



life.augmented

Welcome to **STM32WBA52 workshop**

Hands-on #2
Build basic **p2pServer**
application and connect

Workshop team





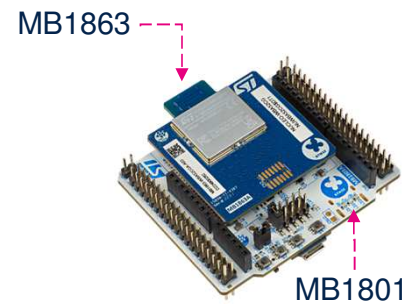
SW prerequisites

- STM32CubeMX software (v6.9.0 or up) - Optional
- STM32CubeWBA MCU package (v1.1.0 or up)
- IDE: STM32CubeIDE (v1.13.0 or up)
- A serial terminal (e.g. TeraTerm)
- **ST BLE ToolBox Smartphone application**

HW prerequisites

- NUCLEO-WBA52
- USB A to Micro-B Cable

Prerequisites Refresh



ST BLE Toolbox



Agenda



1 Hands-on Presentation

3 Step 2 : Application code

2 Step 1 : Profile creation
demystification and details



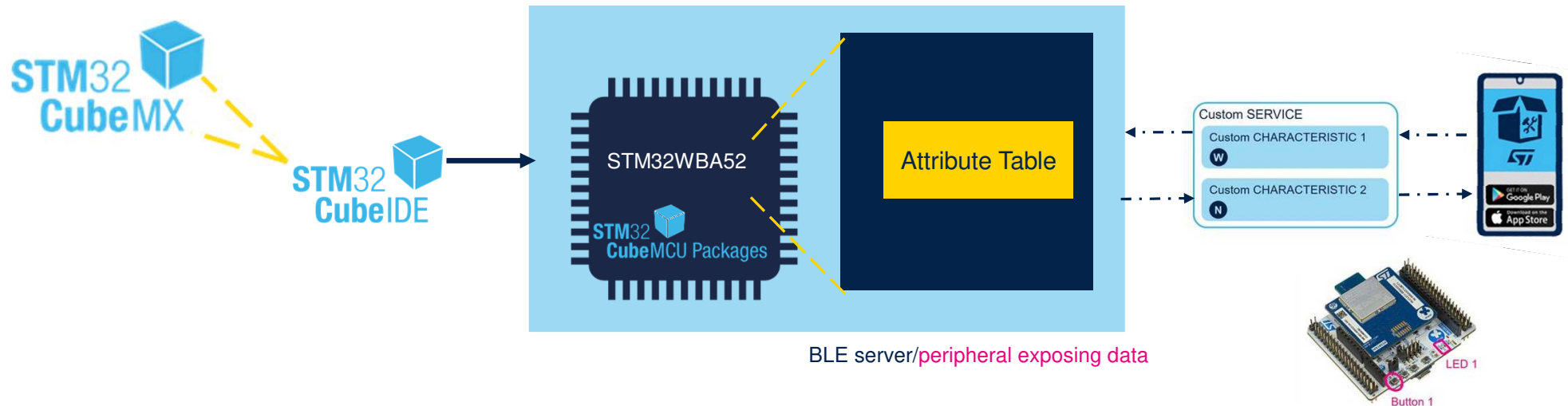
life.augmented

Hands-on presentation



Purpose

- The purpose is to start from WBA52 chipset level and build a basic server (**p2pServer**) application using STM32CubeMX and associated STM32CubeIDE
- In this second part, focus is to enhance existing application code (Hands-on #1) to **control device and share data**



Enhance application code to enable a Bluetooth® Low Energy Application Profile (**p2pServer**)



Legenda

- Slides including following symbol are purely theoretical ones



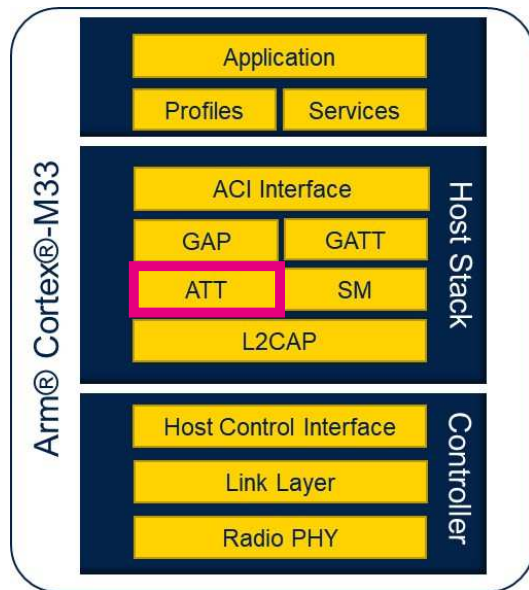
- Source code for development is included inside blue boxes

```
HAL_Delay(500);
```

Step1 : GAP/GATT custom application configuration : **Profile creation**



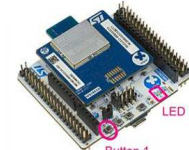
What is a Bluetooth Low Energy Profile **Attribute Protocol (ATT)**



Define **Client** - Server architecture



« I am looking for data »
Client

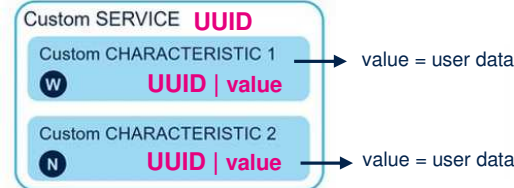


« I have data to share » (sensors, raw data...**what you want !**)
Server

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

user data is accessible trough attribute

service
└─ characteristic
 └─ data





What is a Bluetooth Low Energy Profile ?

