life.augmented

Hands-on

Build basic p2pServer application and connect

STM32

Agenda

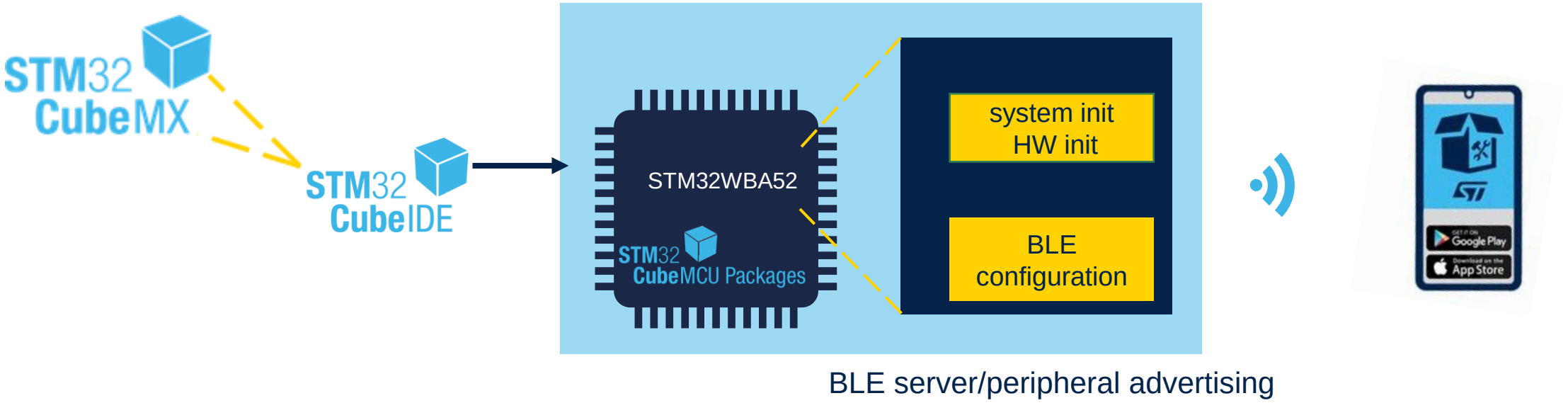
1. Hands-on presentation
2. Step 1: STM32CubeMX initialization for STM32WBA Nucleo board
3. Step2 : Advertising and BLE application configuration and explanation
4. Step 3 : Code generation and user application code
5. Step 4 : Adding logs
6. Step 5 : Adding service, profile and characteristic

life.augmented

Hands-On presentation

Purpose

- The purpose is to start from WB5A52 chipset level and build a basic server (p2pServer) application using STM32CubeMX and associated STM32CubeIDE
- In this first part, focus is to get device visible and connectable from my smartphone

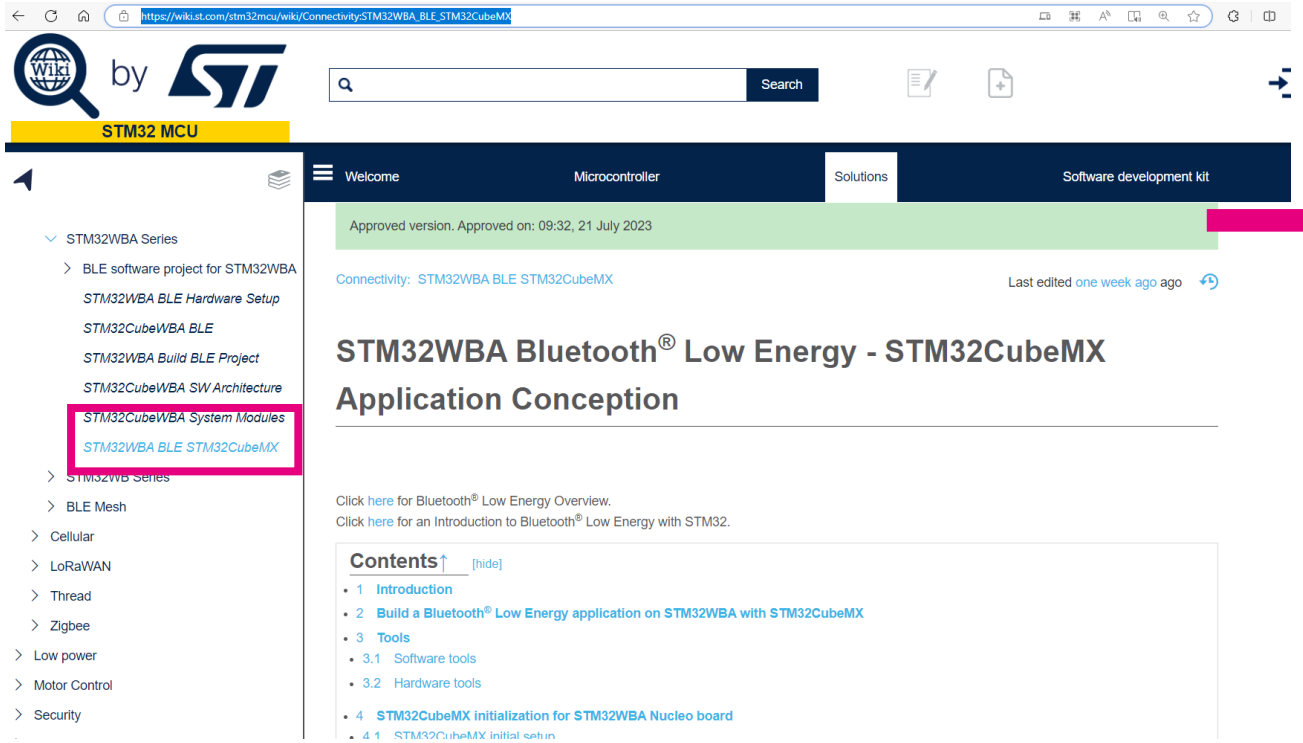


BLE server/peripheral advertising

Unpack NUCLEO-WBA52, plug to laptop,
install your favorite ST BLE ToolBox App and Let's start !

Source

Hands-On based on
https://wiki.st.com/stm32mcu/wiki/Connectivity:STM32WBA_BLE_STM32CubeMX



step by step guideline to build a BLE peripheral application

BLE server/peripheral
exposing data to central device

life.augmented

Step 1 : STM32CubeMX initialization for STM32WBA Nucleo board

life.augmented

STM32CubeMx capabilties

STM32CubeMx allow to start design within 3 options

1	**Example application** complete application running over NUCLEO
2	**Board level** all the hardware is already configured (NUCLEO_WBA52)
3	**Chipset level** require to configure your HW (PCB) & your application

→ STM32WBA wiki page focus

↓ Hands-on focus. As customer let's build my own App

STM32CubeMx design from chispet level complete journey

STM32CubeMx initialisation for STM32WBA Nucleo board

STM32WBA IPs & peripherals configuration

Clock Tree configuration

BLE configuration : Advertising, Service, Characteristic

Code generation & application code management over CubeIDE

Hands-on Focus

STM32CubeMx design from chispet level Hands-on focus (1/2)

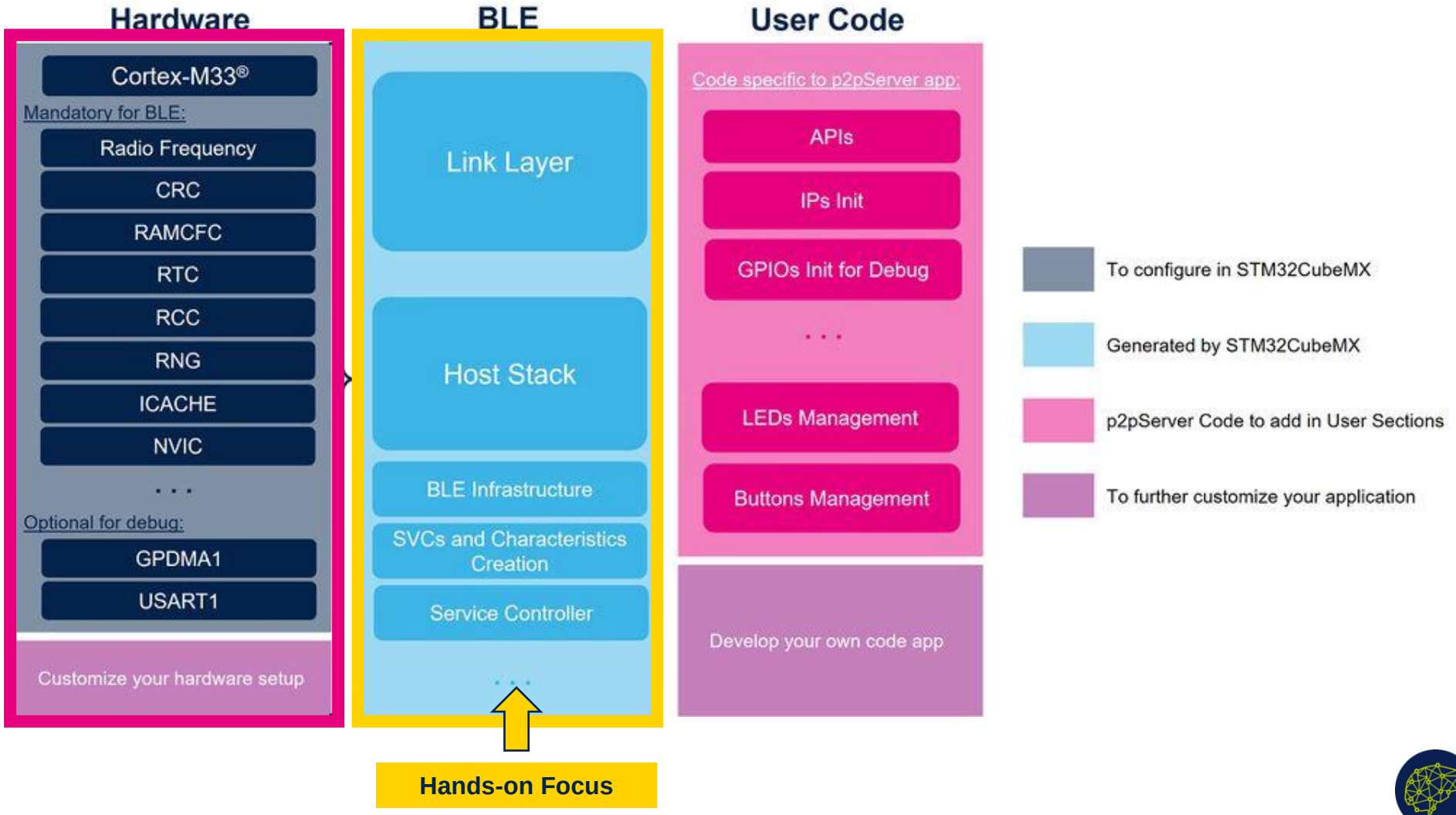
3	**Chipset level** require to configure your HW (PCB) & your application

To ease Hands-on session use Hands-on_WS_WBA52.ioc
All HW IPs & required peripheral to use RF are already initialized : NVIC, RNG, RCC,...
Thanks to Hands-on_WS_WBA52.ioc let's focus on BLE application design

Copy Hands-on_WS_WBA52.ioc on your local repository :
example : C:\users\...\STM32WBA_WS\project

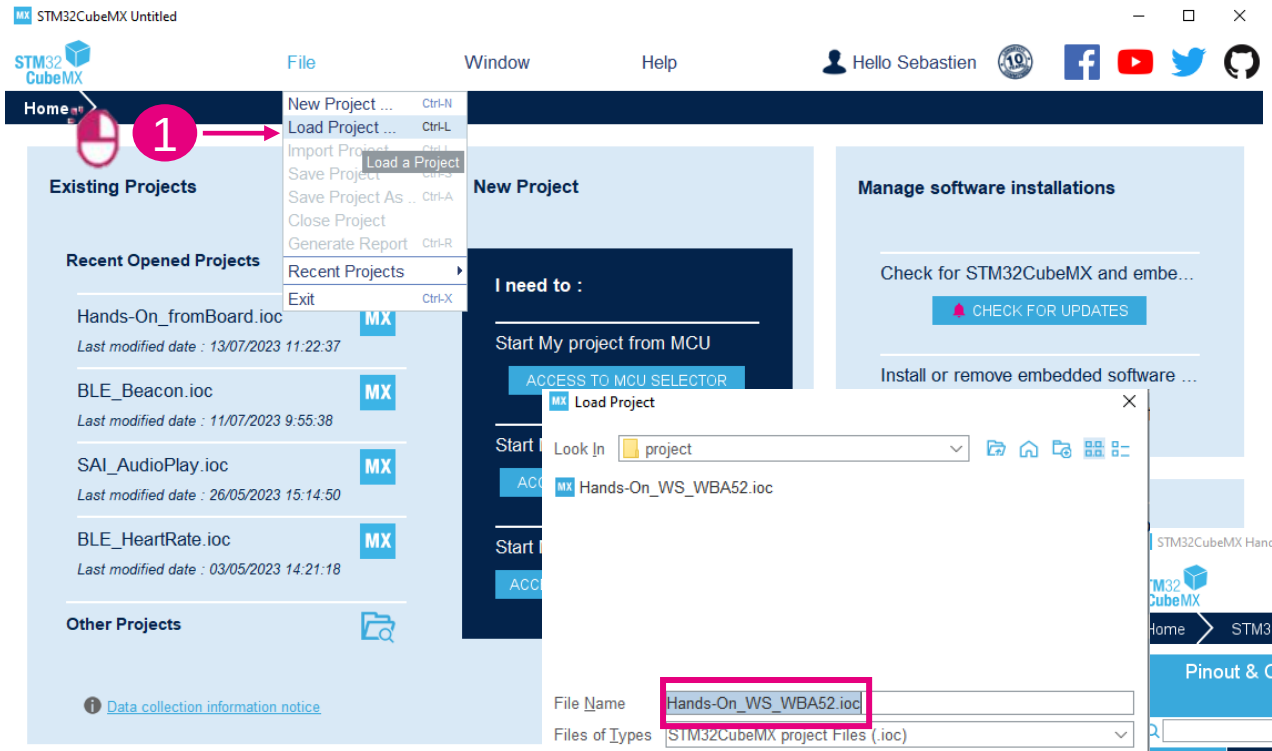
STM32CubeMx design from chispet level Hands-on focus (2/2)

Hands-On_WS_WBA52.ioc

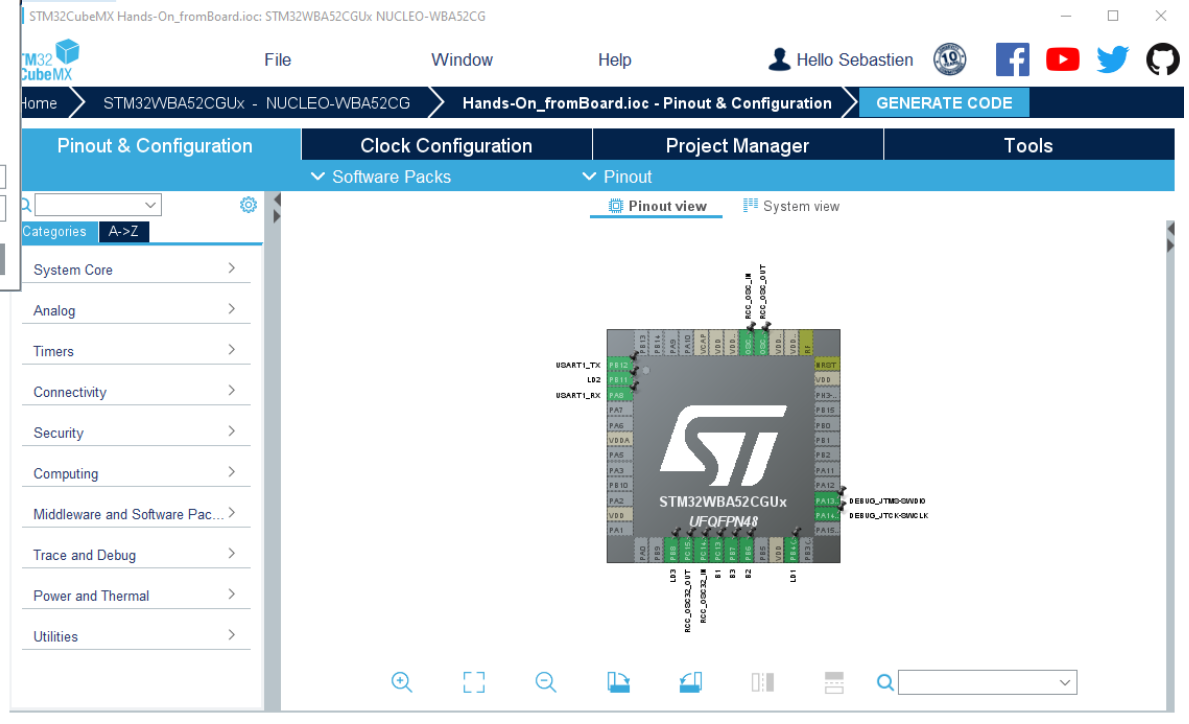


Start STM32CubeMX

CubeMx 6.9.0



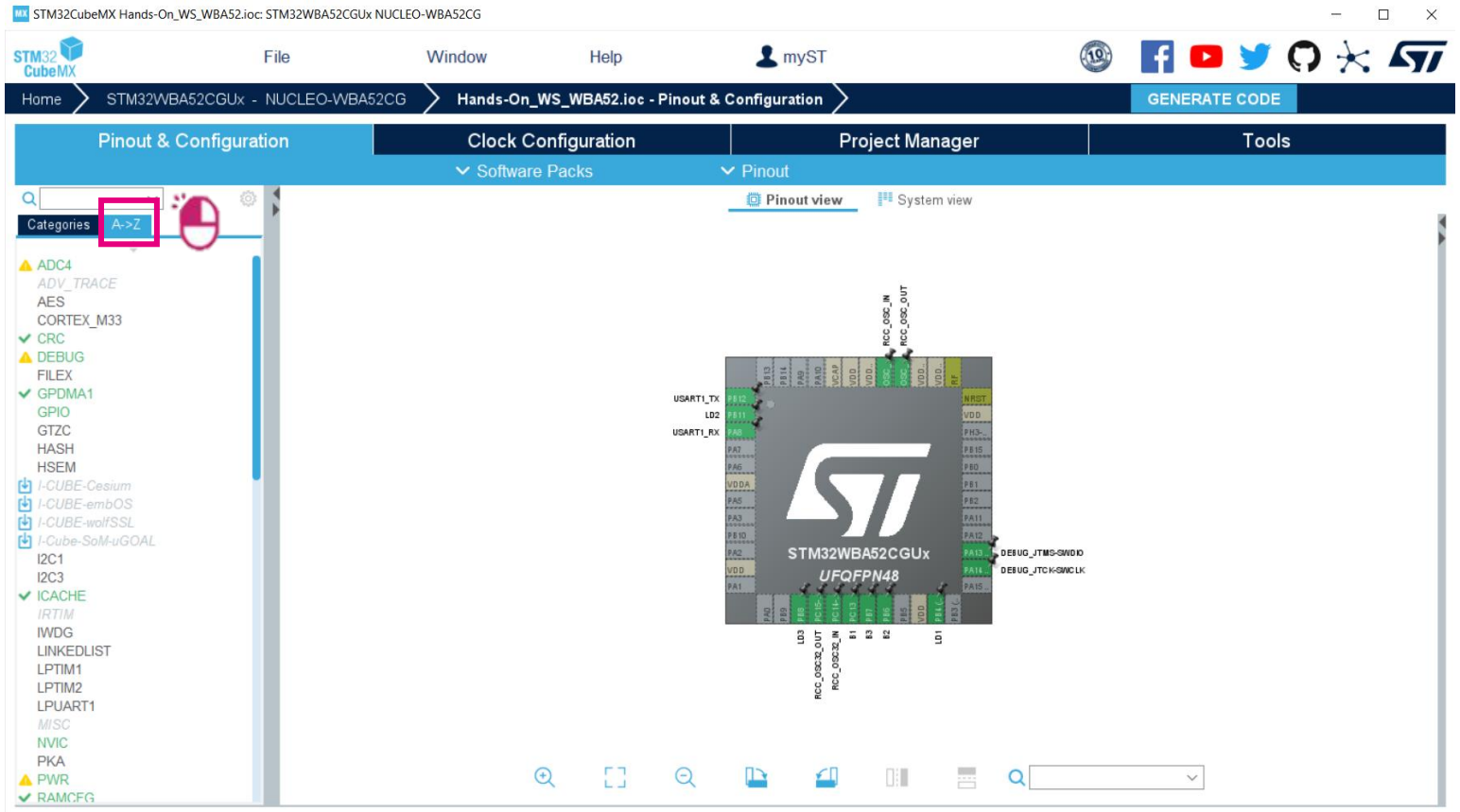
2



life.augmented

Peripherals in place to start BLE configuration !

