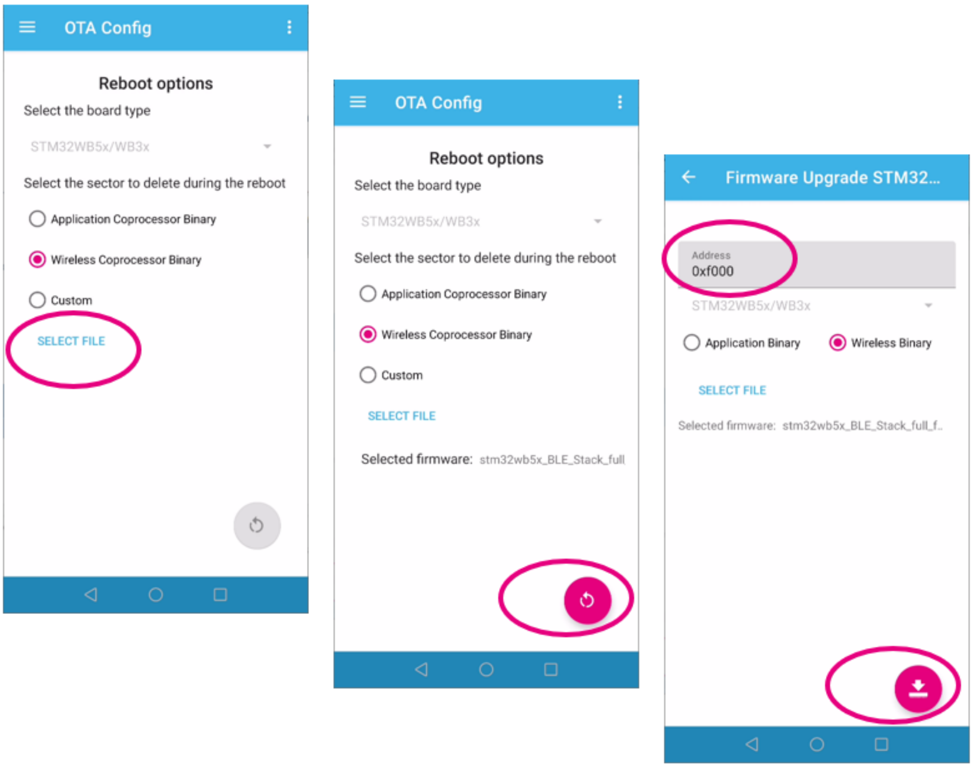
Step 1. Open the STBLESensor app and connect your board.

Step 2. Go to the [**Firmware Upgrade**] section.

Step 3. Select the wireless coprocessor binary (M0+ core).

Step 4. Download the *stm32wb5x_BLE_Stack_full_fw.bin* file from *ASTRA1B\Applications\AssetTracking\Binaries\STM32WB_Copro_Wireless_Binaries_STM32WB5x.zip* .

Figure 70. FUOTA for the M0+ core



Important: *Do not change the 0xf000 address.*

10.2 STM32WL55JC firmware upgrade

10.2.1 How to update the STM32WL55JC M4 or single core via STLINK-V3MINI

The STM32WL55JC of the STEVAL-ASTRA1B board comes preprogrammed with the AT command slave firmware. Anyway, firmware upgrade is possible.

To download the application code into the STM32WL55JC M4 core via STLINK-V3MINI, follow the procedure below.

Step 1. Check the programming jumper position according to UM2966, table 4.

Step 2. Connect the STLINK-V3MINI to the STEVAL-ASTRA1B programming connector J300 (see UM2966, section 2.2, for further information).

Step 3. Suspend the LoRa processing on the STM32WL55JC.

Step 3a. Send the `!loraconf-0` command through the debug console USB (or via Bluetooth® Low Energy in the STBLESensor app).

Step 3b. Issue the custom command [**Board settings**]>[**Read custom commands**]>[**LoRa conf**]>[**LoRa config**]>[**Set to FALSE**].

Step 4. Use the STM32CubeProgrammer programmer and download the *LoRaWAN_AT_Slave_V1.X.X.hex* file.

Revision history

Table 5. Document revision history

Date	Revision	Changes
22-Apr-2022	1	Initial release.
26-Oct-2022	2	Updated introduction, *Section 1.1 Overview, Section 1.2 Architecture, Section 2.1 Firmware library structure, Section 2.2 State machine, Section 2.3 HMI: LEDs and buttons, Section 4.2 Relevant files, Section 4.2.1 app_astra.c/.h, Section 4.2.2 astra_confmng.c/.h, Section 4.2.3 astra_datamng.c/.h, Section 4.2.5 SM_APP.c/.h, Section 5.1 STEVAL-ASTRA1B advertising packet, Section 5.2 Data exchange, Section 7 LoRaWAN® stack, Section 10.1.1 How to update the STM32WB5MMG M4 core via STLINK-V3MINI, Section 10.1.2 How to update the STM32WB5MMG M0+ core via STLINKV3MINI, Section 10.1.6 Firmware upgrade over-the-air (FUOTA) for the STM32WB5MMG M0+ core, and Section 10.2.1 How to update the STM32WL55JC M4 or single core via STLINK-V3MINI.* Added *Section 3 FP-ATR-ASTRA1 in STM32CubeMX, Section 3.1 Prerequisites, Section 3.2 Board selector, Section 3.2.1 Board selector with peripherals in default mode, Section 3.2.2 Board selector without peripherals mode, Section 3.3 FP-ATR-ASTRA1 pack dependencies, Section 3.3.1 Middleware, Section 3.3.2 Embedded utilities, Section 3.3.3 STM32CubeMX expansion packs, Section 3.4 FP-ATR-ASTRA1 customization, Section 3.4.1 [**Platform Settings**], Section 3.4.2 [**Parameter Settings**], Section 3.4.3 [**ASTRA ENGINE**] settings, Section 3.5 Create your own project with STM32CubeMX example selector, Section 4.1 Overview, and Section 4.2.4 astra_sysmng.c/.h.*
11-Jul-2023	3	Added Section 6 NFC dynamic memory. Updated Section 2.2 State machine, Section 8 Using the STEVAL-ASTRA1B with the STAssetTracking app and the DSH-ASSETRACKING dashboard.

Contents

List of tables

List of figures