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

Profile

Service **UUID**

Characteristic

R **UUID**

Service **UUID**

Characteristic 1

W **UUID**

Characteristic 2

R N **UUID**

- All attributes have a type which is identified by a UUID (**U**niversally **U**nique **I**dentifier)
- Characteristic can take 3 types of properties: **READ**, **WRITE**, **NOTIFY**
- 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**





Bluetooth® Low Energy standard profile vs. proprietary profile

Standard Heart Rate Profile

Heart Rate Service 0x180D

**Heart Rate Measurement
Characteristic** 0x2A37

**Body sensor Location
Characteristic** 0x2A38

Proprietary ST P2P Server Profile

P2P Service 0x0000FE40CC7A482A984A7F2ED5B3E58F

My_LED_Char 0x0000FE41CC7A482A984A7F2ED5B3E58F

SWITCH_C 0x0000FE42CC7A482A984A7F2ED5B3E58F

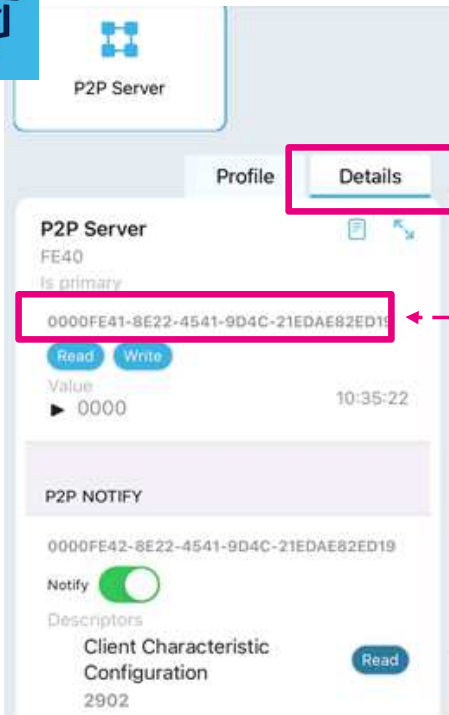
Define by the **SIG**, define the role, requirements, behavior and the structure of Attribute Table of each entity (central & peripheral)

Any standard smartphone App will be able to communicate

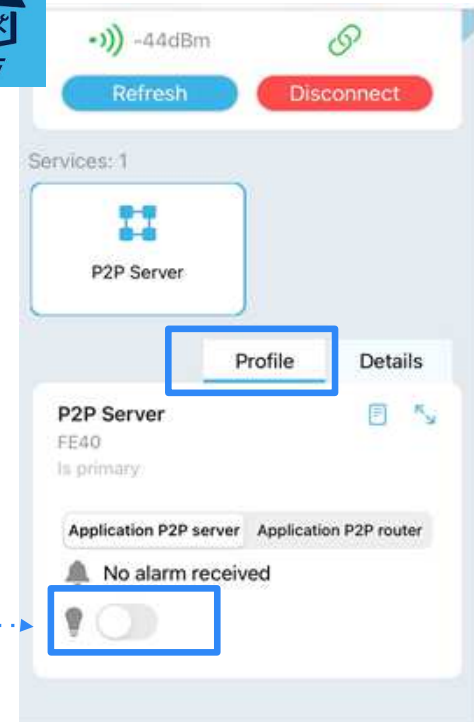
Define your own behavior using your own Attribute Table based in 128 bits UUID

Only your own App will be able to communicate





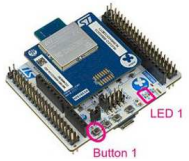
Proprietary profile ST Toolbox App



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



Data exchanges what is the magic behind ?

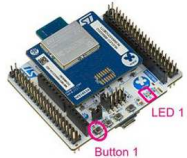


#1 At profile initialization and entry point (**handle @**) will be created in RAM to expose data to client



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 (**handle**) 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 handle make data exchange possible