Hands-On_WS_WBA52.ioc



Hands-On_WS_WBA52.ioc

- HW configuraton
- enable STM32_WPAN (**BLE middleware activation**)

Peripherals in place to start BLE configuration ! Wiki explanations

https://wiki.st.com/stm32mcu/wiki/Connectivity:STM32WBA_BLE_STM32CubeMX

BLE activation

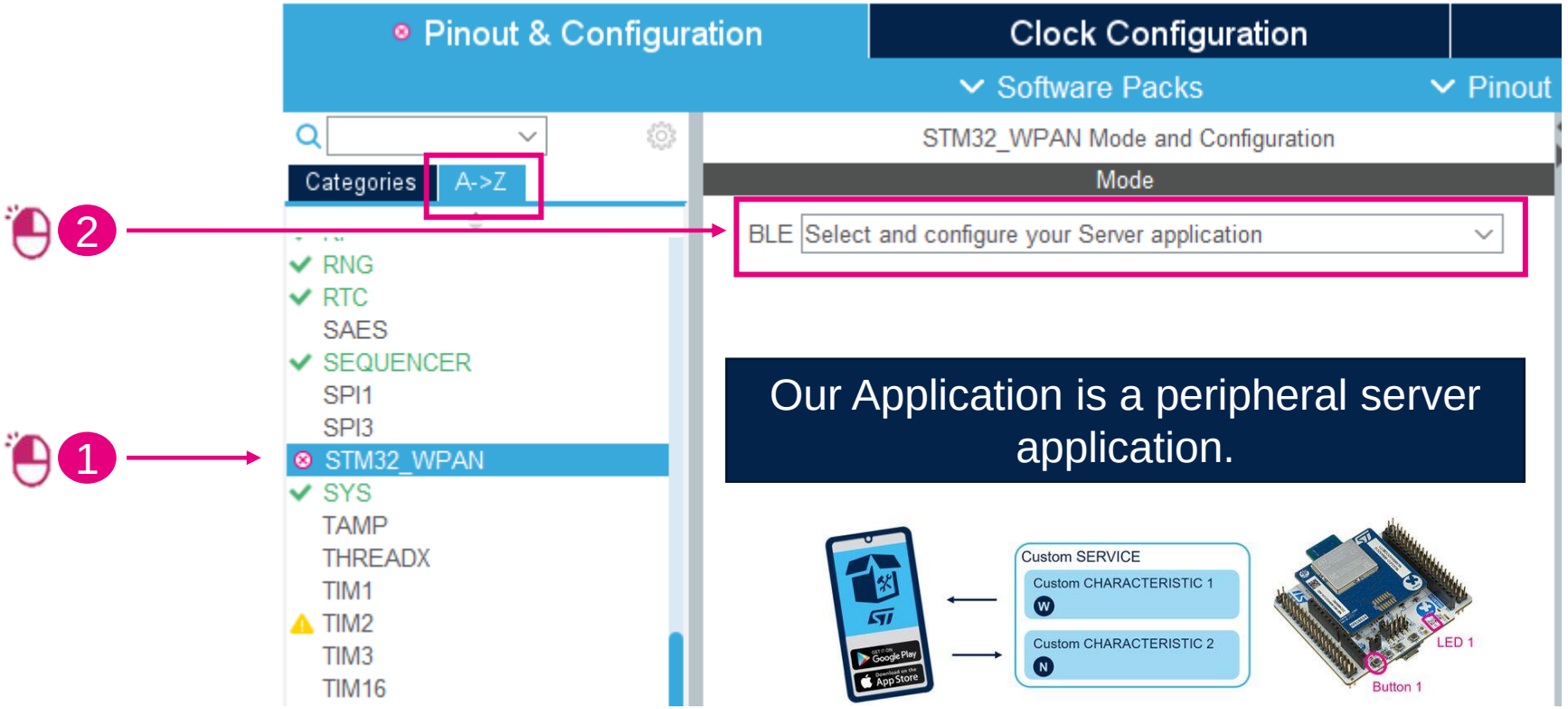
debug

ADC4	By default, PHY calibration is based on temperature. Therefore, the temperature sensor channel must be activated.
CRC	The cyclic redundancy check is used to verify Bluetooth® Low Energy data transmission or storage integrity.
RAMCFG	Activating an SRAM is mandatory for the application. We dynamically modify the RAM configuration (System Clock Manager (SCM) module). This allows us to manage cases where we use low power, for example.
ICACHE	The instruction cache (ICACHE) is introduced on the C-AHB code bus of the ARM Cortex-M33® processor to improve performance when fetching instructions and data from internal memories.
RNG	The random number generator (RNG) provides the application with full entropy outputs as 32-bit samples. It is necessary to activate it, because the link layer regularly requests RNG.
RCC	Reset and Clock Control manages the different kind of reset and generates all clocks for the bus and peripherals.
RF	Activating an SRAM is mandatory for the application. We dynamically modify the RAM configuration (System Clock Manager (SCM) module). This allows us to manage cases where we use low power, for example.
RTC	The real-time clock (RTC) provides an automatic wake-up to manage all low-power modes.
NVIC	All interrupts including the core exceptions are managed by the nested vectored interrupt controller (NVIC).
USART1	USART1 is enabled to allow the display of traces on a terminal.
GPDMA1	The general purpose direct memory access controller (GPDMA) is used to perform programmable data transfers between memory-mapped peripherals and/or memories via linked-list, upon the control of an off-loaded CPU.

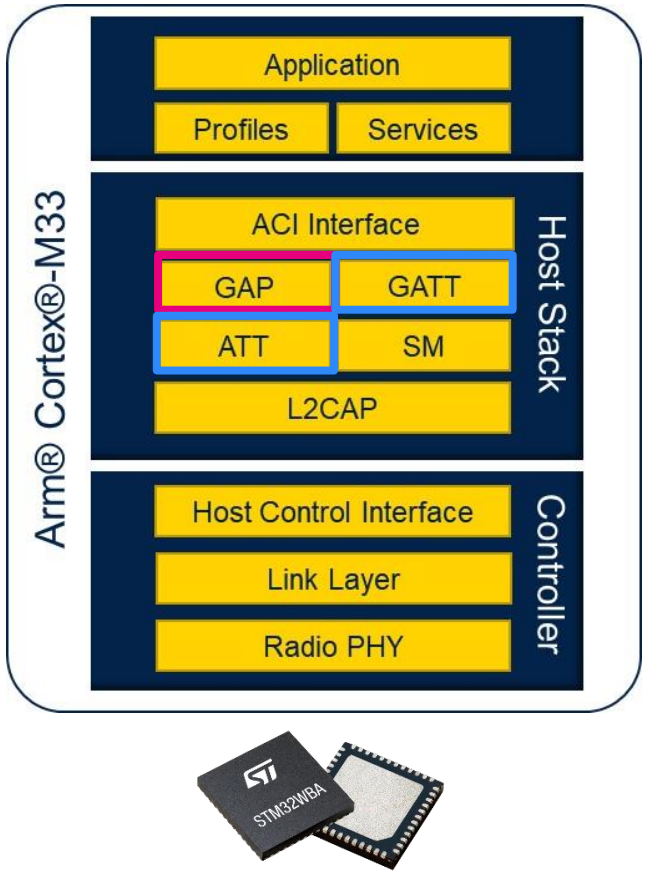
Step2 : Advertising and BLE GAP/GATT custom application configuration

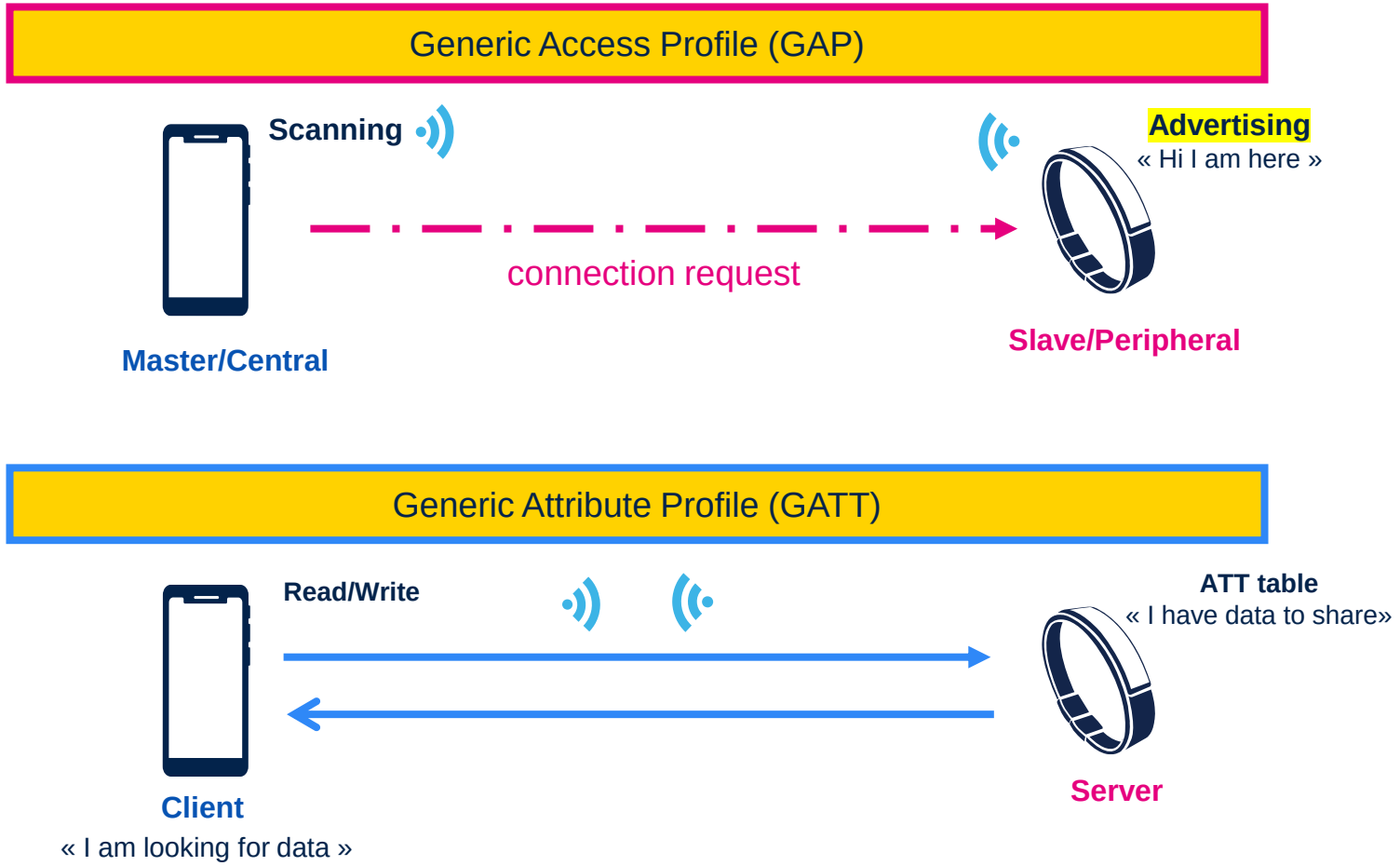
life.augmented

Enabling Bluetooth Low Energy



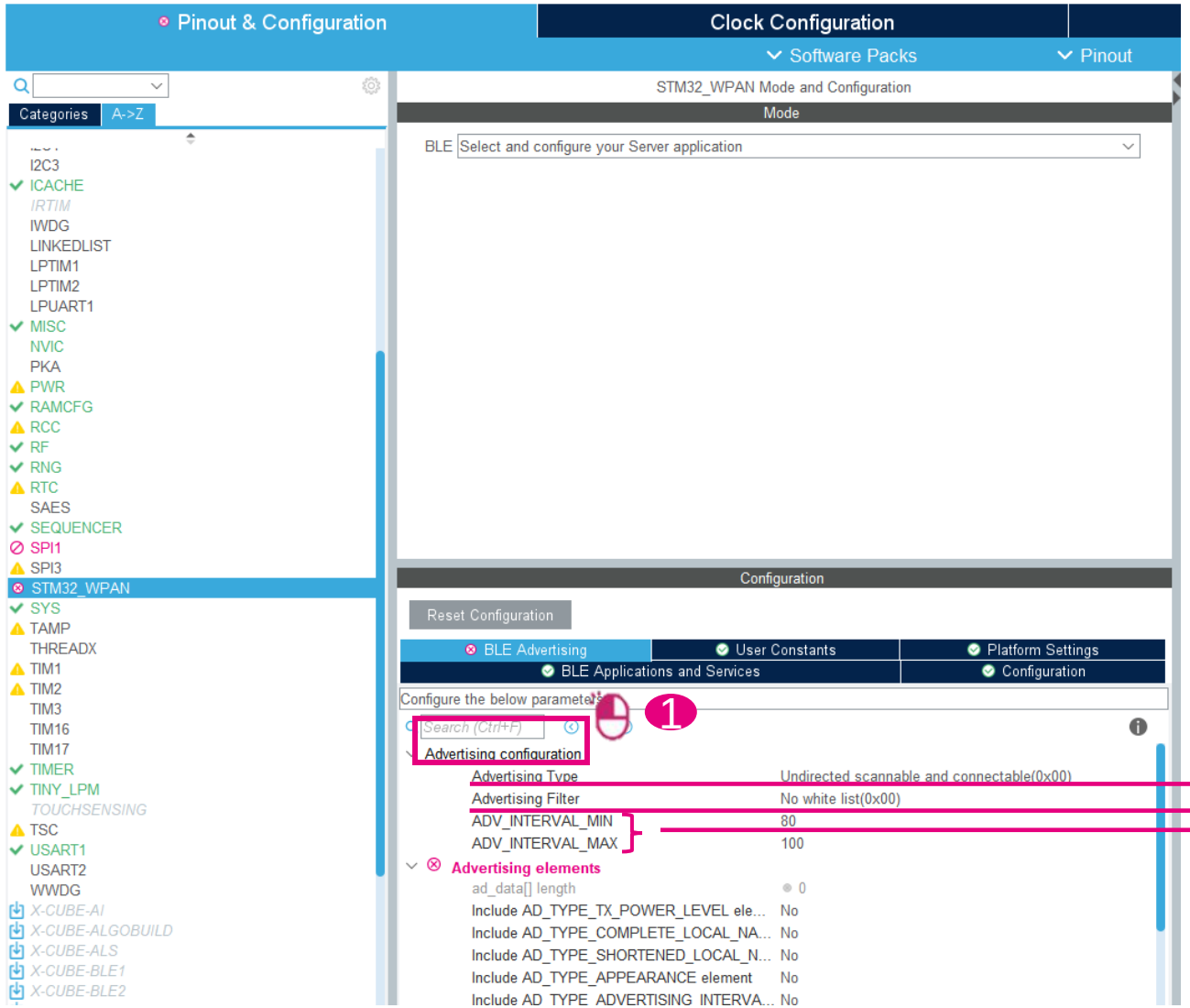
Bluetooth Low Energy Connection roles vs. Data roles





In the general run of things….
a Central is acting as GATT Client, a peripheral as a GATT server

Advertising Configuration

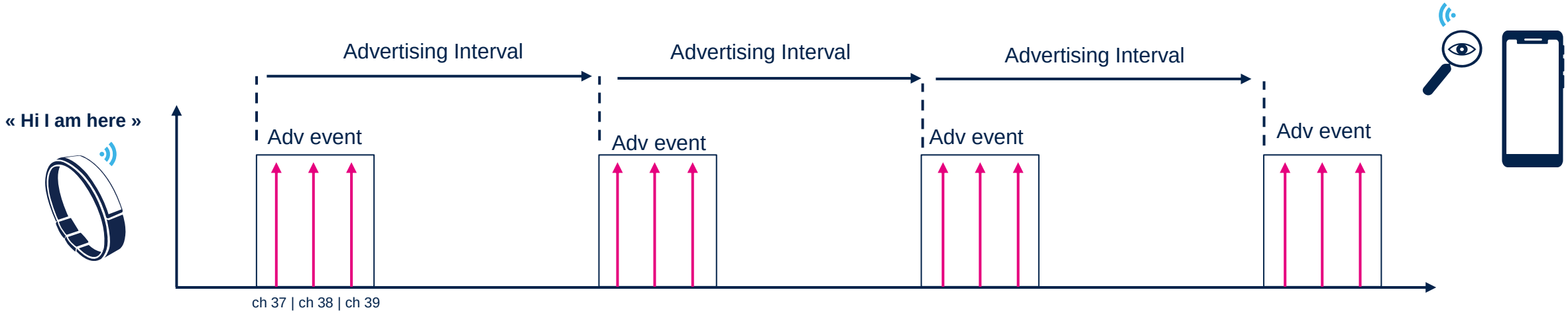


Advertising Type
accept connection requests from any peer device

Advertising Filter
In general used in case of Privacy.

Advertising Interval
Advertising set = (MIN+MAX)/2
Min & Max used in case of multi connections

Advertising Configuration
Advertising Interval

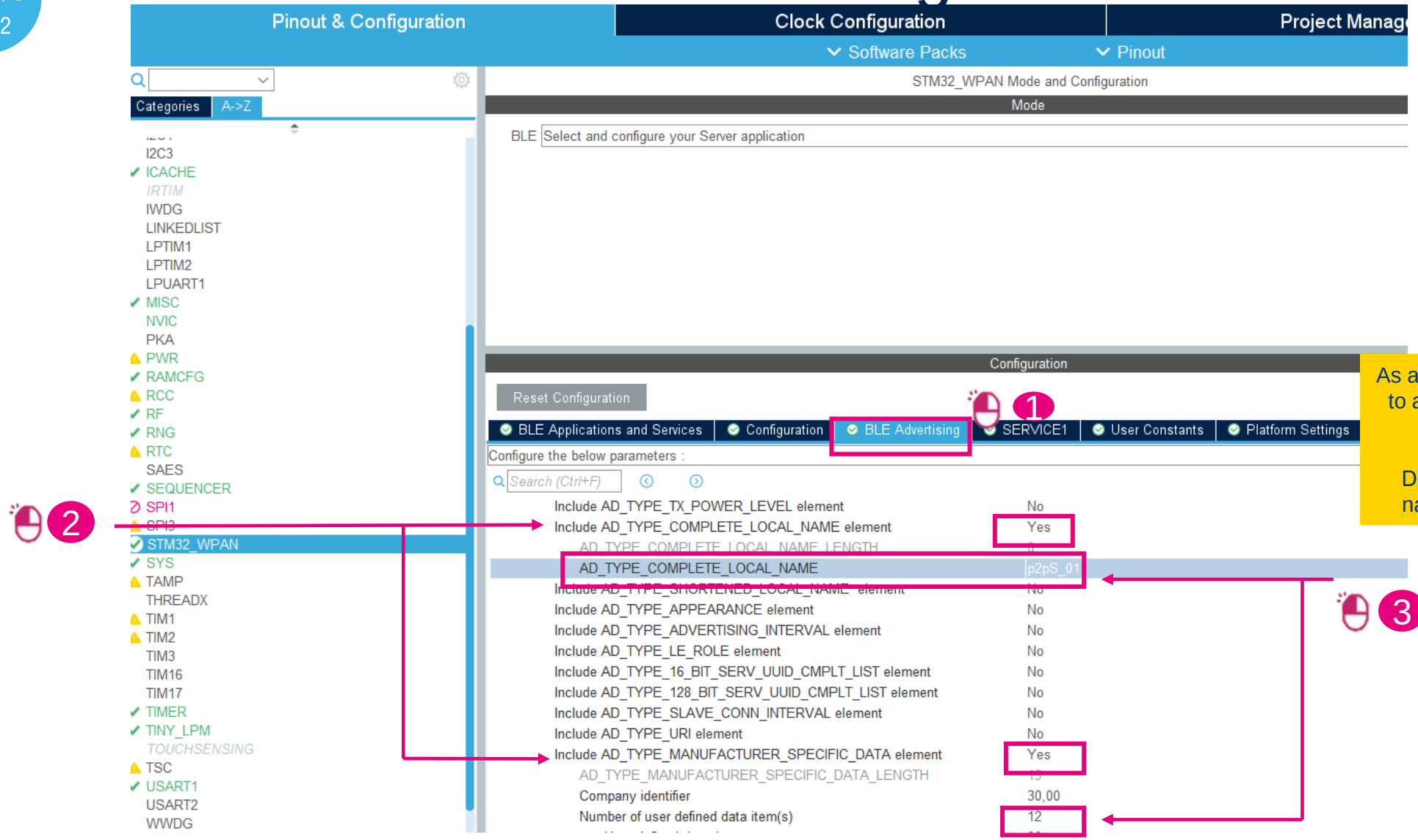


- The advertising interval value ranges all the way from **20** milliseconds up to **10.24** seconds in small increments of **625** microseconds.
- The advertising interval greatly impacts battery life and should be chosen carefully.