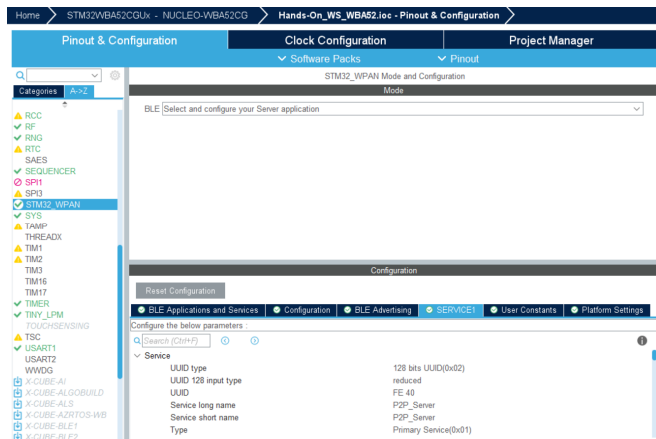
Profile Creation

Hands-on #1



You succeed

start back from running .ioc

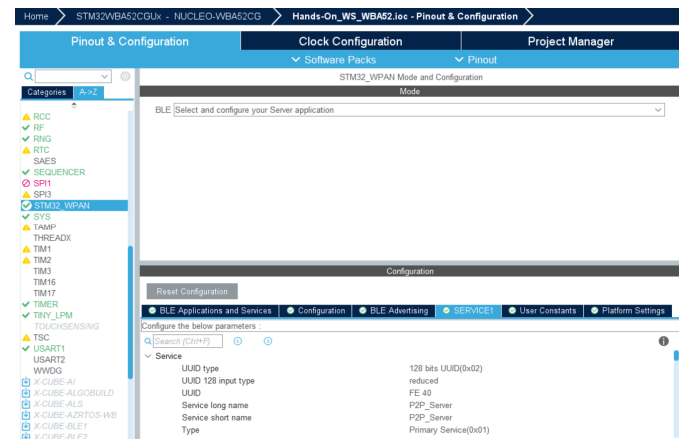


Hands-on #1



More or less...

load Hands-On_WS_WBA52_backup.ioc





Profile Creation Service

Pinout & Configuration

Clock Configuration

Software Packs

Pinout

STM32_WPAN Mode and Configuration

Mode

BLE Select and configure your Server application

Configuration

Reset Configuration

BLE Advertising

SERVICE1

BLE Applications and Services

User Constants

Platform Settings

Configure the below parameters :

Search (Ctrl+F)

Server Mode

Number of services

1

BLE Services Configuration

Peripheral Role	1
Central Role	0
BLE_CFG_SVC_MAX_NBR_CB	7
BLE_CFG_CLT_MAX_NBR_CB	0

P2P Server Profile

P2P Service

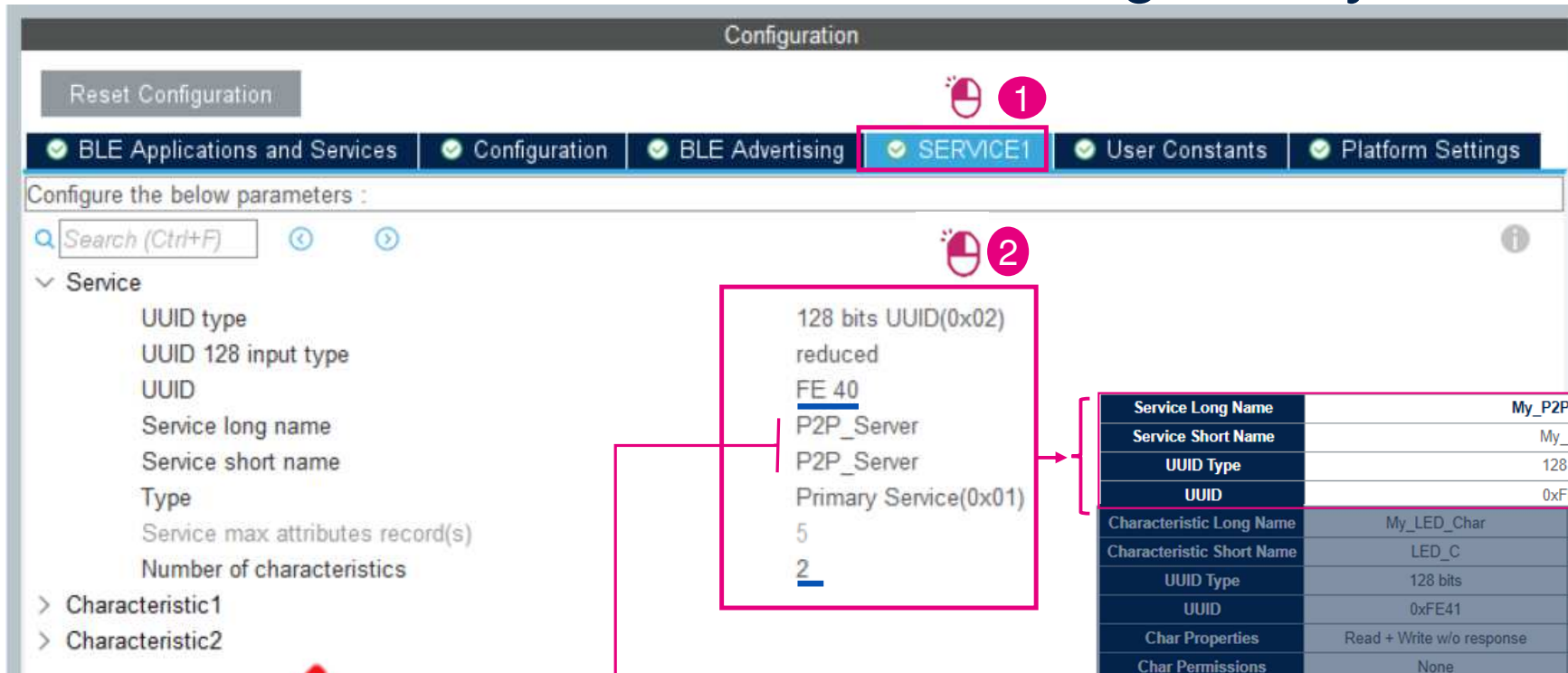
My_LED_Char

SWITCH_C



Profile Creation

Configure my P2P Service



service & characteristic naming used to
name function at code generation
Use : "P2P_Server"