connectivity latency vs. power consumption efficiency

- The advertising event is the slot where peripheral will be able to push for advertising data “Hello I am here – this is my name”
- The advertising event is around **~3ms** considering legacy advertising (31 bytes)

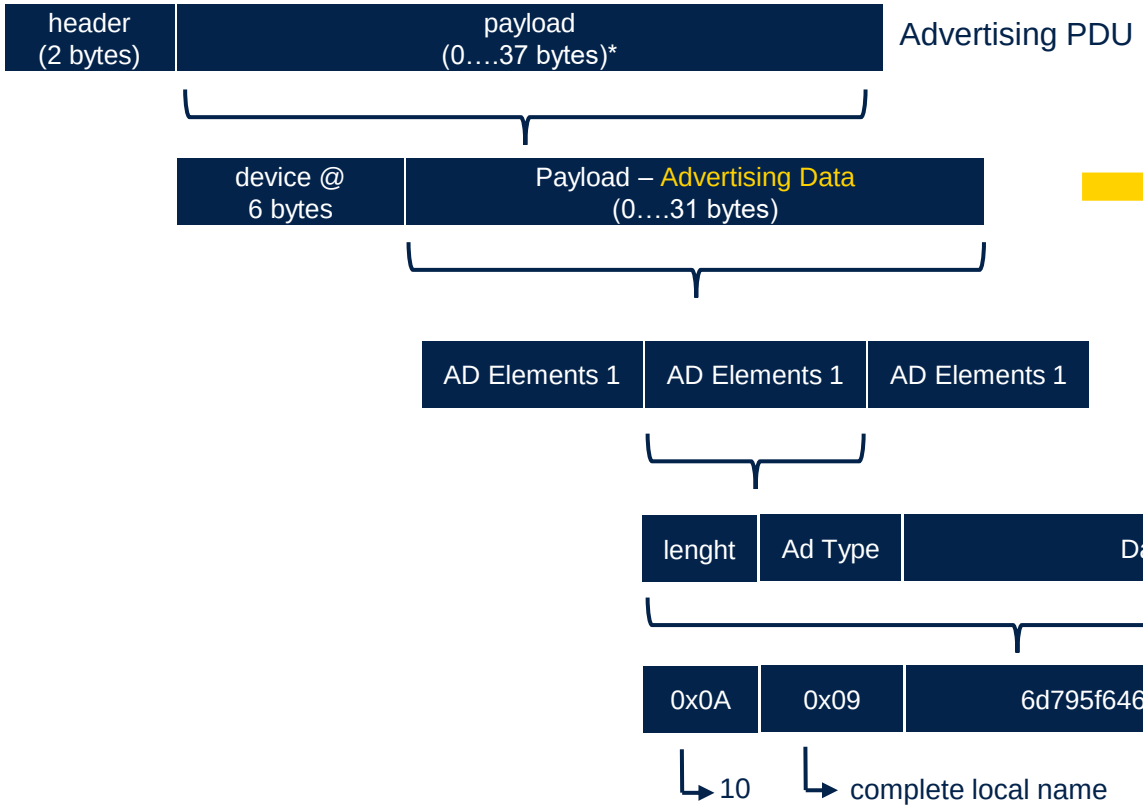
Advertising Elements Local Name



As a server, our application will have to advertise waiting for connection request from a client.

Define here your “custom” local name part of advertising frame.

Advertising Elements
Advertising PDU



The Advertising Data consists of one or more Advertising Data elements
AD Element/Type are listed at Bluetooth SIG website

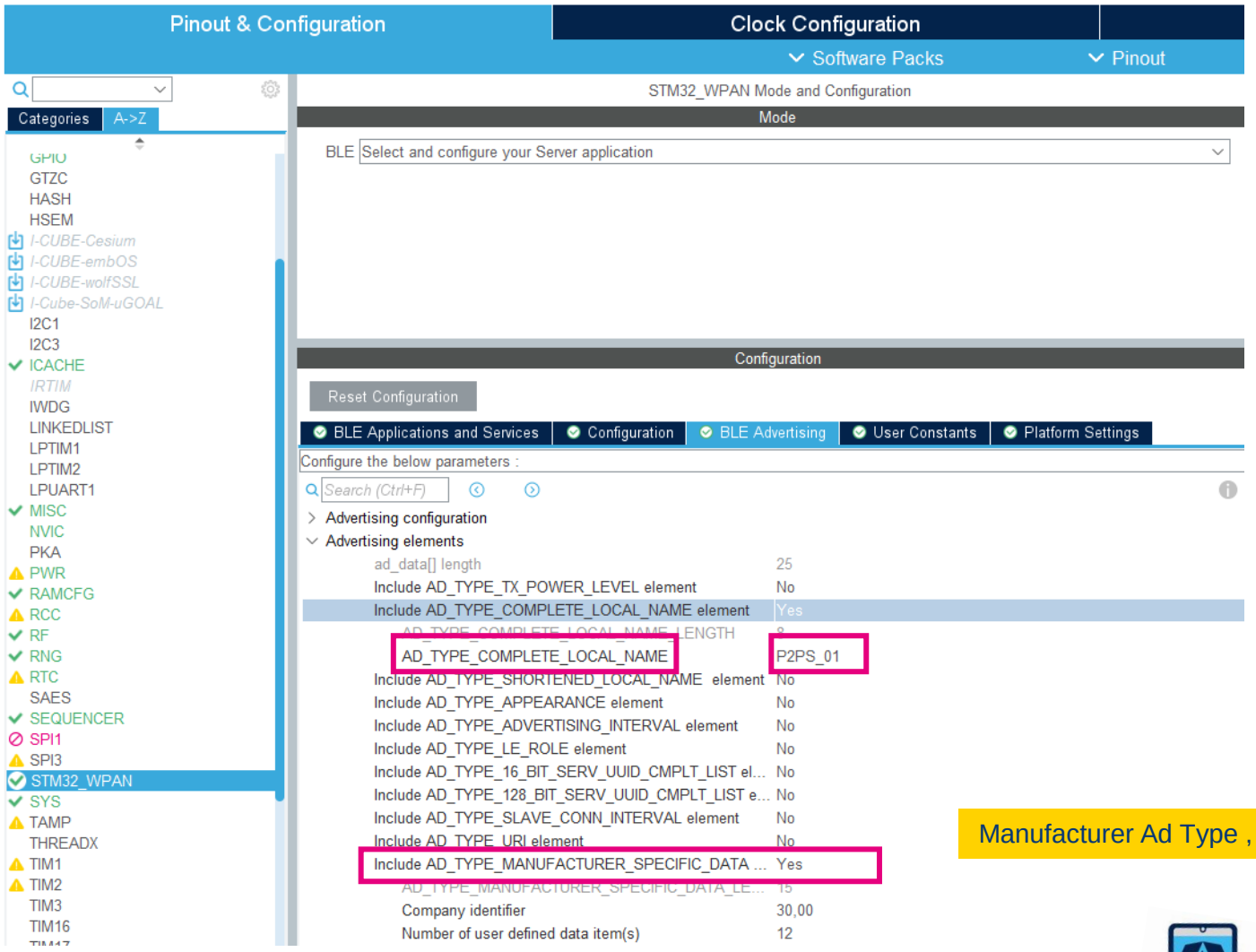
Most commonly used AD elements :
- 0x01 = Flags (**mandatory for connectable device**)
- 0x09 = Complete Local Name
- 0xFF = Manufacturer Data

added by aci_gap_set_discoverable command

You can push for what you want over the air ! All data need to be prefix using dedicated Ad Type

* up to 37 bytes in legacy advertising , up to 1650 in case of advertising extension (packet of 250 chained)

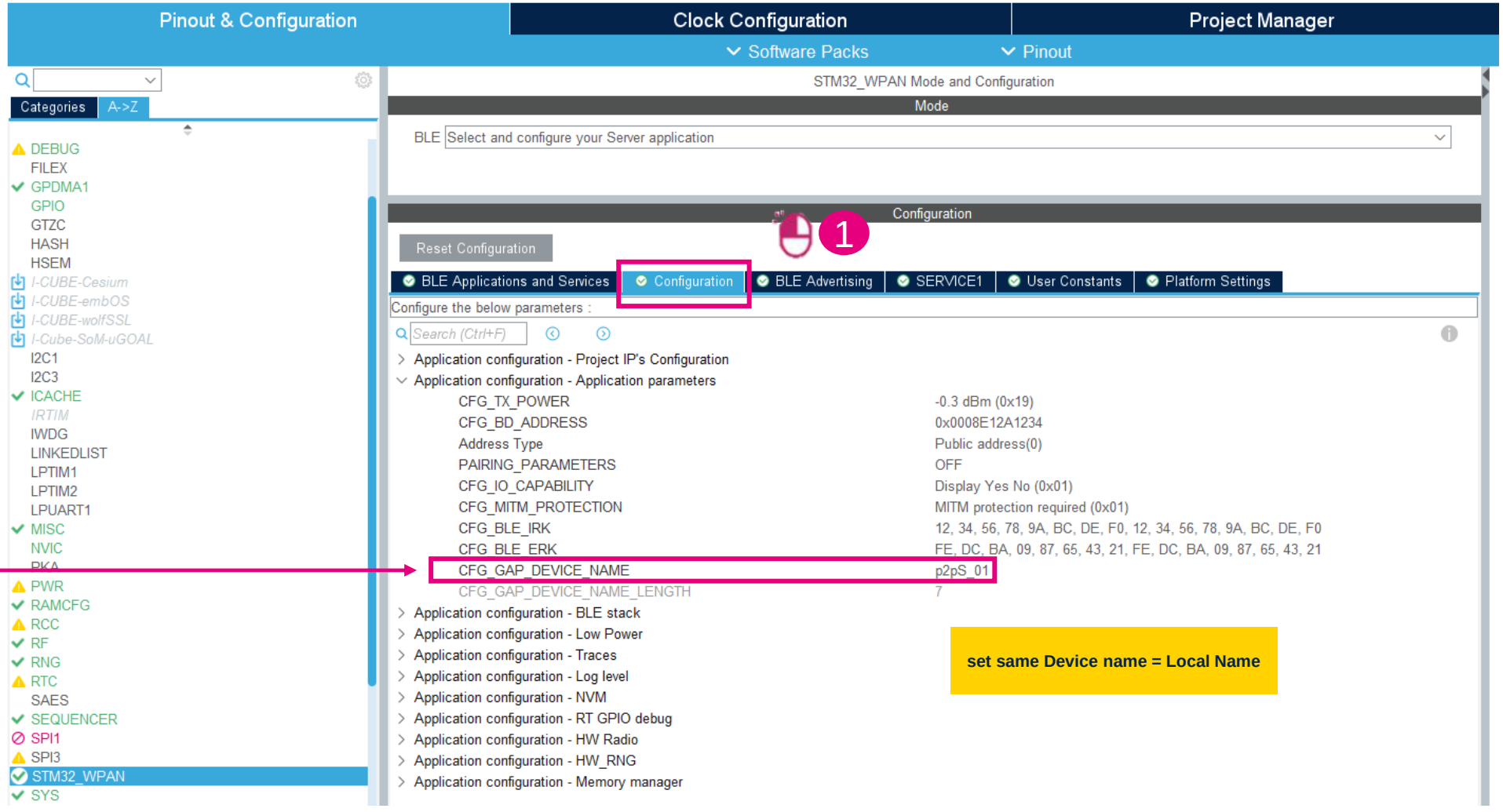
Advertising Elements Manufacturer Data



Manufacturer Ad Type , with company ID 0x30 (STMicroelectronics)

Allow to detect device as ST device and to connect as P2P profile

Customize Device Name

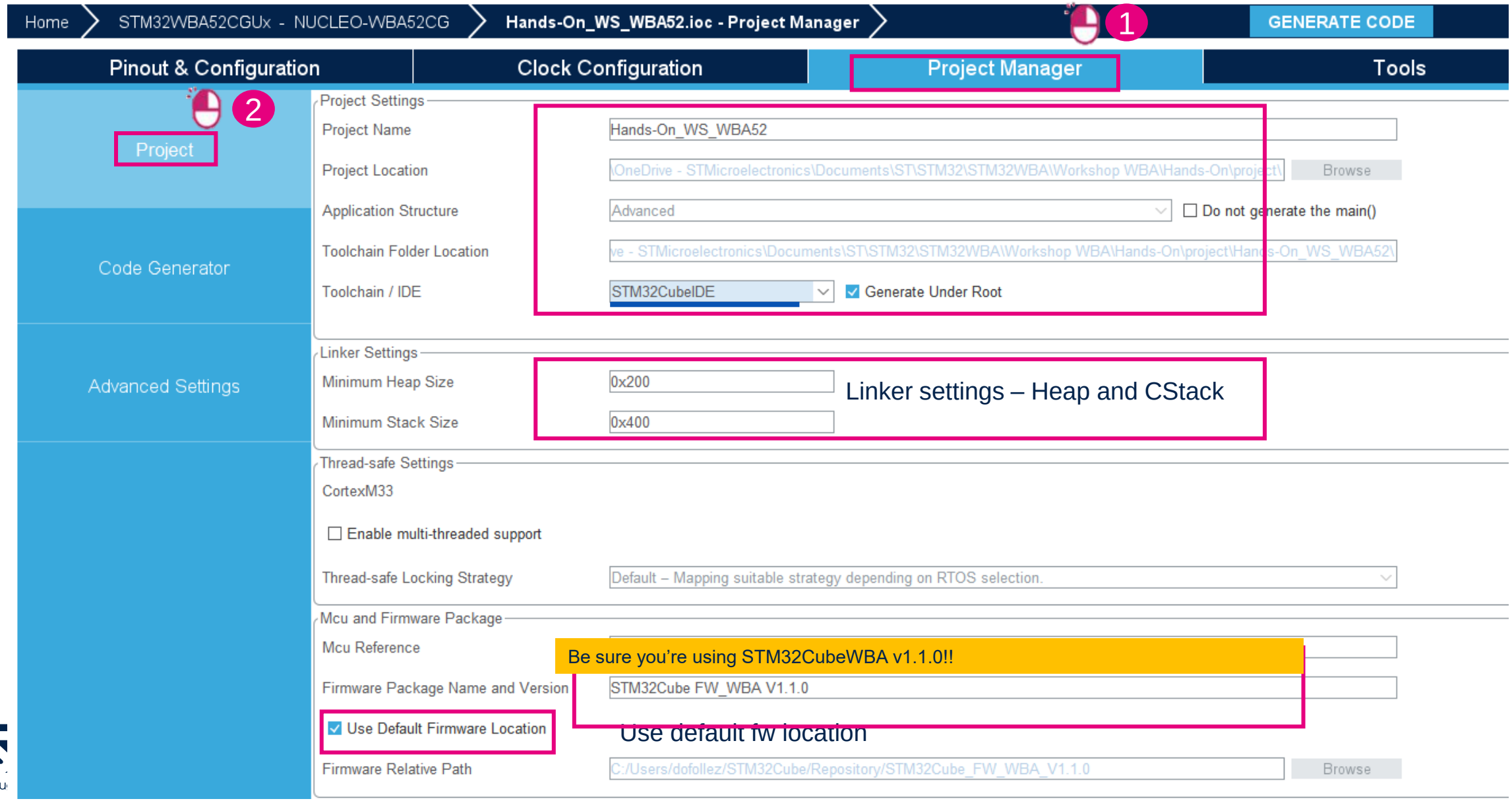


iOS displays Local Name (advertising data) prior to a 1st connexion.
After a 1st connexion iOS displays Device name (thanks to look up table : associates BLE MAC @ & Device Name)

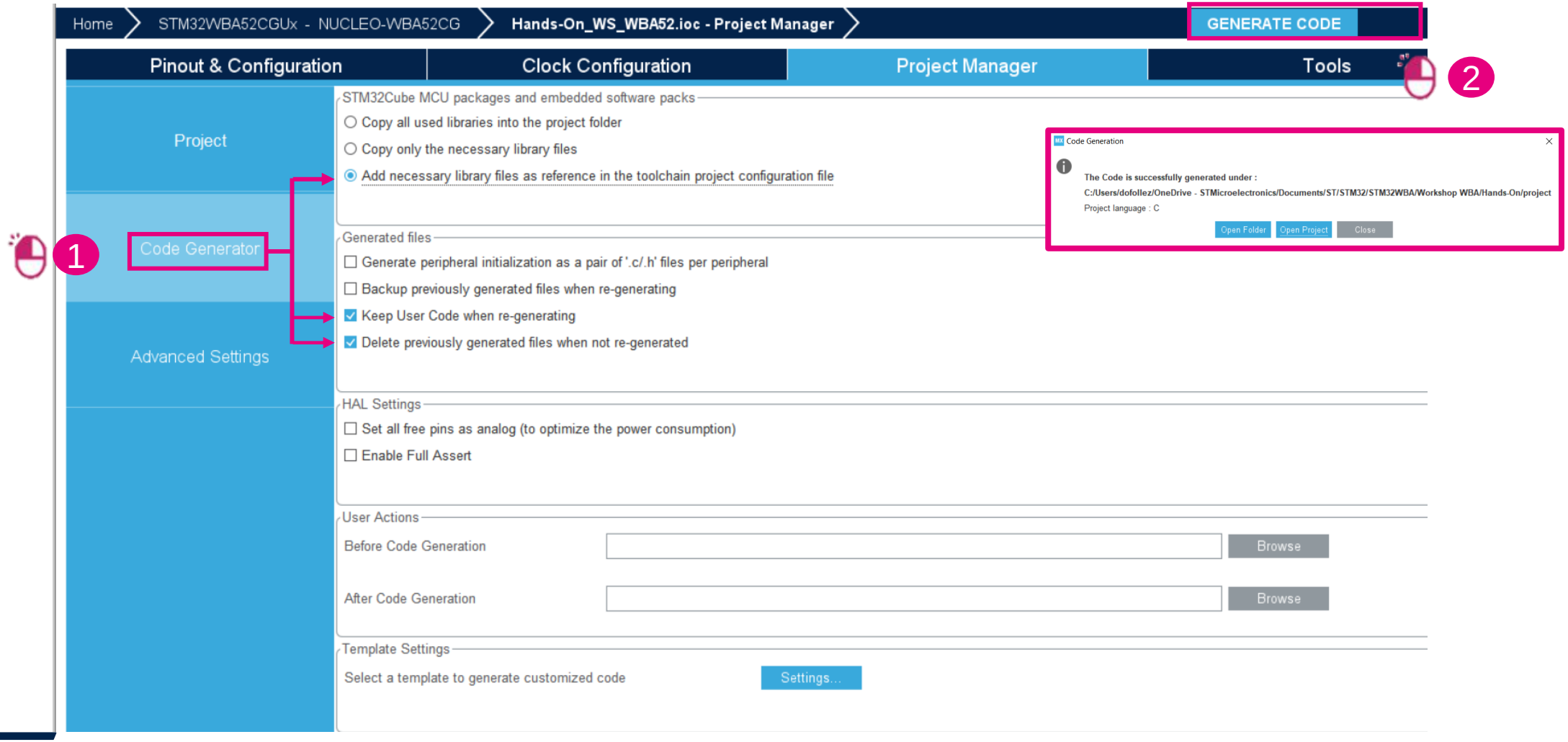
Step 3 : Code generation and user application code

life.augmented

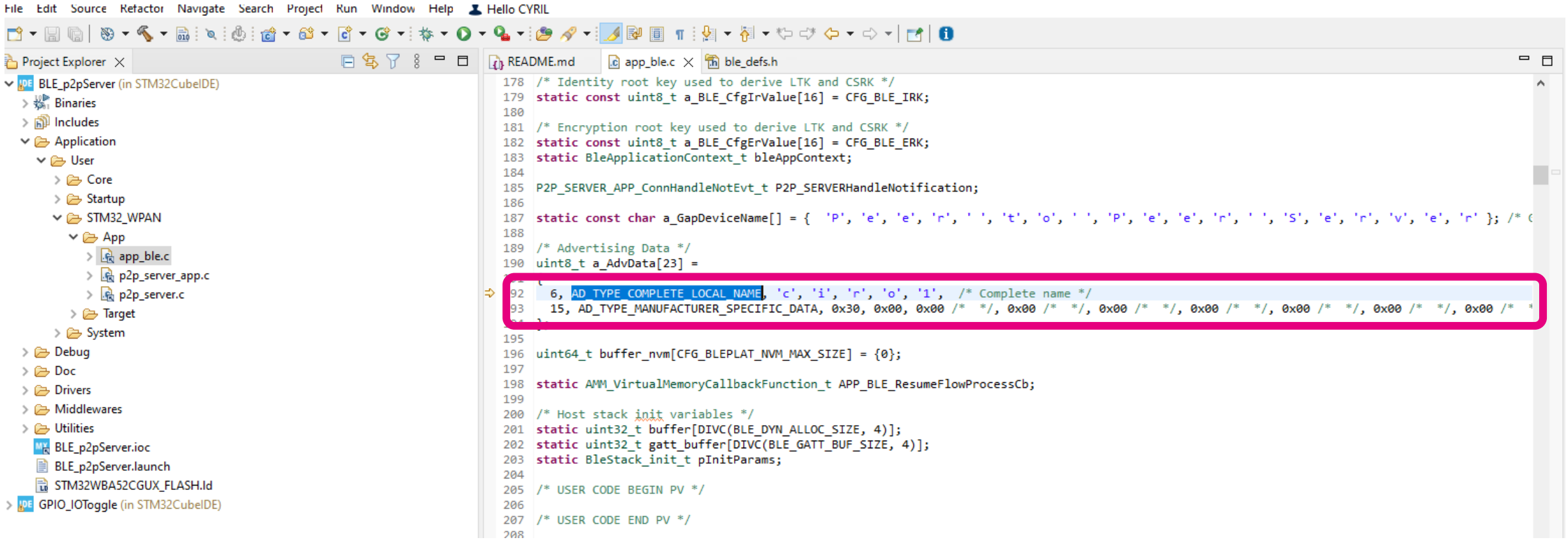
Project configuration



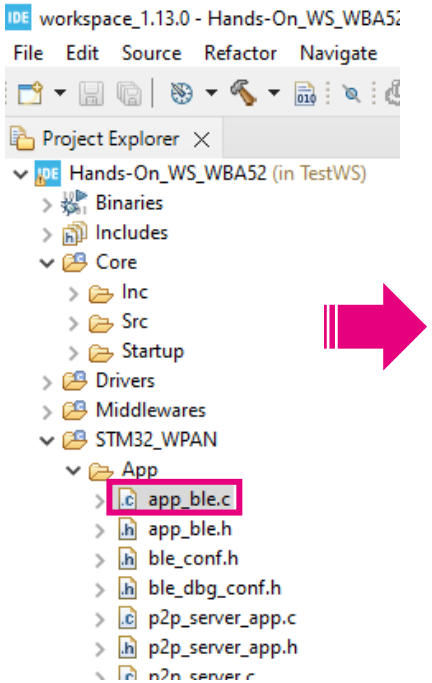
Project configuration



Here are our ADV data



Open Project
Add application code to move to discoverable



#1
: Set device discoverable at init :
In app_ble.c > function APP_BLE_Init()

(ADV_MIN+ADV_MAX)/2

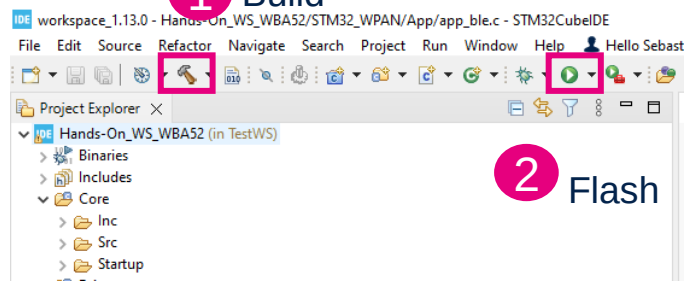
```
/* USER CODE BEGIN APP_BLE_Init_2 */
APP_BLE_Procedure_Gap_Peripheral(PROC_GAP_PERIPH_ADVERTISE_START_FAST);
/* USER CODE END APP_BLE_Init_2 */
```

#2 : Set device discoverable at disconnection :
In app_ble.c > SVCCTL_App_Notification - HCI_DISCONNECTION_COMPLETE_EVT_CODE

```
/* USER CODE BEGIN EVT_DISCONN_COMPLETE */
APP_BLE_Procedure_Gap_Peripheral(PROC_GAP_PERIPH_ADVERTISE_START_FAST);
/* USER CODE BEGIN EVT_DISCONN_COMPLETE */
```

At disconnection, stack is not moving back to advertising, this is an application decision

1 Build



2 Flash

Open Project > Add application code to move to discoverable > Build& Flash

life.augmented

Open your App and Connect

Download on the App Store

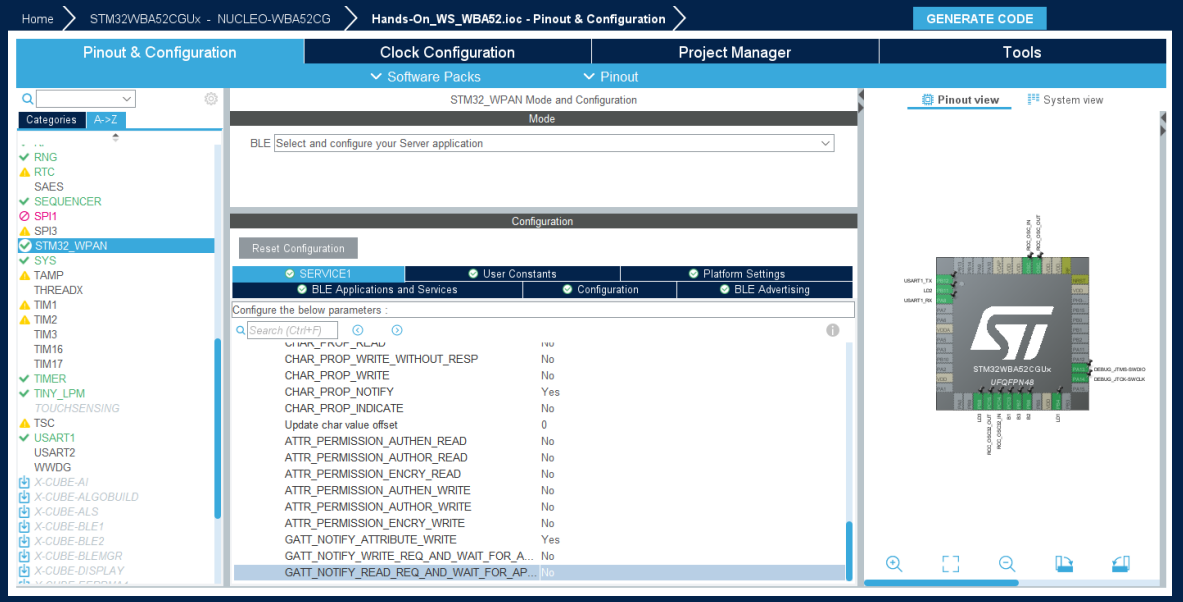
GET IT ON Google Play

Device should be visible and connectable

life.augmented

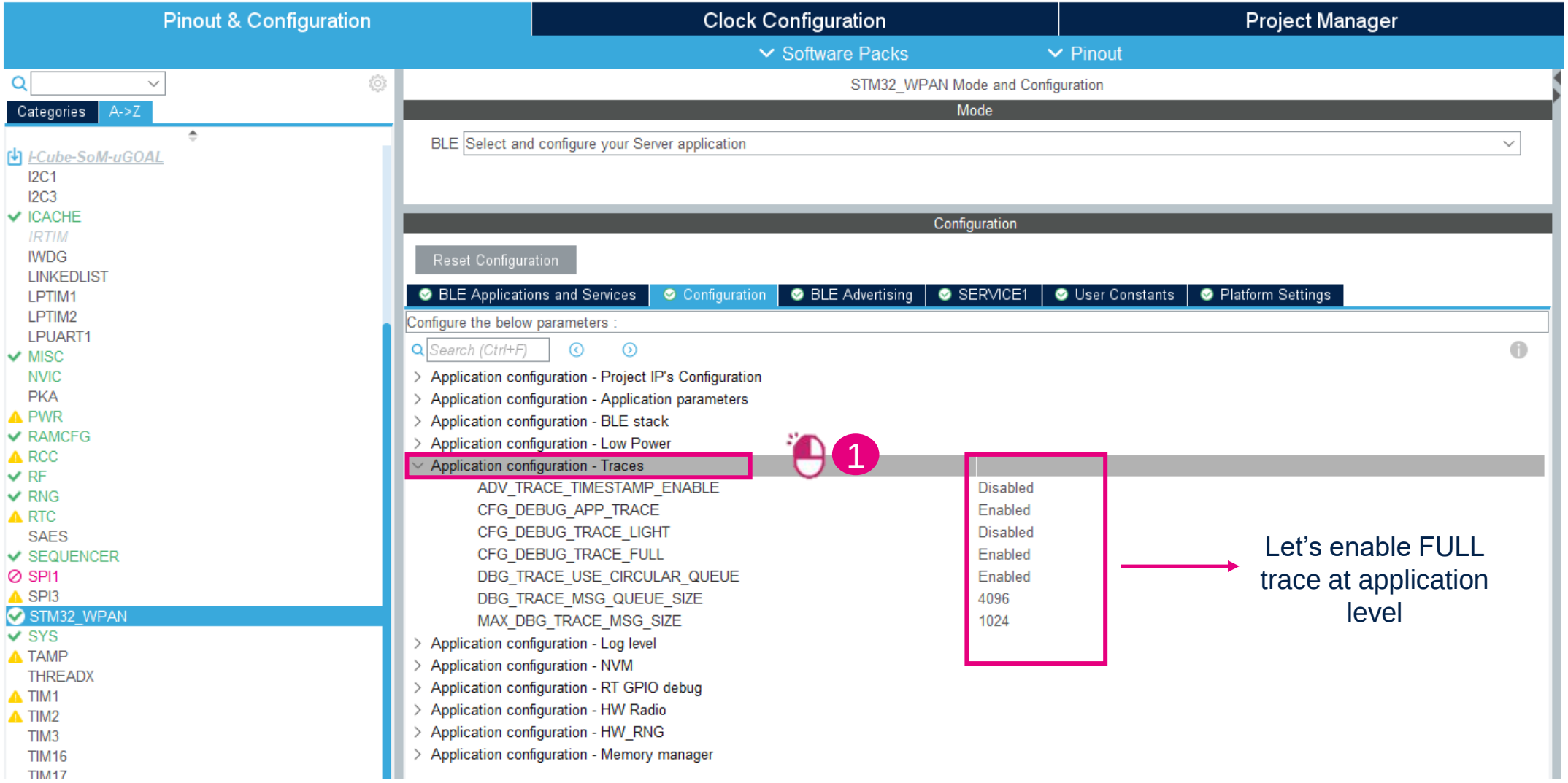
Add debug capabilities

Move back to CubeMx

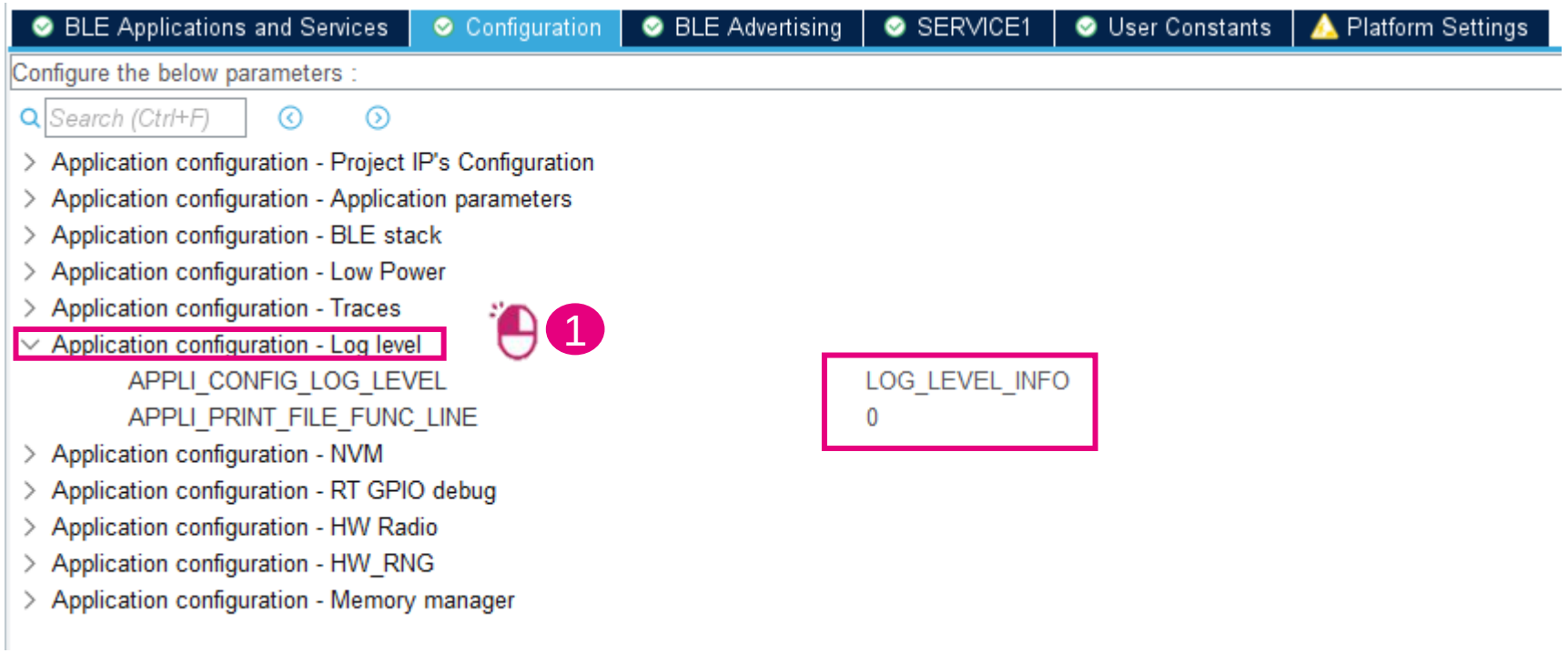


life.augmented

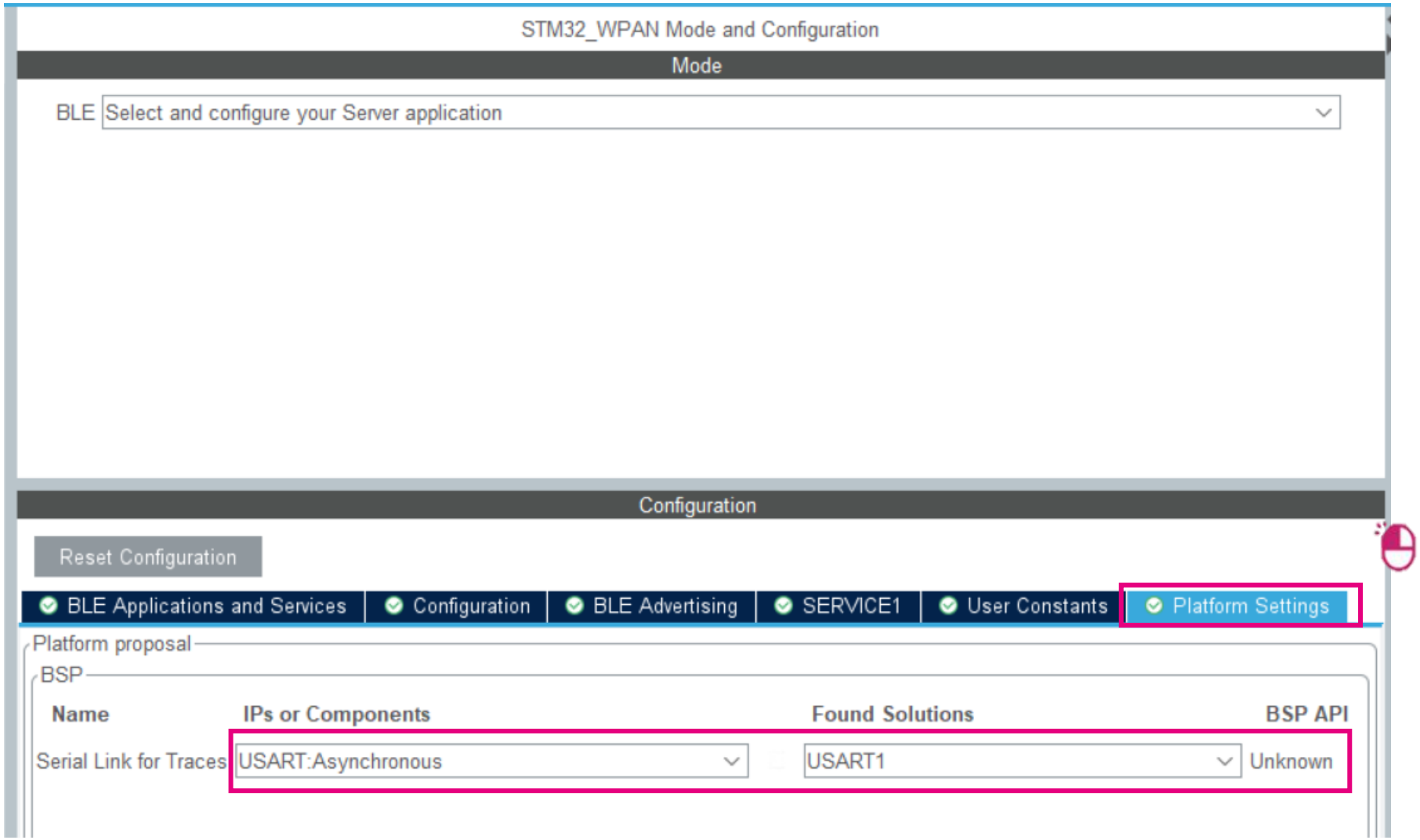
Application configuration
Traces & logs



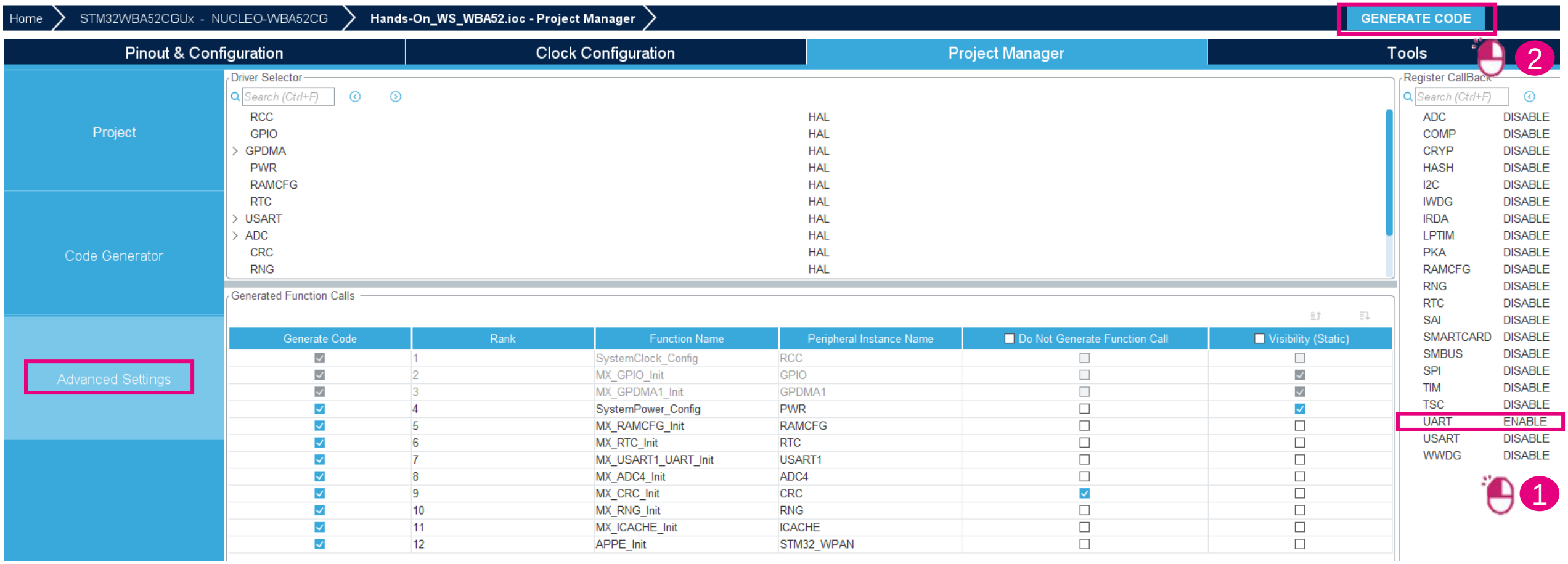
Application configuration
Trace & Logs: configure log level