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

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

Configure the below parameters :

Search (Ctrl+F)

> Service

Characteristic1

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_PROP_BROADCAST: No

CHAR_PROP_READ: Yes

CHAR_PROP_WRITE WITHOUT RESP: Yes

CHAR_PROP_WRITE: No

CHAR_PROP_NOTIFY: No

CHAR_PROP_INDICATE: No

ATTR_PERMISSION_AUTHEN_READ: No

ATTR_PERMISSION_AUTHOR_READ: No

ATTR_PERMISSION_ENCRY_READ: No

ATTR_PERMISSION_AUTHEN_WRITE: No

ATTR_PERMISSION_AUTHOR_WRITE: No

ATTR_PERMISSION_ENCRY_WRITE: No

GATT_NOTIFY_ATTRIBUTE_WRITE: Yes

GATT_NOTIFY_WRITE_REQ_AND_WAIT_FOR_APPL_RESP: No

GATT_NOTIFY_READ_REQ_AND_WAIT_FOR_APPL_RESP: No

> Characteristic2



Characteristic short name used at code generation
Use : "LED_C"

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



Profile Creation

Configure 2nd Characteristic

BLE Applications and Services Configuration BLE Advertising SERVICE1

Configure the below parameters :

Search (Ctrl+F)

> Service

> Characteristic1

✓ Characteristic2

1

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_PROP_BROADCAST No

CHAR_PROP_READ No

CHAR_PROP_WRITE_WITHOUT_RESP No

CHAR_PROP_WRITE No

CHAR_PROP_NOTIFY Yes

CHAR_PROP_INDICATE No

Update char value offset 0

ATTR_PERMISSION_AUTHEN_READ No

ATTR_PERMISSION_AUTHOR_READ No

ATTR_PERMISSION_ENCRY_READ No

ATTR_PERMISSION_AUTHEN_WRITE No

ATTR_PERMISSION_AUTHOR_WRITE No

ATTR_PERMISSION_ENCRY_WRITE No

GATT_NOTIFY_ATTRIBUTE_WRITE Yes

GATT_NOTIFY_WRITE_REQ_AND_WAIT_FOR_APPL_RESP No

GATT_NOTIFY_READ_REQ_AND_WAIT_FOR_APPL_RESP No

2

!

life.augmented

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

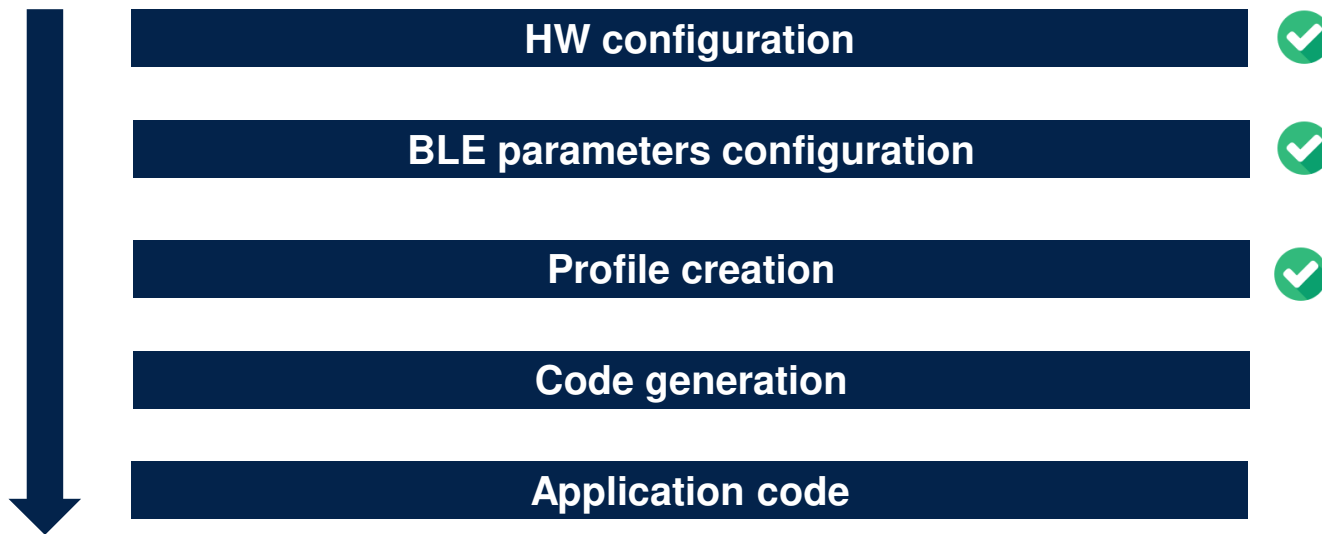
	Characteristic 1	Characteristic 2
UUID type	128 bits UUID (0x02)	128 bits UUID (0x02)
UUID 128 Input type	Reduced	Reduced
UUID	FE 41	FE 42
Characteristic long name	My_LED_Char	My_Switch_Char
Characteristic Short Name	LED_C	SWITCH_C
Value length	2	2
Length characteristic	Variable	Variable
Encryption key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