Platform Settings
Trace & Logs: BSP settings



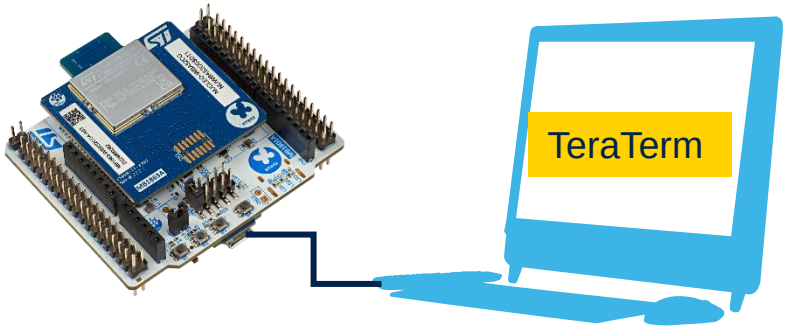
Project configuration
Advanced settings

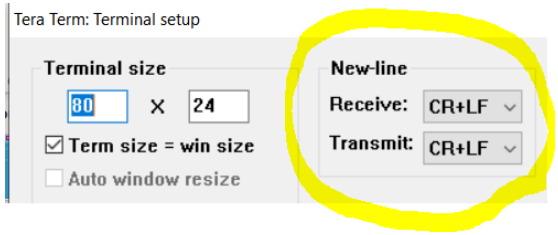


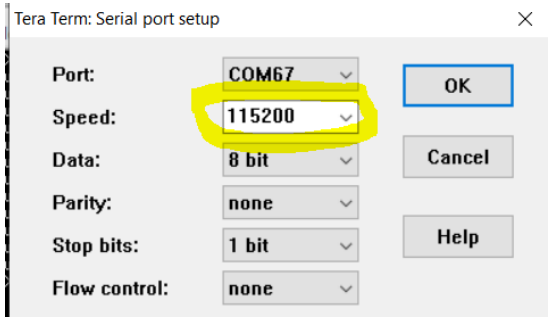
Regenerate Code
Open back existing project – refresh sources – build and flash

life.augmented

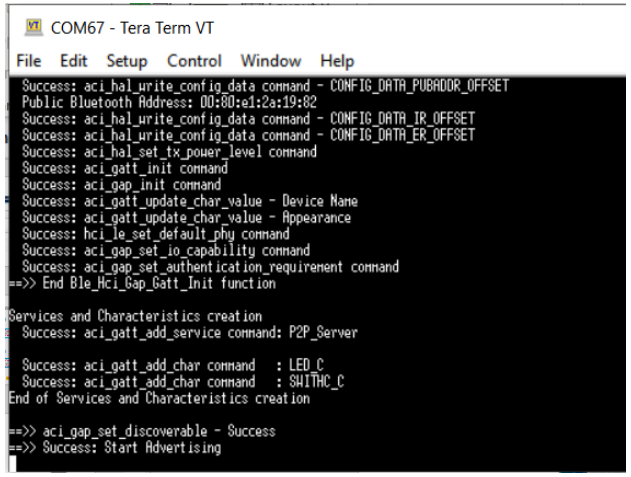
Open your App and Connect



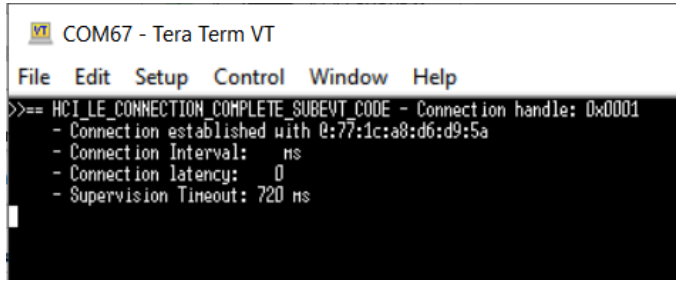




1 reset device



2 Connect



Hands On #2 : BLE Profiles

life.augmented

Configuration completed
What's next : code generation ?

HW configuration

BLE parameters configuration for advertising

Our Cube MX already includes profile configuration. In this second step we want to focus on what a BLE profile is

Profile Demystification

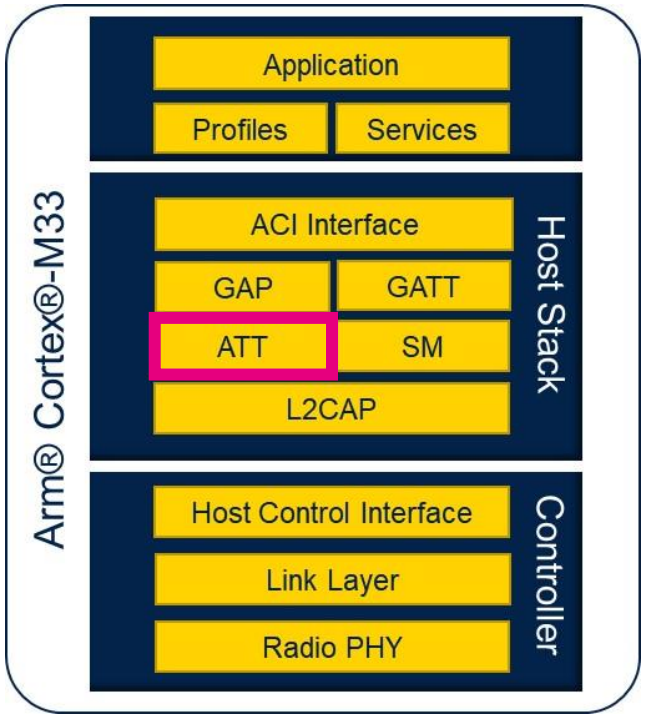
Code generation

Application code

Profile Theory

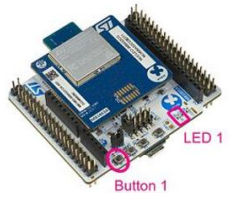
life.augmented

What is a Bluetooth Low Energy Profile
Attribute Protocol (ATT)



Define Client - Server architecture

« I am looking for data »
Client

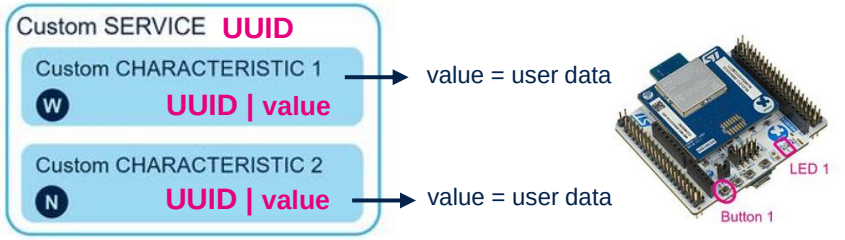


« I have data to share» (sensors, raw data...**what you want** !)
Server

Logical data structure – **« How to access data »** : Attribute table

user data is accessible trough attribute

service → characteristic → data



life.augmented

What is a Bluetooth Low Energy Profile

A profile is a collection of data (**attributes**) exposes by device trough associated Service and Characteristic

- All attributes have a type which is identified by a UUID (**U**niversally **U**nique **Id**entifier)
- Characteristic can take 3 types of propreties: **R**EAD, **W**RITE, **N**OTIFY
- Profile can be defined by **Bluetooth® SIG**
  - **UUID : 16 bits**
    Service Heart Rate **0x180D**
    Characteristic Heart Rate Measurement **0x2A37**
- Profile can be a **custom** (proprietary) profile
  - **UUID : 128 bits**
    Service P2P **0000FE40**-cc7a-482a-984a-7f2ed5b3e58f
    Characteristic LED **0000FE41**-cc7a-482a-984a-7f2ed5b3e58f

life.augmented

BLE standard profile vs. proprietary profile

Standard Heart Rate Profile

Heart Rate Service 0x180D

- **Heart Rate Measurement Characteristic** 0x2A37
- **Body sensor Location Characteristic** 0x2A38

Define by the **SIG**, define the role, requirements, behavior and the structure of Attribute Table of each entity (central & peripheral)

Any standard smartphone App will be able to communicate

Proprietary ST P2PServer Profile

P2P Service 0x0000FE40CC7A482A984A7F2ED5B3E58F

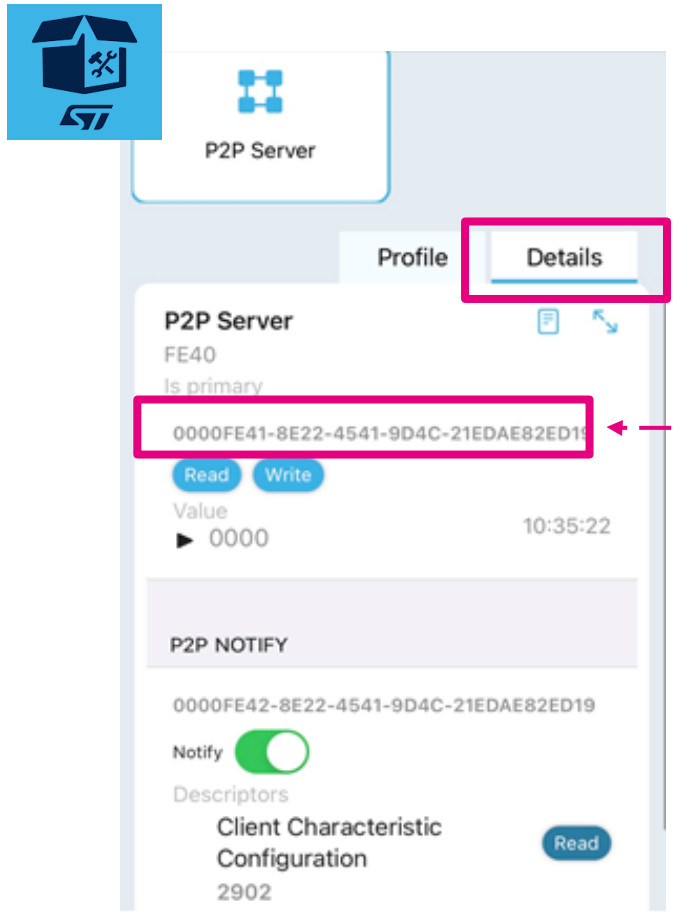
- **My_LED_Char** 0x0000FE41CC7A482A984A7F2ED5B3E58F
- **SWITCH_C** 0x0000FE42CC7A482A984A7F2ED5B3E58F

Define your own behavior using your own Attribute Table based in 128 bits UUID

Only your own App will be able to communicate

life.augmented

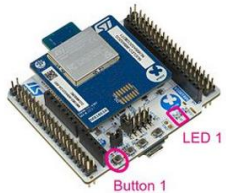
Proprietary profile
ST Toolbox App



ST ToolBox App knows that My_LED_Char proprietary UUID is defined to toggle led .
As consequence App displays nice **toggle button**

life.augmented

Data exchanges
what is the magic behind

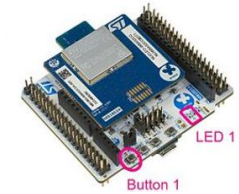


#1 At profile initialization and entry point (**handler @)** will be created in RAM to expose data to client

WBA52 - Attribute Table (RAM)

@ Serv service UUID	**@ Char** W R N char UUID	**@ Char +1** user data

#2 As soon as connected client will discover server attribute table (handler) and will be able to access (**write/read**) data



#3 Application will update data (ie : push button), client will receive **notification** with data updates

@ Serv	**@ Char** W R N	**@ Char +1** update data

BLE write, read, notify procedures using the right attribute handler make data exchange possible

life.augmented

Coming back to our example

ST BLE Toolbox

Download on the App Store

GET IT ON Google Play

Notification - SWITCH

Write - LED

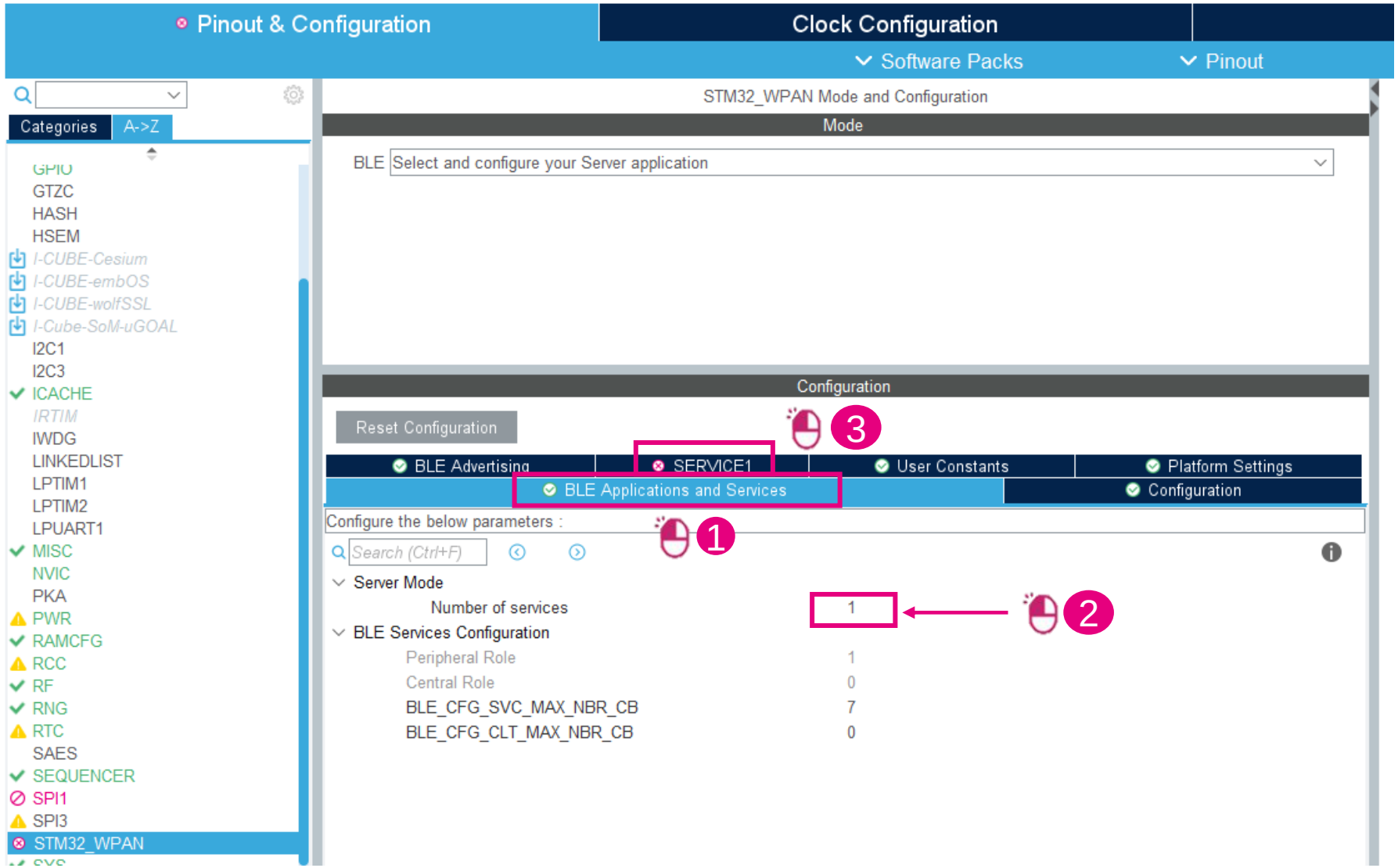
GATT Client **GAP central**

GATT Server **GAP peripheral**

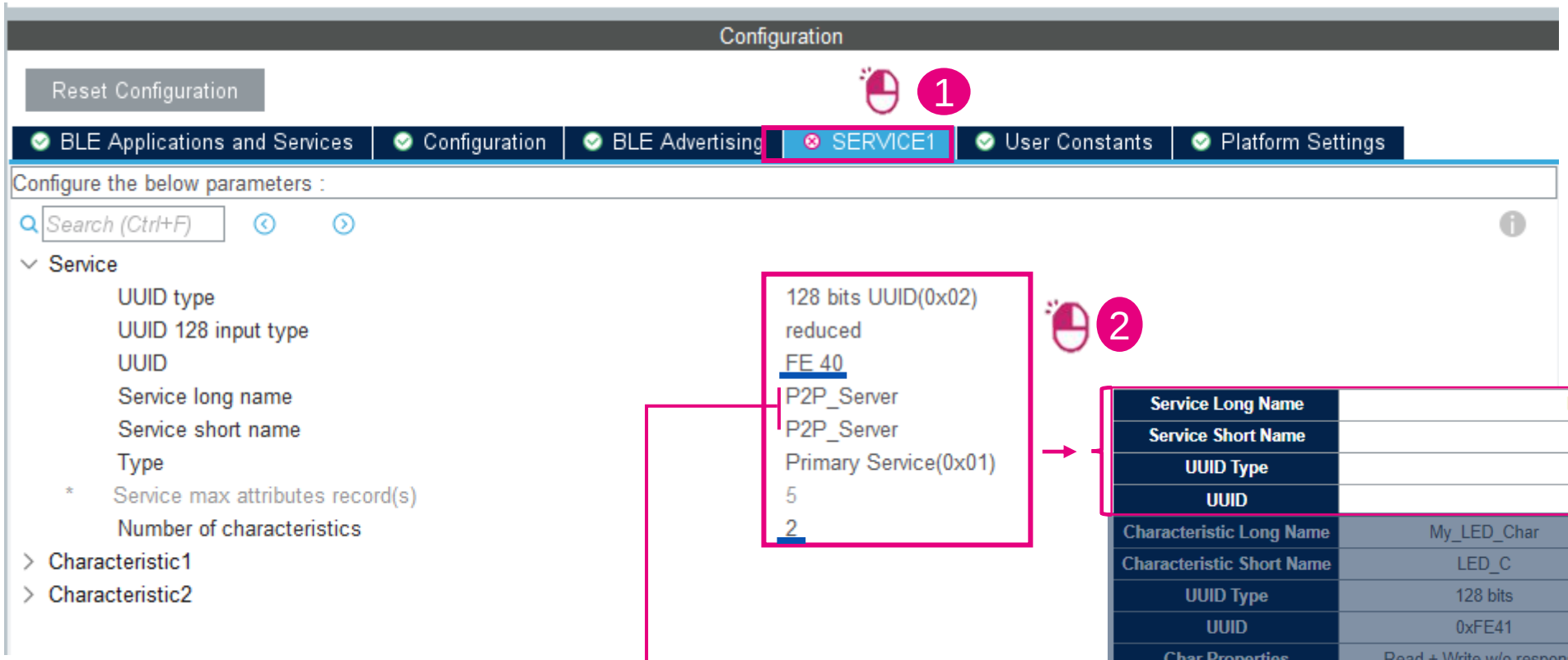
How a profile is done in CubeMX?