Characteristic short name used at code generation
Use : **"SWITCH_C"**



Configuration completed

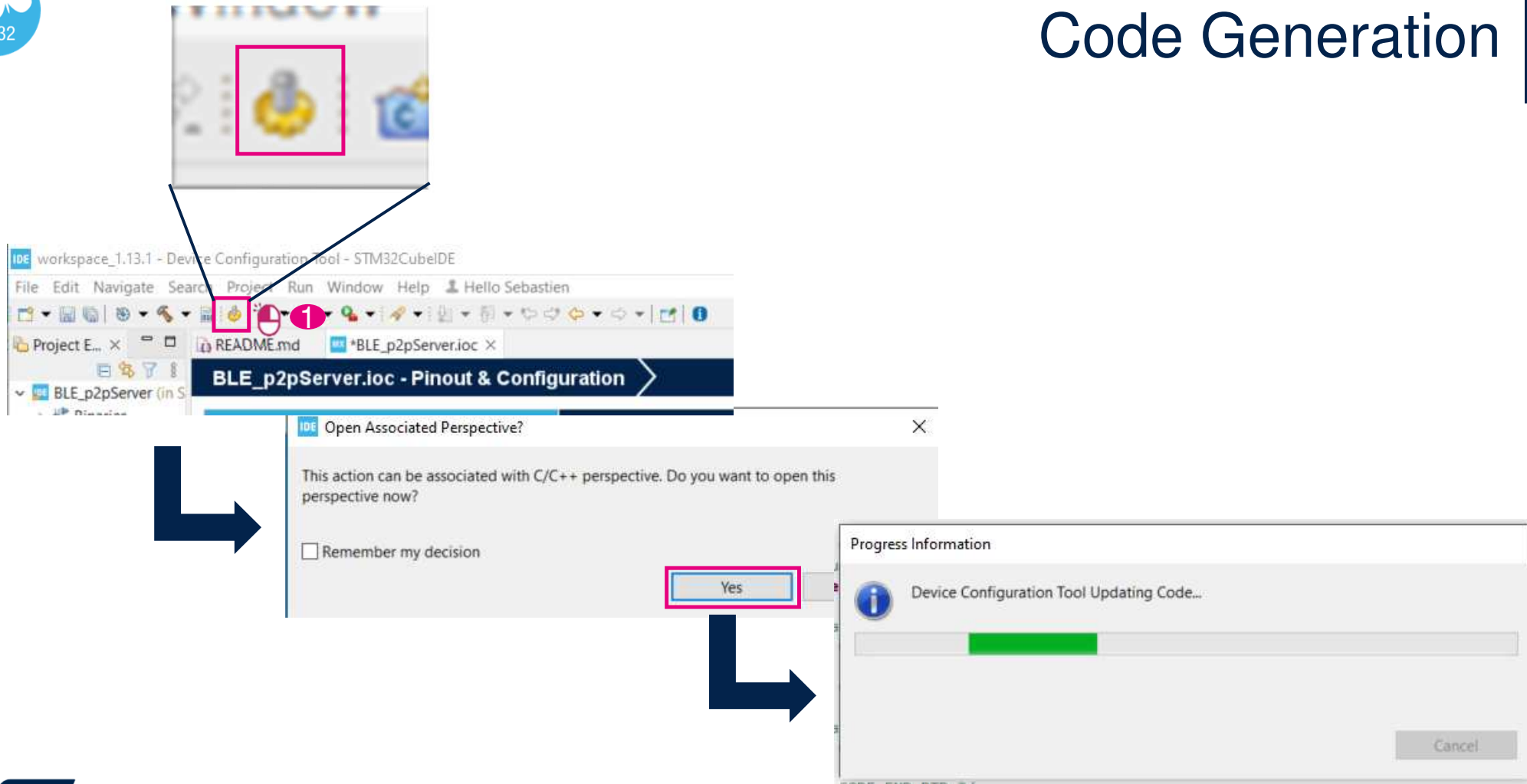
What's next - Yes code generation



Step 2 : Code generation and user application code



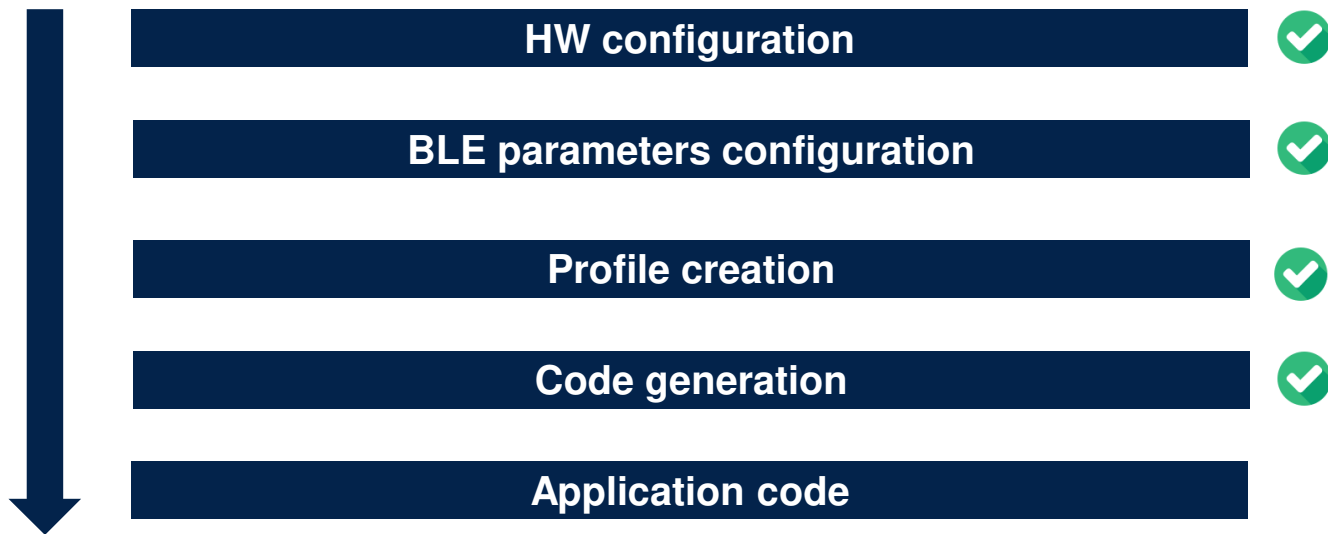
Code Generation





Configuration completed

What's next - Yes code generation





Remove previous code

Why should I remove previous functional code ?



As we have created profile, STM32CubeMX generated new skeleton code with more friendly APIs

Let's use this API to move to discoverable !

```
/* USER CODE BEGIN APP_BLE_Init_2 */
tBleStatus status;
status = aci_gap_set_discoverable(ADV_TYPE, ADV_INTERVAL_MIN, ADV_INTERVAL_MAX,
                                CFG_BD_ADDRESS_TYPE,
                                ADV_FILTER,
                                0, 0, 0, 0, 0);
if (status != BLE_STATUS_SUCCESS) {
    return;
}

status = aci_gap_delete_adv_type(ADV_TYPE_TX_POWER_LEVEL);
if (status != BLE_STATUS_SUCCESS) {
    return;
}

status = aci_gap_update_adv_data(sizeof(a_AdvData), (uint8_t*) a_AdvData);
if (status != BLE_STATUS_SUCCESS) {
    return;
}
/* USER CODE END APP_BLE_Init_2 */
```



Search for “APP_BLE_Init_2”

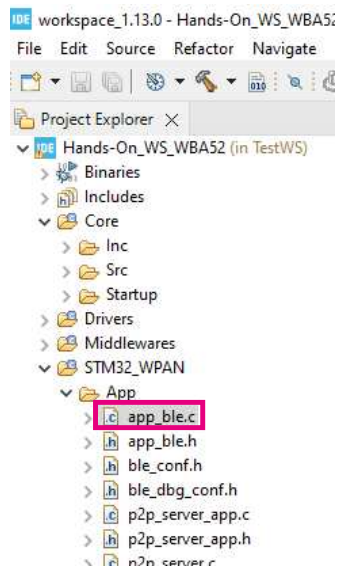
```
/* USER CODE BEGIN EVT_DISCONN_COMPLETE */