life.augmented

Profile Creation Service



Profile Creation
Configure my P2P Service

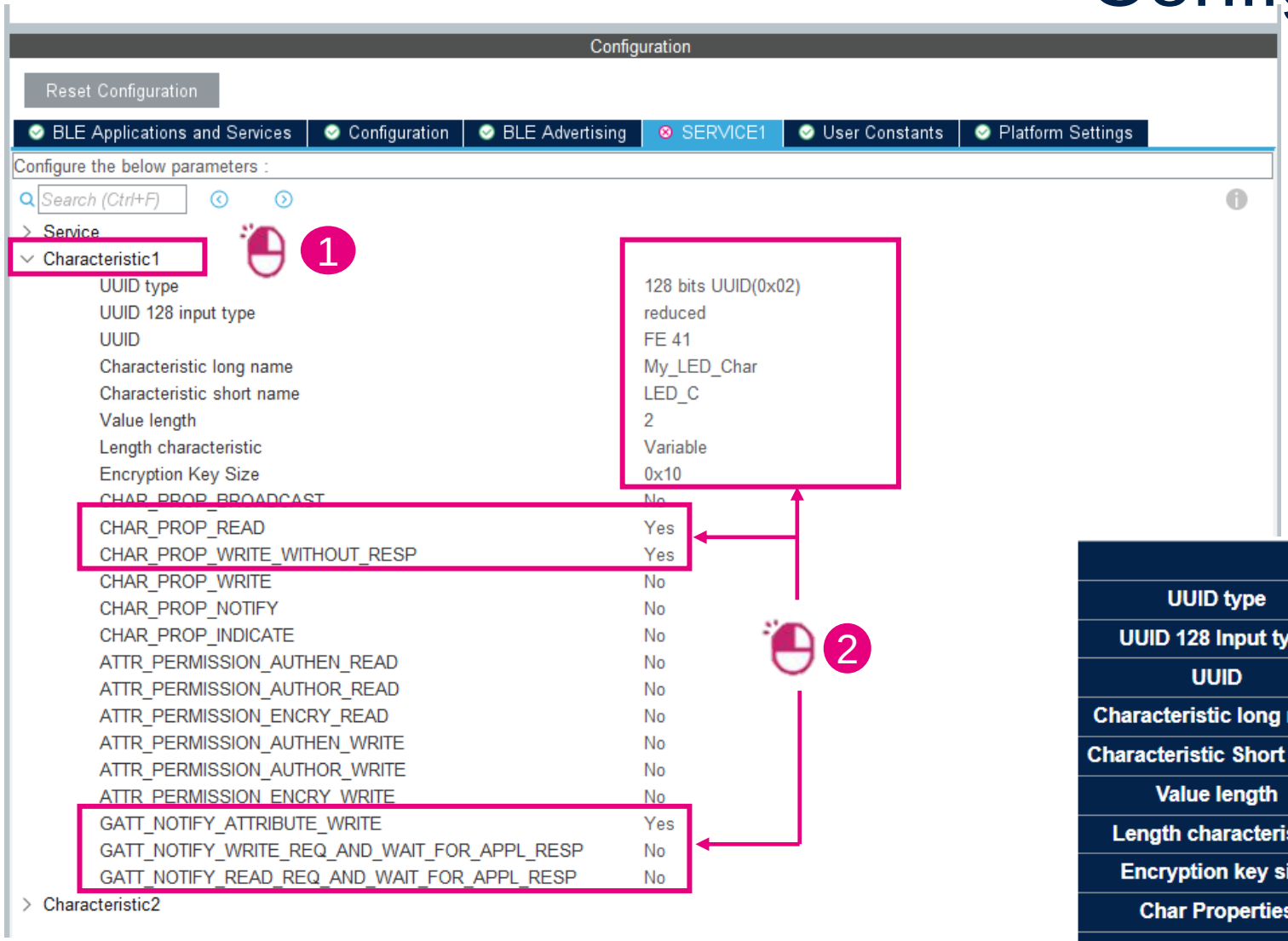


Service Long Name	My_P2P_Server	
Service Short Name	My_P2P	
UUID Type	128 bits	
UUID	0xFE40	
Characteristic Long Name	My_LED_Char	My_Switch_Char
Characteristic Short Name	LED_C	SWITCH_C
UUID Type	128 bits	128 bits
UUID	0xFE41	0xFE42
Char Properties	Read + Write w/o response	Notify
Char Permissions	None	None
Char GATT Events	GATT_NOTIFY_ATTRIBUTE_WRITE	GATT_NOTIFY_ATTRIBUTE_WRITE

service & characteristic naming used to name function at code generation

UUID : FE 40
The application code will append 112 bits (based on UUID generator) to have a complete **128 bits UUID**

Profile Creation
Configure 1st Characteristic



UUID : FE 41
Application code will complete to have a complete **128 bits UUID**

Properties

Data (**2 bytes**) can be read and write.
The purpose of characteristic 1 is to write data in order to control LED

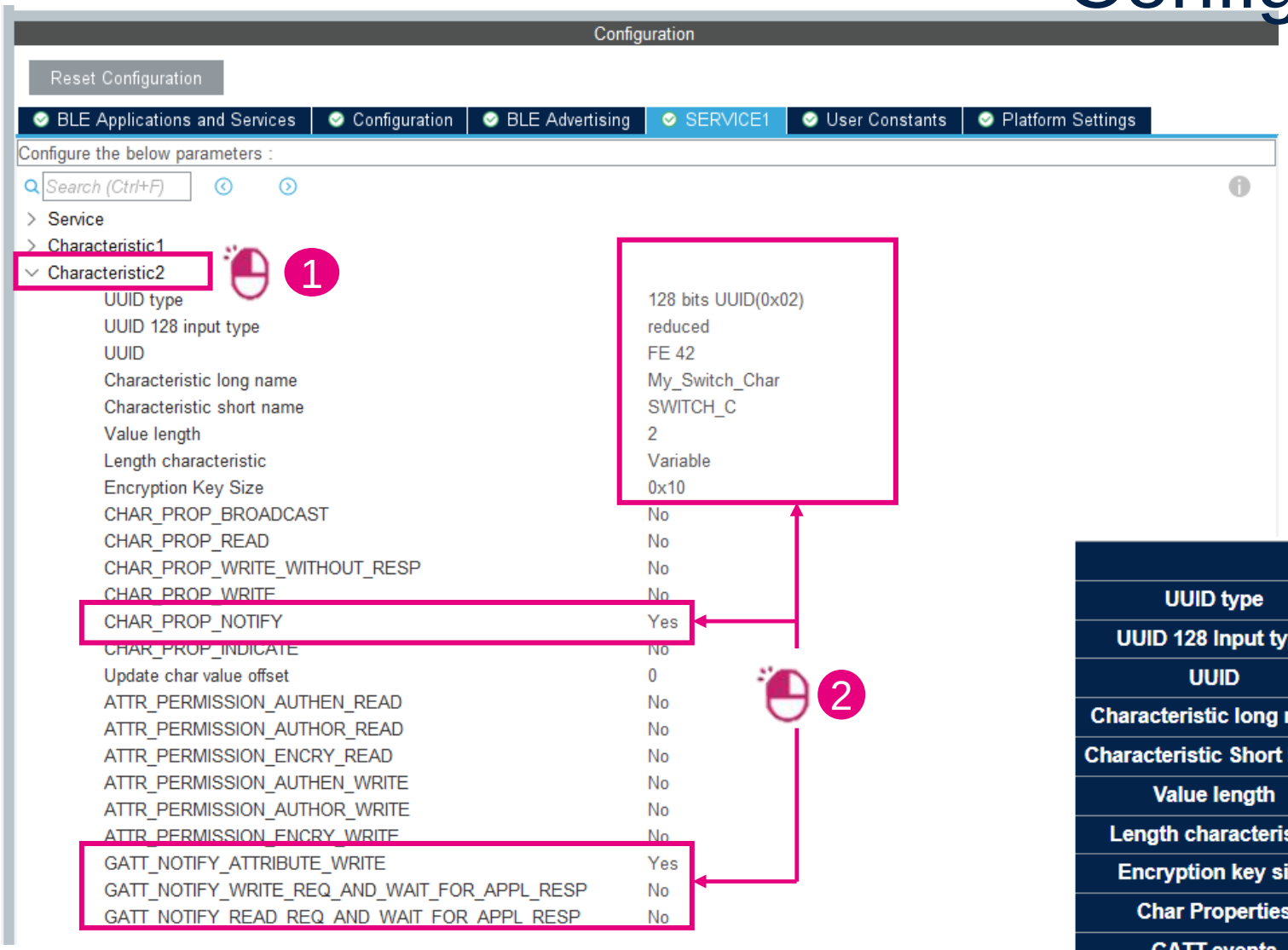
Permission

Thanks to **notify write**, application is informed that attribute has been modified and can accordingly process expected use case

	Characteristic 1	Characteristic 2
UUID type	128 bits UUID (0x02)	128 bits UUID (0x02)
UUID 128 Input type	Reduced	Reduced
UUID	FE 41	FE 42
Characteristic long name	My_LED_Char	My_Switch_Char
Characteristic Short Name	LED_C	SWITCH_C
Value length	2	2
Length characteristic	Variable	Variable
Encryption key size	0x10	0x10
Char Properties	READ | WRITE_WITHOUT_RESP	NOTIFY
GATT events	GATT_NOTIFY_ATTRIBUTE_WRITE	GATT_NOTIFY_ATTRIBUTE_WRITE

life.augmented

Profile Creation
Configure 2nd Characteristic



UUID : FE 42
Application code will complete to have a complete **128 bits UUID**

Properties

Data (**2 bytes**) as a **notify** characteristic
Each time user press button over NUCLEO, information sent to client

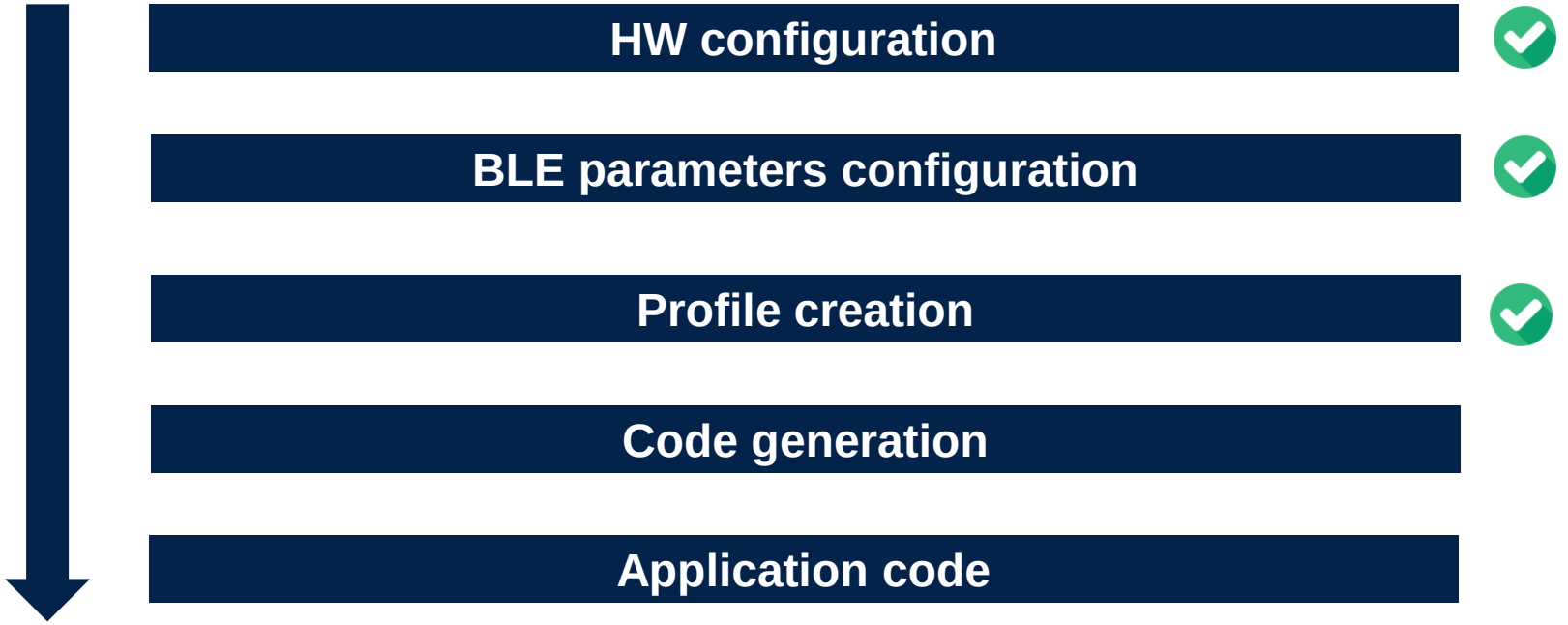
Permission

Here permission has not impact. The server is here sending data to client

	Characteristic 1	Characteristic 2
UUID type	128 bits UUID (0x02)	128 bits UUID (0x02)
UUID 128 Input type	Reduced	Reduced
UUID	FE 41	FE 42
Characteristic long name	My_LED_Char	My_Switch_Char
Characteristic Short Name	LED_C	SWITCH_C
Value length	2	2
Length characteristic	Variable	Variable
Encryption key size	0x10	0x10
Char Properties	READ | WRITE_WITHOUT_RESP	NOTIFY
GATT events	GATT_NOTIFY_ATTRIBUTE_WRITE	GATT_NOTIFY_ATTRIBUTE_WRITE

life.augmented

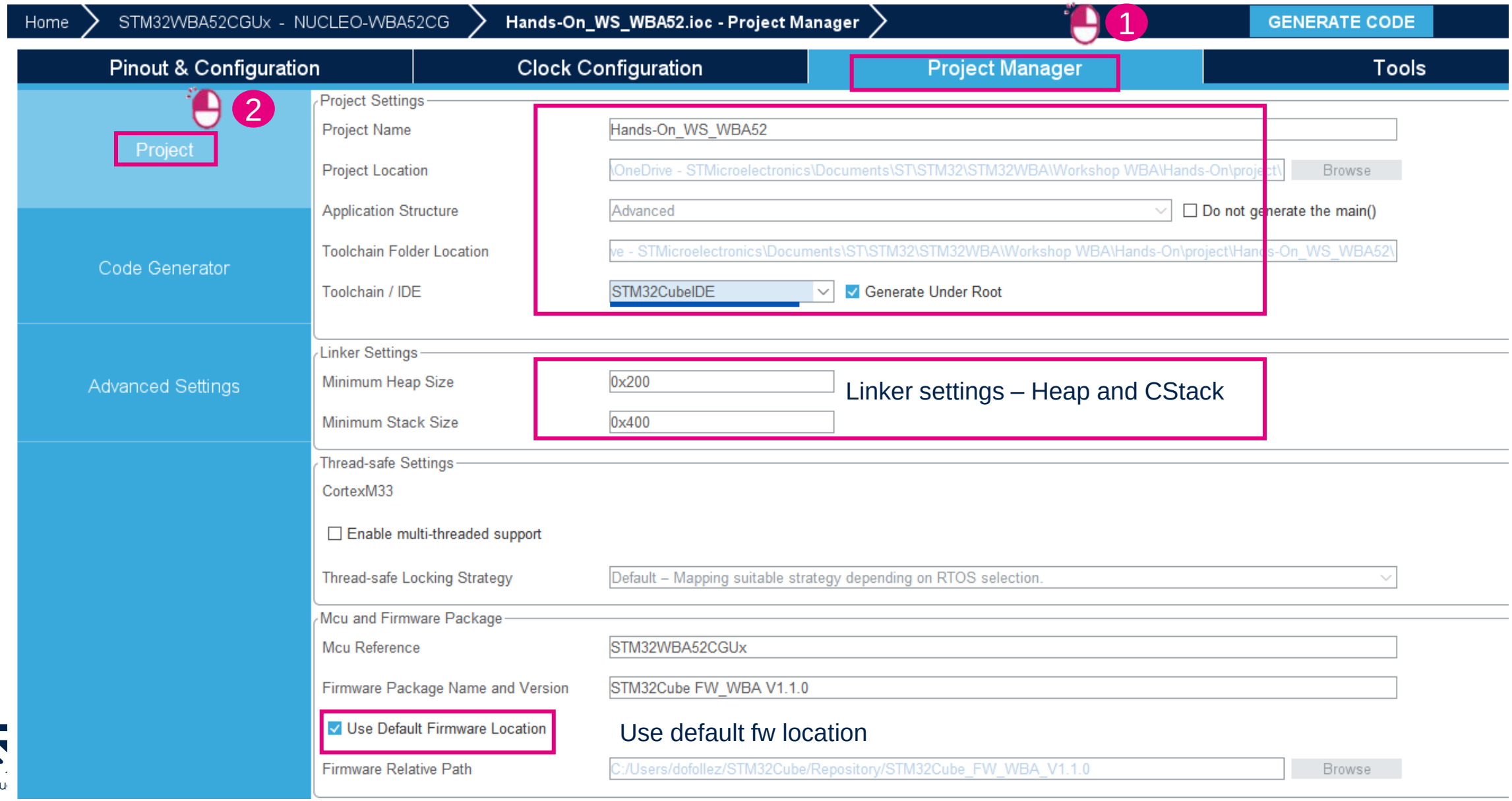
Configuration completed
What's next - Yes code generation



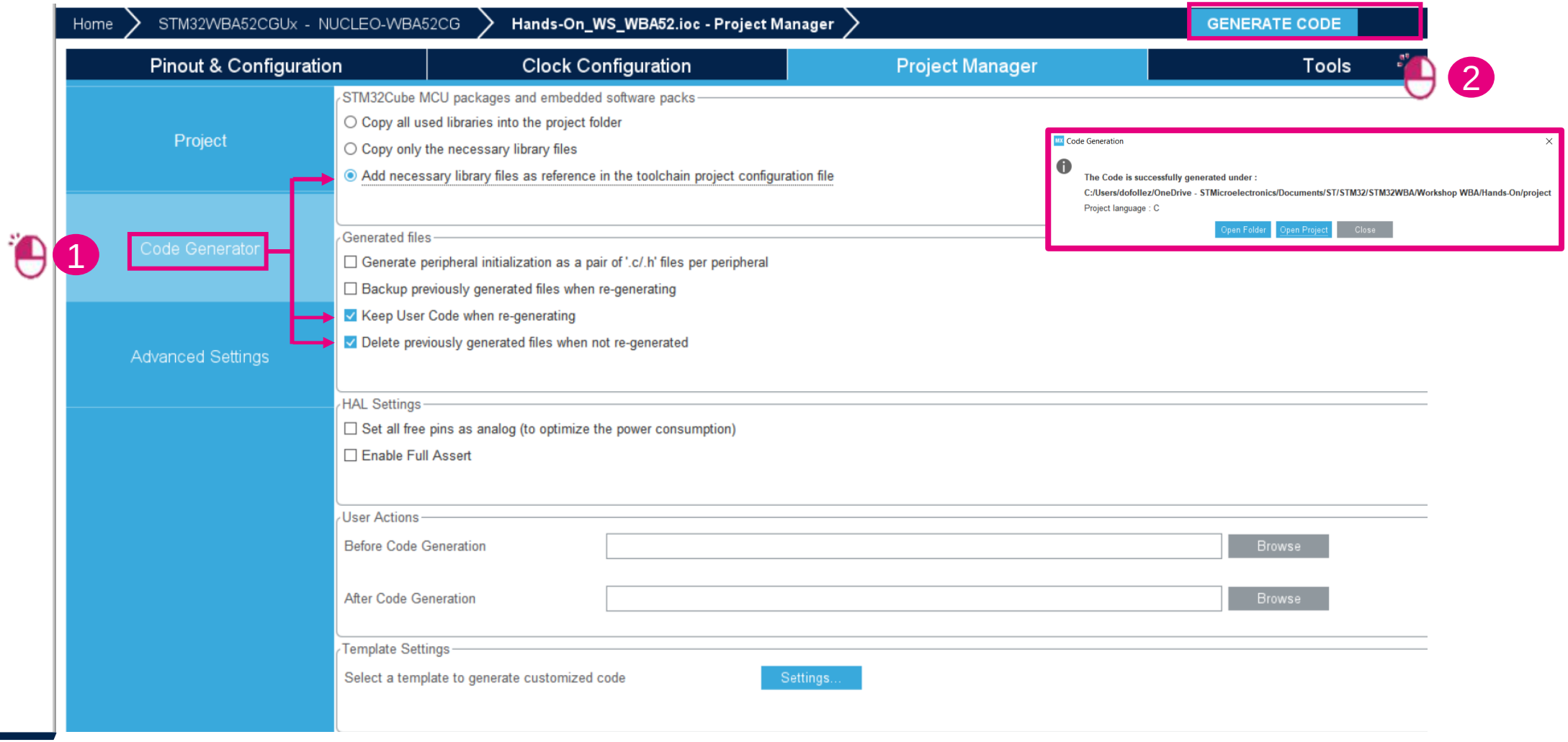
Step 2 : Code generation and user application code