tBleStatus status;
status = aci_gap_set_discoverable(ADV_TYPE, ADV_INTERVAL_MIN, ADV_INTERVAL_MAX,
                                CFG_BD_ADDRESS_TYPE,
                                ADV_FILTER,
                                0, 0, 0, 0, 0);
if (status != BLE_STATUS_SUCCESS) {
    return;
}
/* USER CODE END EVT_DISCONN_COMPLETE */
```



Search for “EVT_DISCONN_COMPLETE ”



Add application code to move to discoverable



Set device discoverable at init :

In app_ble.c > function APP_BLE_Init()

```
/* 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 */
```

(ADV_MIN+ADV_MAX)/2

Search for “APP_BLE_Init_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 END EVT_DISCONN_COMPLETE */
```

Search for “EVT_DISCONN_COMPLETE”

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

Open Project

Add application code to move to discoverable

Build& Flash

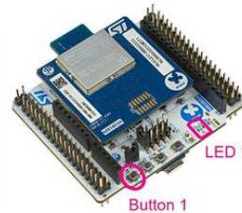
Please refer to cheatsheet for copy/paste



Add application code Toggle LED from client

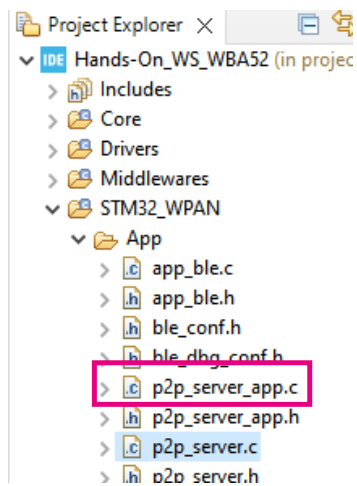


write to My_LED_Char (FE 41)



	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 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_EVT */  
HAL_GPIO_TogglePin(GPIOB, LD2_Pin|LD3_Pin|LD1_Pin);  
/* USER CODE END Service1Char1_WRITE_NO_RESP_EVT */
```