life.augmented

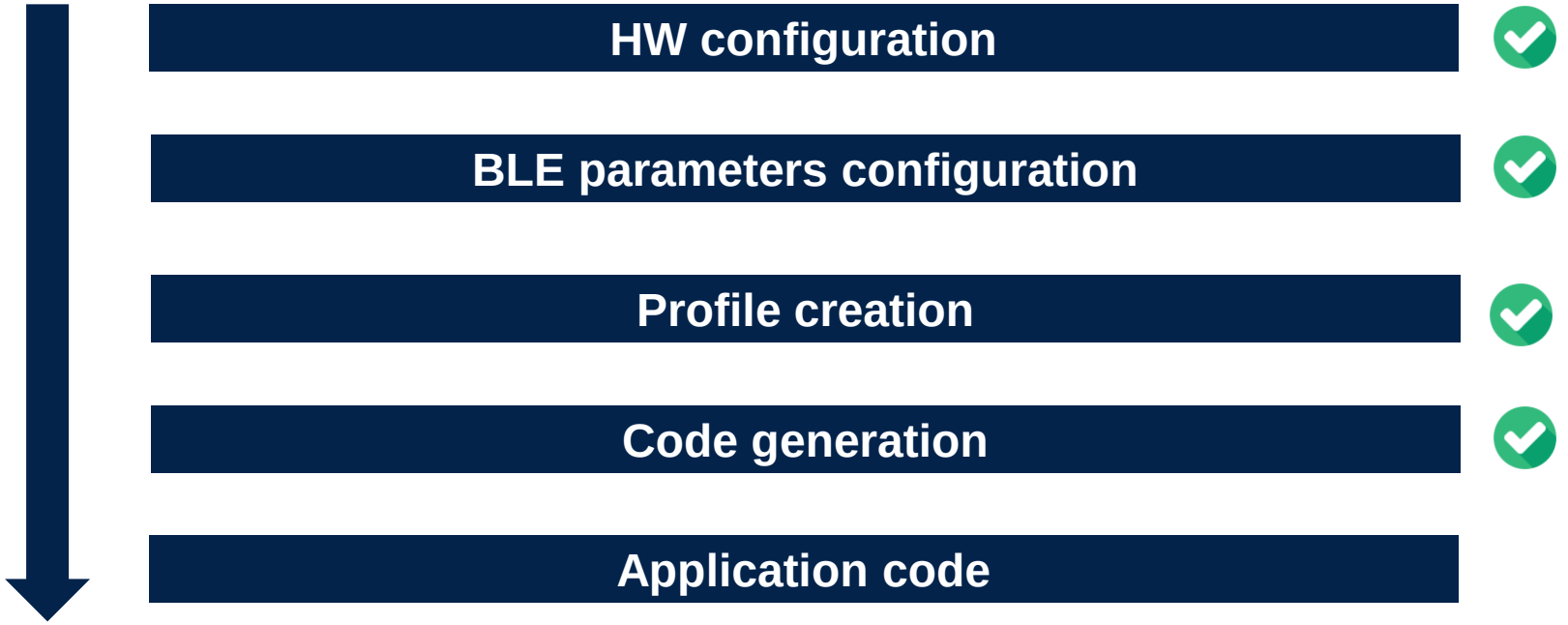
Project configuration



Project configuration

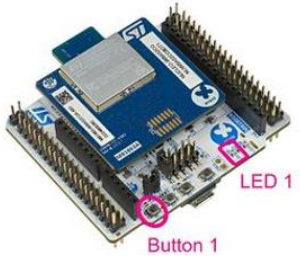


Configuration completed
What's next - Yes code generation

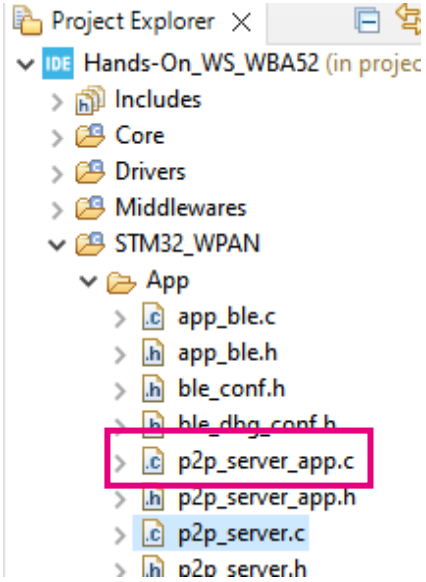


Add application code
Toggle LED from client

write to My_LED_Char (FE 41)



	Characteristic 1	Characteristic 2
UUID type	128 bits UUID (0x02)	128 bits UUID (0x02)
UUID 128 Input type	Reduced	Reduced
UUID	FE 41	FE 42
Characteristic long name	My_LED_Char	My_Switch_Char
Characteristic Short Name	LED_C	SWITCH_C
Value length	2	2
Length characteristic	Variable	Variable
Encryption key size	0x10	0x10
Char Properties	READ | WRITE_WITHOUT_RESP	NOTIFY
GATT events	GATT_NOTIFY_ATTRIBUTE_WRITE	GATT_NOTIFY_ATTRIBUTE_WRITE



write client procedure triggers an ACI_GATT_ATTRIBUTE_MODIFIED_VSEVT_CODE at server application level

in p2p_server_app.c / function P2P_SERVER_Notification()

```
/* USER CODE BEGIN Service1Char1_WRITE_NO_RESP_EV*/
HAL_GPIO_TogglePin(GPIOB, LD2_Pin|LD3_Pin|LD1_Pin);
/* USER CODE END Service1Char1_WRITE_NO_RESP_EV */<
```

life.augmented

How to add a task in sequencer ?

#1 Define a **TaskID** for your « new task » :

In app_conf.h
define a new ID in enum CFG_Task_Id_t
(USER code section)

```
/**
 * These are the lists of task id registered to the sequencer
 * Each task id shall be in the range [0:31]
 */
typedef enum
{
  CFG_TASK_HCI_ASYNCH_EVT_ID,
  CFG_TASK_LINK_LAYER,
  CFG_TASK_LINK_LAYER_TEMP_MEAS,
  CFG_TASK_BLE_HOST,
  CFG_TASK_BPKA,
  CFG_TASK_HW_RNG,
  CFG_TASK_AMM_BCKGND,
  CFG_TASK_FLASH_MANAGER_BCKGND,
  CFG_TASK_BLE_TIMER_BCKGND
  /* USER CODE BEGIN CFG_Task_Id_t */
  TASK_BUTTON_1,
  /* USER CODE END CFG_Task_Id_t */
  CFG_TASK_NBR /* Shall be LAST in the list */
} CFG_Task_Id_t;
```

#2 **UTIL_SEQ_RegTask()** to register your task in the sequencer

```
UTIL_SEQ_RegTask(1U << TASK_BUTTON_1, UTIL_SEQ_RFU, APPE_Button1Action);
```

It associates a callback to your Task.
To be done only Once

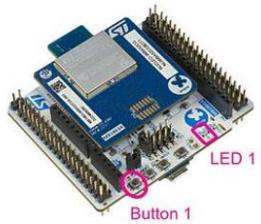
#3 **UTIL_SEQ_SetTask()** to notify the sequencer shall execute the registered task

```
UTIL_SEQ_SetTask(1U << TASK_BUTTON_1, CFG_SEQ_PRIO_0);
```

It notify the sequencer that the task must be triggered.
It will generate a call to registered function
(here : APPE_Button1Action())

life.augmented

Add application code
Raise an alarm from device to Smartphone(1/3)

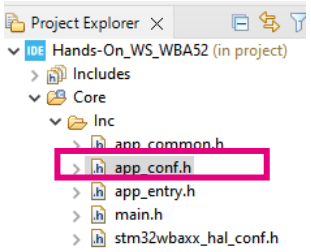


press button

notify peer device trough SWITCH_C (FE 42)

	Characteristic 1	Characteristic 2
UUID type	128 bits UUID (0x02)	128 bits UUID (0x02)
UUID 128 Input type	Reduced	Reduced
UUID	FE 41	FE 42
Characteristic long name	My_LED_Char	My_Switch_Char
Characteristic Short Name	LED_C	SWITCH_C
Value length	2	2
Length characteristic	Variable	Variable
Encryption key size	0x10	0x10
Char Properties	READ | WRITE_WITHOUT_RESP	NOTIFY
GATT events	GATT_NOTIFY_ATTRIBUTE_WRITE	GATT_NOTIFY_ATTRIBUTE_WRITE

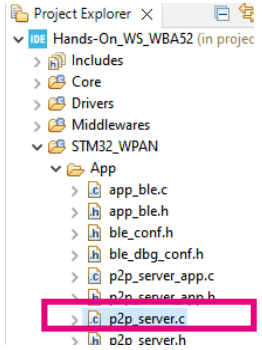
On press button use notify procedure use to push data to client



#1 need to define specific task for button press

In app_conf.h

```
/* USER CODE BEGIN CFG_Task_Id_t */
TASK_BUTTON_1,
/* USER CODE END CFG_Task_Id_t*/
```



#2 register a « button task »

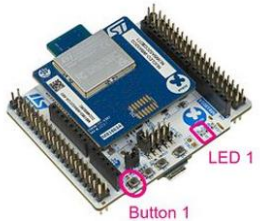
in p2p_server_app.c / function P2P_SERVER_APP_Init

```
/* USER CODE BEGIN Service1_APP_Init */
 UTIL_SEQ_RegTask( 1U << TASK_BUTTON_1, UTIL_SEQ_RFU, P2P_SERVER_Switch_c_SendNotification);
 /* USER CODE END Service1_APP_Init */
```

Function generated by CubeMx as per as Characteristic Short Name

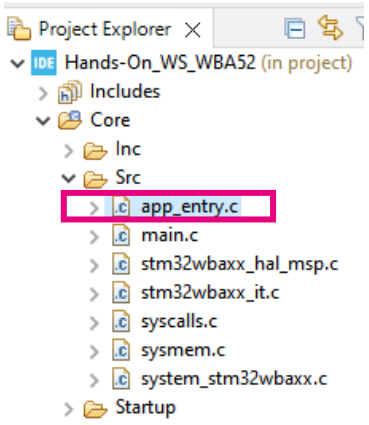
life.augmented

Add application code
Raise a notification from device to Smartphone(2/3)



notify peer device trough SWITCH_C (FE 42)

press button



#3 Manage Button1 interrupt : implement IRQ callback

In app_entry.c / function HAL_GPIO_EXTI_Rising_Callback

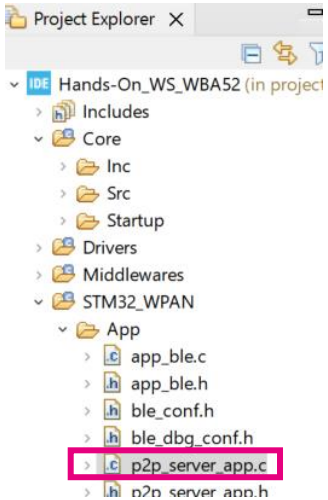
```
  /* USER CODE BEGIN FD_WRAP_FUNCTIONS */
void HAL_GPIO_EXTI_Rising_Callback(uint16_t GPIO_Pin)
{
  if (GPIO_Pin == B1_Pin)
  {
    UTIL_SEQ_SetTask(1U << TASK_BUTTON_1, CFG_SEQ_PRIO_0);
  }

  return;
} /* USER CODE END FD_WRAP_FUNCTIONS */
```

Add application code
Raise a notification from device to Smartphone(3/3)

notify peer device trough SWITCH_C (FE 42)

	Characteristic 1	Characteristic 2
UUID type	128 bits UUID (0x02)	128 bits UUID (0x02)
UUID 128 Input type	Reduced	Reduced
UUID	FE 41	FE 42
Characteristic long name	My_LED_Char	My_Switch_Char
Characteristic Short Name	LED_C	SWITCH_C
Value length	2	2
Length characteristic	Variable	Variable
Encryption key size	0x10	0x10
Char Properties	READ | WRITE_WITHOUT_RESP	NOTIFY
GATT events	GATT_NOTIFY_ATTRIBUTE_WRITE	GATT_NOTIFY_ATTRIBUTE_WRITE



#4 Manage BLE notification procedure

In p2p_server_app.c/ function P2P_SERVER_Switch_c_SendNotification

```
/* USER CODE BEGIN Service1Char2_NS_1*/

a_P2P_SERVER_UpdateCharData[0] = 0x01; /* Device Led selection */
a_P2P_SERVER_UpdateCharData[1] = 0x00;
/* Update notification data length */
p2p_server_notification_data.Length = (p2p_server_notification_data.Length) + 2;

notification_on_off = Switch_c_NOTIFICATION_ON;

/* USER CODE END Service1Char2_NS_1*/
```

Peer to Peer Service - SWITCH Characteristic		
Byte Index	0	1
Name	Button Selection	Status
Value	0x01: button 1	0x00 or 0x01

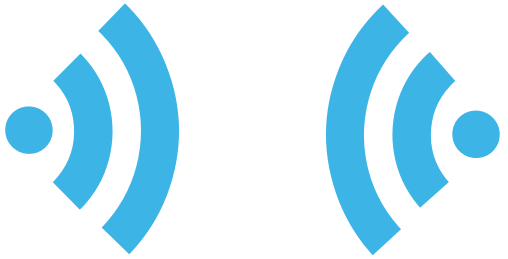
STM32WBA Bluetooth® LE – Peer 2 Peer Applications - stm32mcu

P2P_SERVER_UpdateValue

aci_gatt_update_char_value BLE stack API

life.augmented

Open your App and Connect

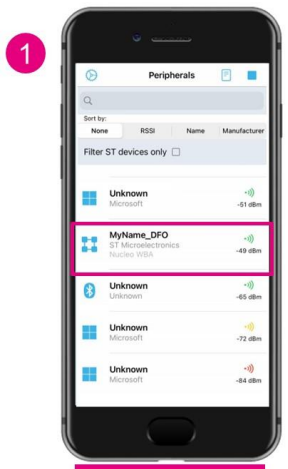


ST BLE Toolbox

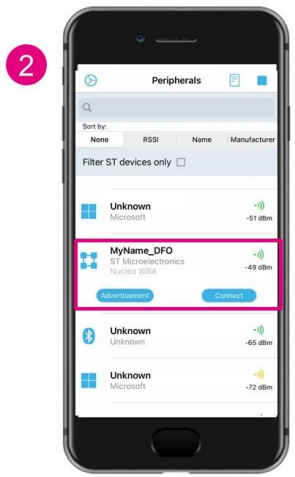
Download on the App Store

GET IT ON Google Play

Open your App and Connect (1/2)

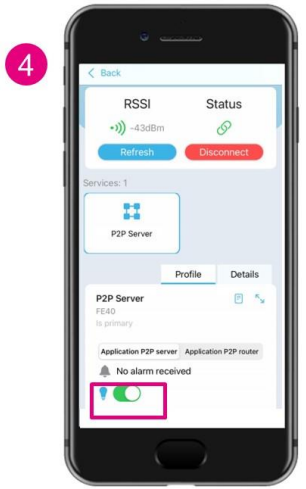


click on device



click on connect

access to profile



control LED status on Nucleo

	Characteristic 1
UUID type	128 bits UUID (0x02)
UUID 128 Input type	Reduced
UUID	FE 41
Characteristic long name	My_LED_Char
Characteristic Short Name	LED_C
Value length	2
Length characteristic	Variable
Encryption key size	0x10
Char Properties	READ | WRITE_WITHOUT_RESP
GATT events	GATT_NOTIFY_ATTRIBUTE_WRITE

write client procedure control LED

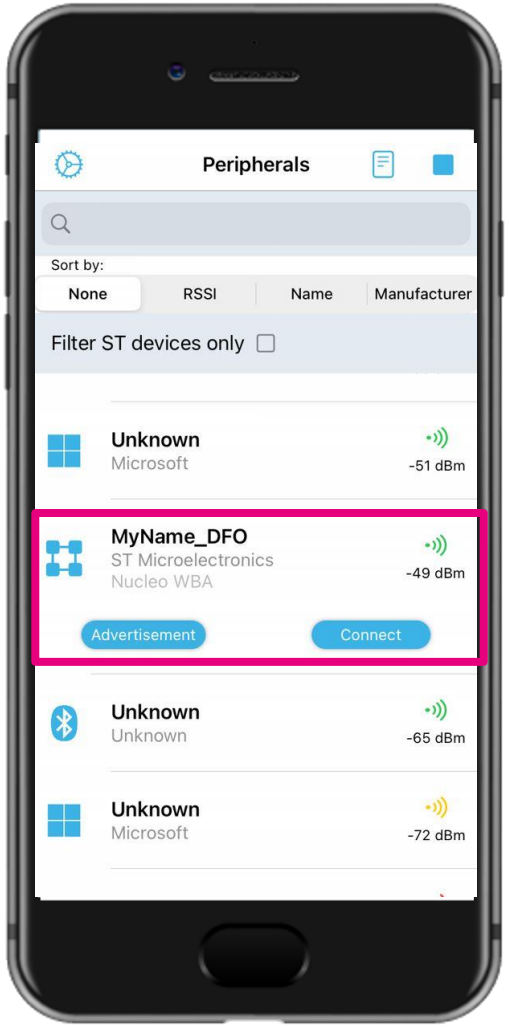
push button 1 and notify device

	Characteristic 2
UUID type	128 bits UUID (0x02)
UUID 128 Input type	Reduced
UUID	FE 42
Characteristic long name	My_Switch_Char
Characteristic Short Name	SWITCH_C
Value length	2
Length characteristic	Variable
Encryption key size	0x10
Char Properties	NOTIFY
GATT events	GATT_NOTIFY_ATTRIBUTE_WRITE

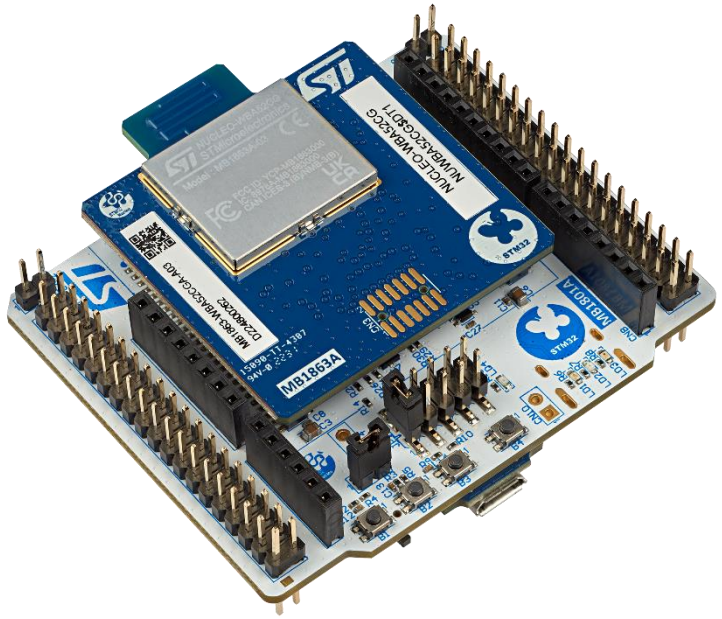
notify procedure use to push data to client

STBLE Toolbox (Connection)

1



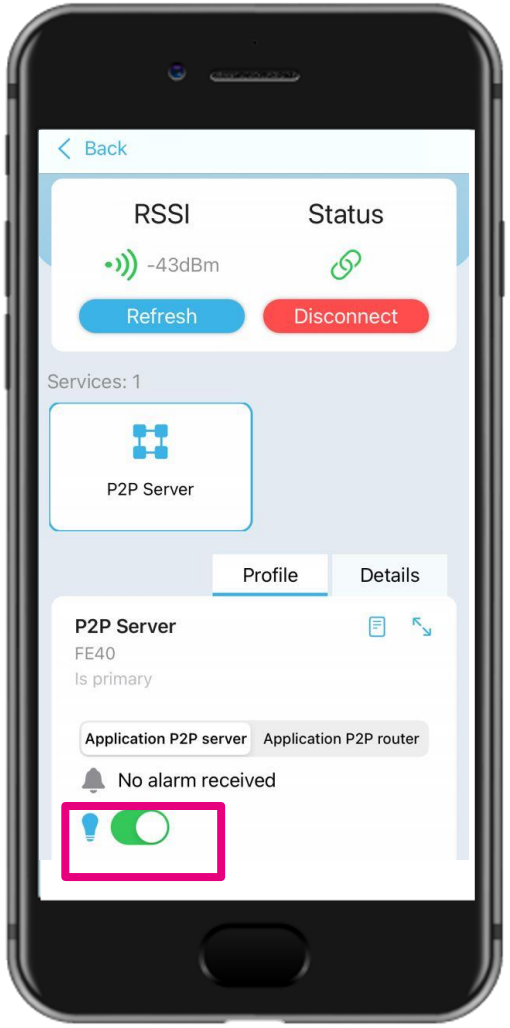
2



click on connect

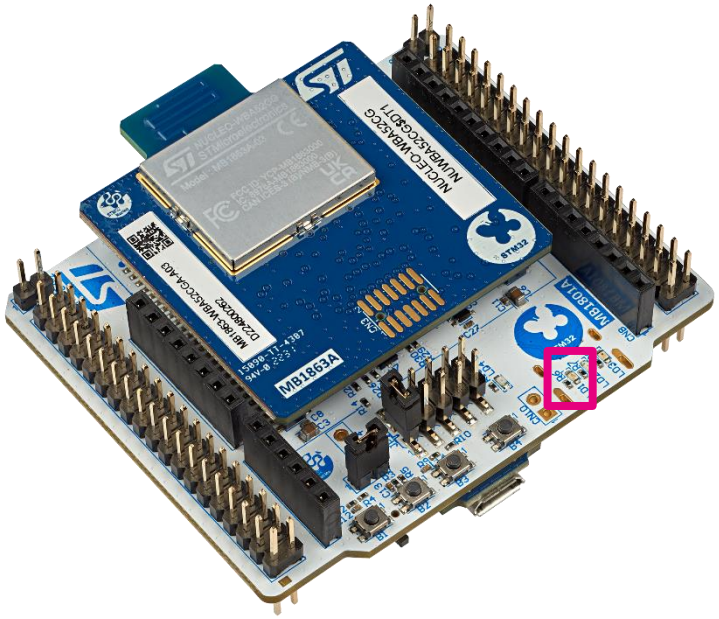
access to profile

STBLE Toolbox (LED)