Search for "Service1Char1_WRITE_NO_RESP_EVT"



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_ASYNC_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_APM_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())

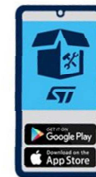


Add application code

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



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

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 */
```

Search for "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 */
```

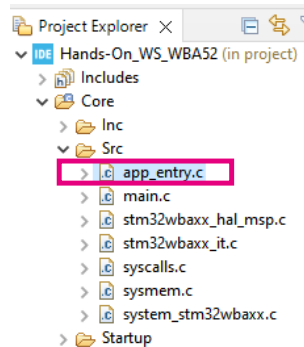
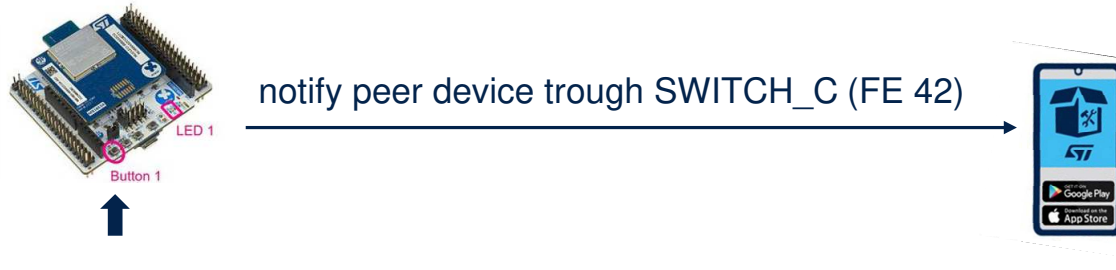
Search for "Service1_APP_Init"

Function generated by CubeMx as per as Characteristic Short Name



Add application code

Raise an alarm from device to Smartphone(2/3)



#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);
    }

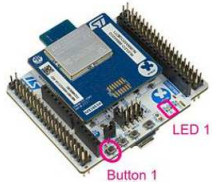
    return;
} /* USER CODE END FD_WRAP_FUNCTIONS */
```

Copy function (weak) at end of file – under FD_WRAP_FUNCTIONS tags



Add application code

Raise an alarm 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

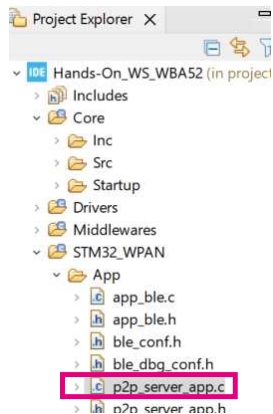
In p2p_server_app.c/ function P2P_SERVER_Switch_c_SendNotification

```
/* 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



Search for "Service1Char2_NS_1"

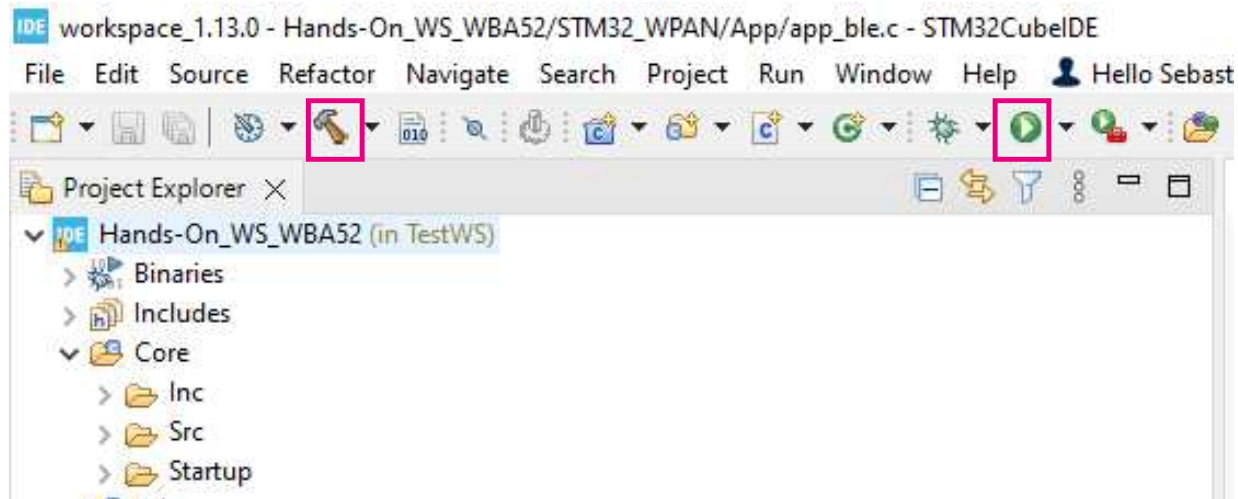
P2P_SERVER_UpdateValue

aci_gatt_update_char_value

BLE stack API

Time to build, flash and execute !