Notification - SWITCH

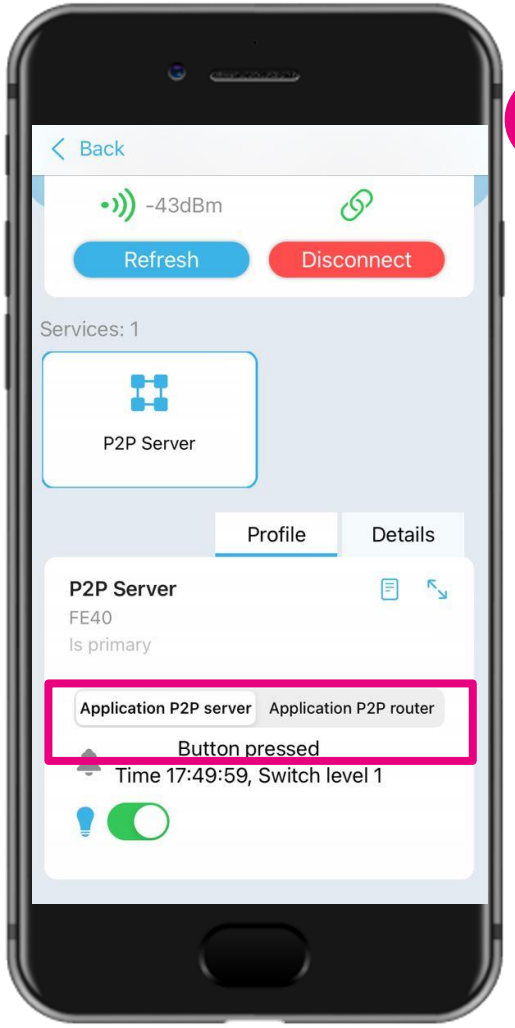
Write - LED



control LED status on Nucleo

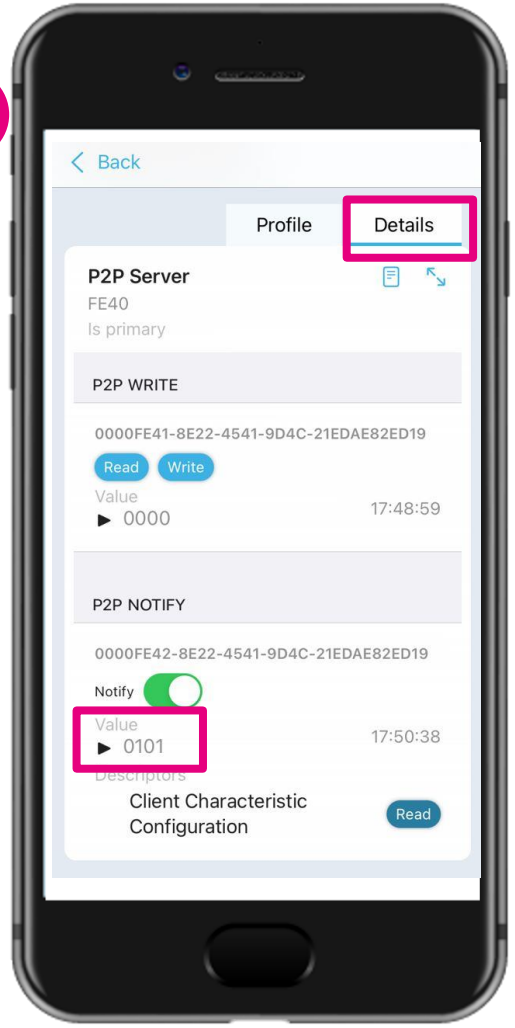
STBLE Toolbox (Push Button)

1

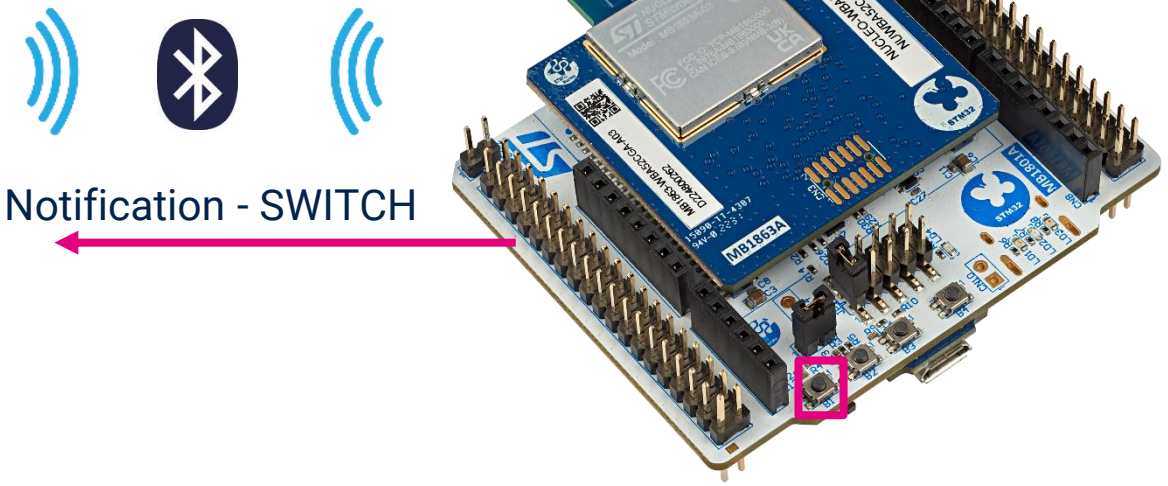


push button 1 and notify device

2



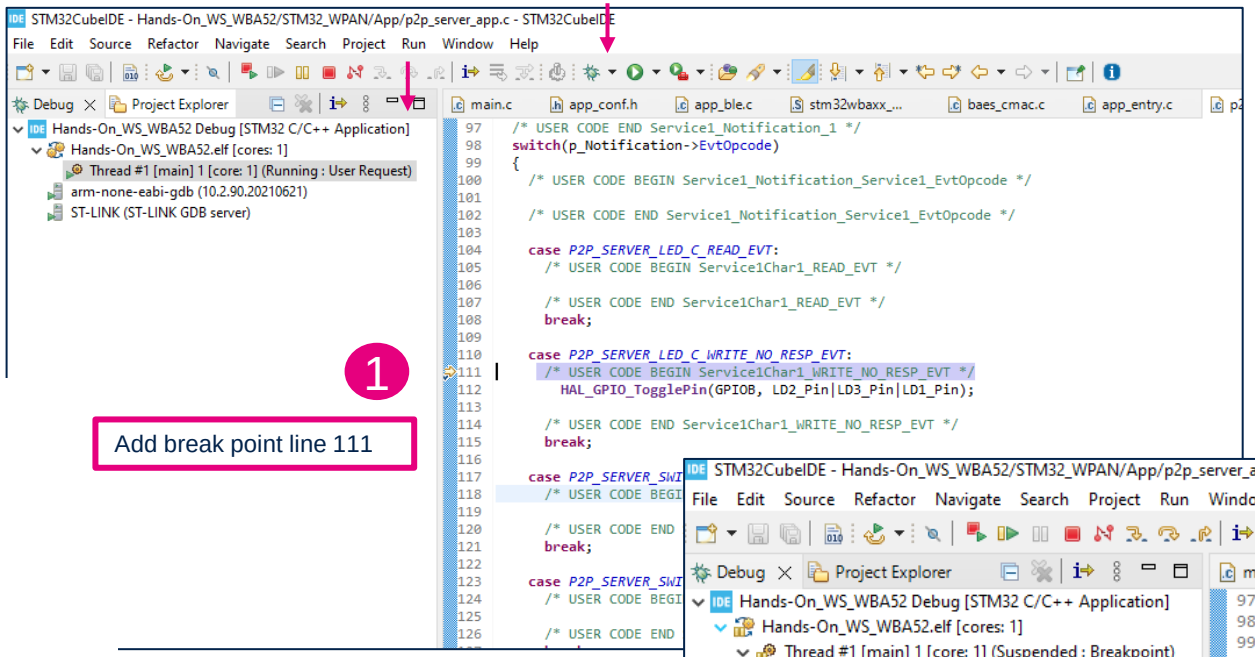
click on details to see bytes sent/received



Open your App and Connect call stack

3

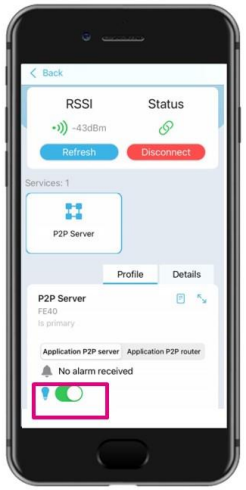
2



1

Add break point line 111

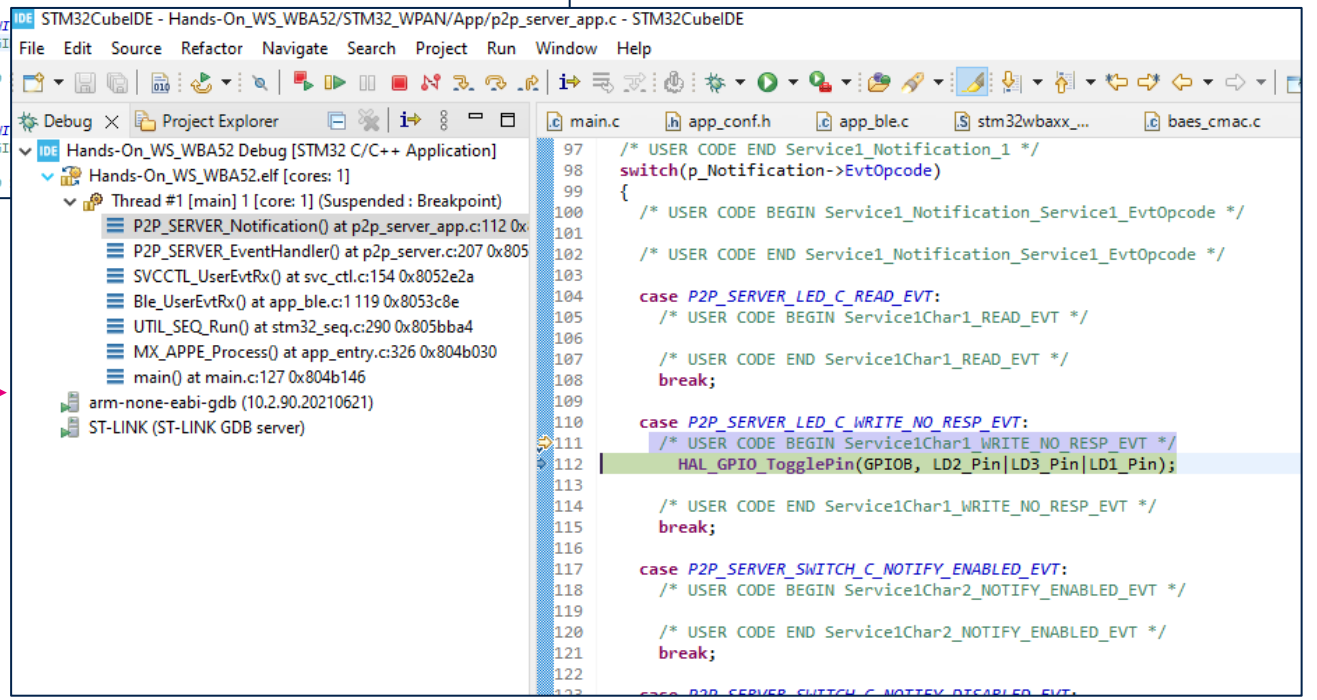
4



control LED status on Nucleo

BLE write procedure initiated by client

ACI_GATT_ATTRIBUTE_MODIFIED event received at application level

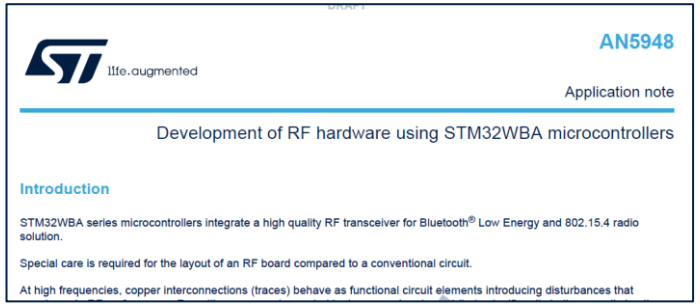


HW project development & certification

life.augmented

A complete set of documentation

AN5948 : Development of RF Hardware using STM32WBA


life.augmented

AN5948

Application note

Development of RF hardware using STM32WBA microcontrollers

Introduction

STM32WBA series microcontrollers integrate a high quality RF transceiver for Bluetooth® Low Energy and 802.15.4 radio solution.

Special care is required for the layout of an RF board compared to a conventional circuit.

At high frequencies, copper interconnections (traces) behave as functional circuit elements introducing disturbances that

COMING SOON on st.com!

- RF basis generalities
- Schematics & components selection guidelines
- STM32WBA5x layout checklist & guidelines

STM32WBA5x CAD resources on st.com

- Download STM32WBA5x symbol
- Download STM32WBA5x footprint

life.augmented

HW design with STM32WBA5x : key points

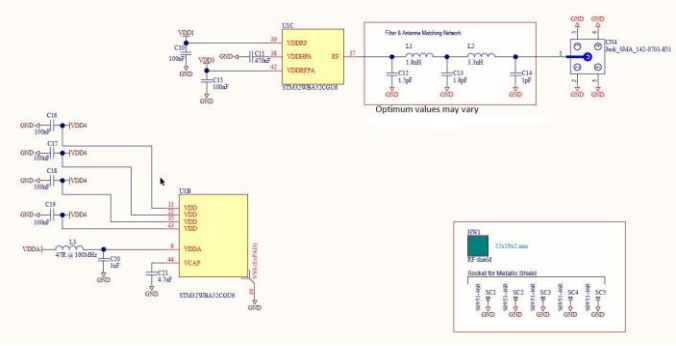
- HW design should be initiated based on documentation shown on previous slides
- Next slides are highlighting the key points you must pay attention when designing schematics and layout:

➢ <u>HSE 32MHz Xtal requirements:</u>

- STM32WBA5x includes internal programmable capacitances to trim the crystal frequency
- Select XTAL with 8pF load capacitors
- Recommended part (or equivalent): NX1612SA-32MHZ-EXS00A-CS09166

➢ <u>LSE or LSI selection. LSE 32kHz Xtal requirements if used:</u>

- LSE mandatory for accurate RTC calendar application.
- BOM optimized (save 32kHz Xtal cost)



CONFIDENTIAL

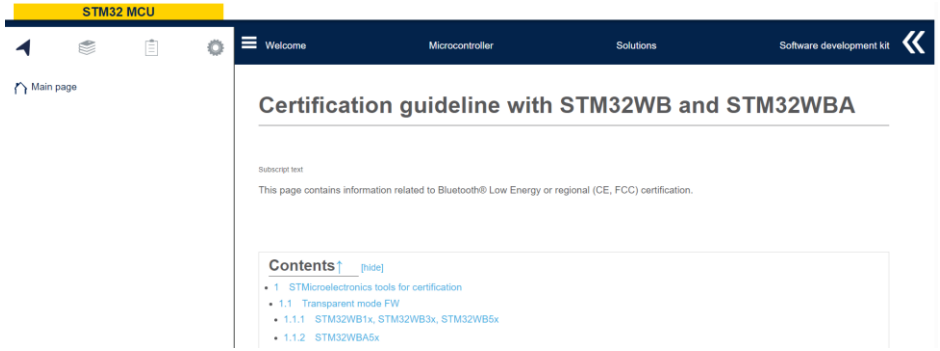
HW design with STM32WBA5x : key points

- RF matching & filtering:
  - Integrated balun so single ended RF matching.
  - Very limited number of discrete components for STM32WBAx matching and filtering.
- Power management. SMPS implementation for STM32WBA55 use case:
  - STM32WBA55 embeds a SMPS that can be used to improve power efficiency.
- Main layout recommendation:
  - Refer our various reference kits layout (Gerbers and Altium files available).
  - 4 layers stack-up recommended but 2 layers is possible.

CONFIDENTIAL

STM32WBA5x : a certified solution

- STM32WBA5x is compliant in regards of regional (CE, FCC etc.) and Bluetooth requirements.
- We are providing complete set of documentation, FW and tools to certify your product.



Full set of tools and documentation
➢ Certification guideline on wiki (Certification guideline with STM32WB and STM32WBA - stm32mcu)
➢ Transparent mode FW available in STM32CubeWBA MCU Package
➢ STM32CubeMonRF PC tool

Bluetooth certification

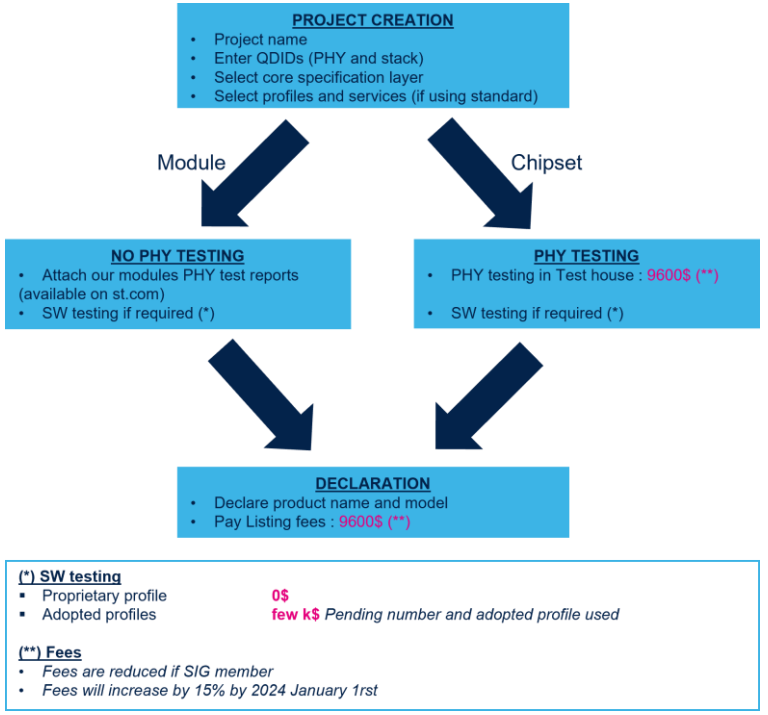
- STM32WBA5x is having reference QDIDs (components and stack) at Bluetooth SIG.
- Customer musty performed PHY testing and declare its product at Bluetooth SIG.

PHY QDID

Package	Part number	Cut version	RF PHY QDID
QFN48	STM32WBA52 (BLE5.4)	1.x	**197135** (TCRL 2022-2)

Stack QDID

Features	Host Stack version	QDID
4.0 HCI Low Energy LL with extended advertising – ATT – GAP – GATT – L2CAP with Enhanced Connected Oriented Channel -SMP BLE 5.3	STM32Cube_WBA_BLE_HCI_STACK STM32Cube_WBA_BLE_FULL_STACK	**198195** (TCRL 2022-1)



Refer wiki Certification guideline on st.com

Thank you

© STMicroelectronics - All rights reserved.
The STMicroelectronics corporate logo is a registered trademark of the STMicroelectronics group of companies. All other names are the property of their respective owners.

life.augmented