1 Build



2 Flash & Run

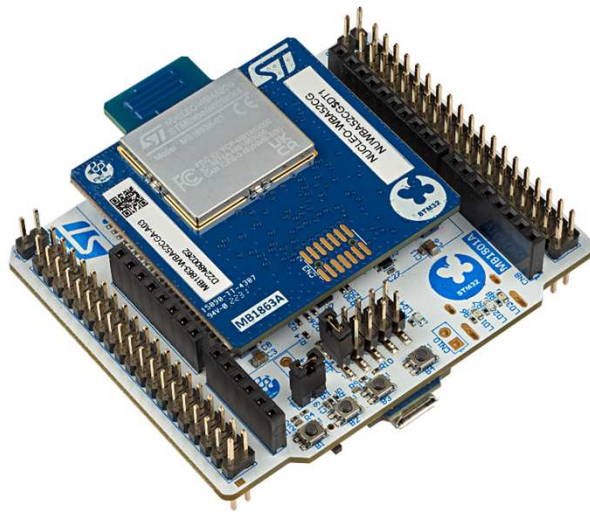
Open Project

Add application code to move to discoverable

Build& Flash



Open your App and Connect

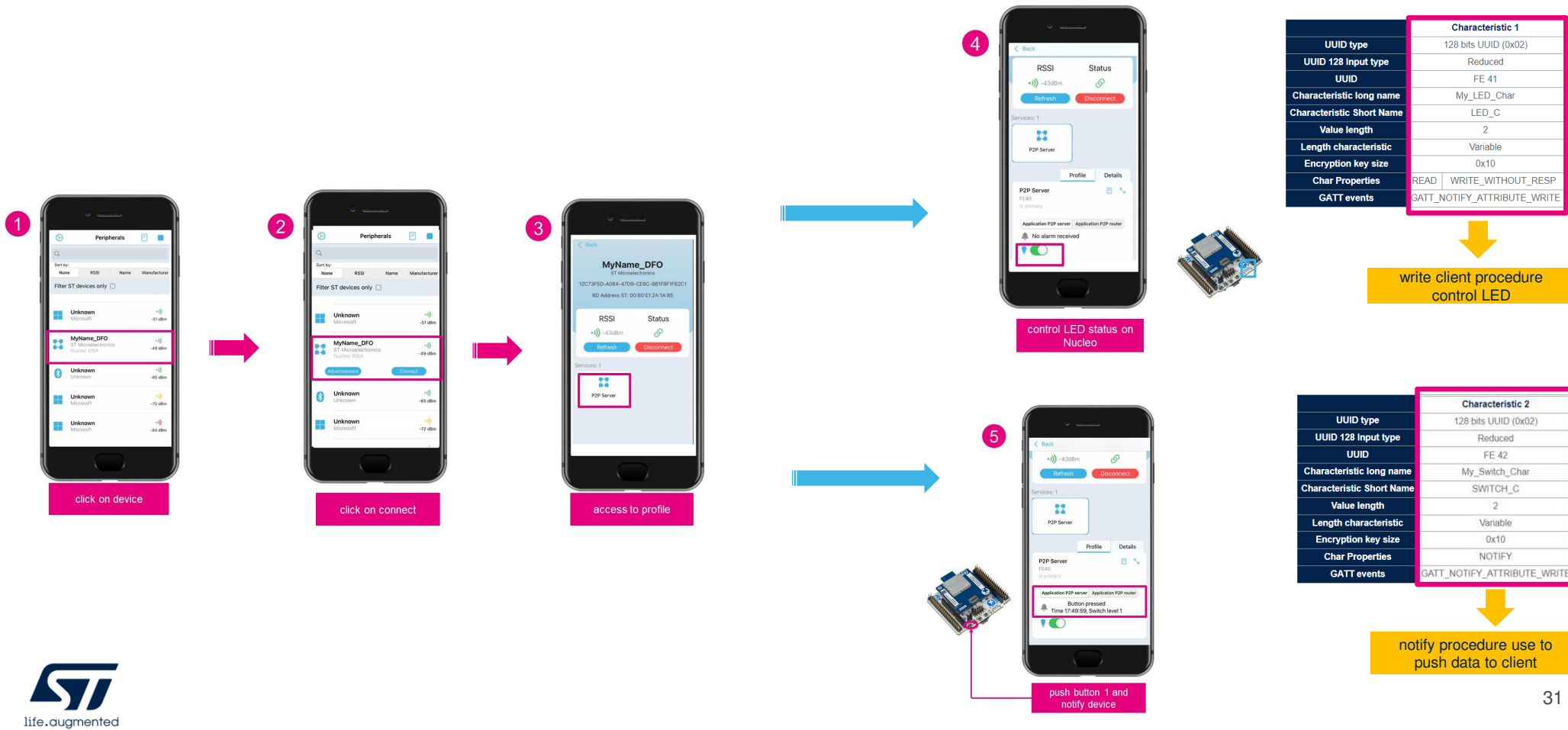


ST BLE Toolbox





Open your App and Connect (1/2)





Bonus : Open your App and Connect call stack

4



control LED status on Nucleo



BLE write procedure initiated by client

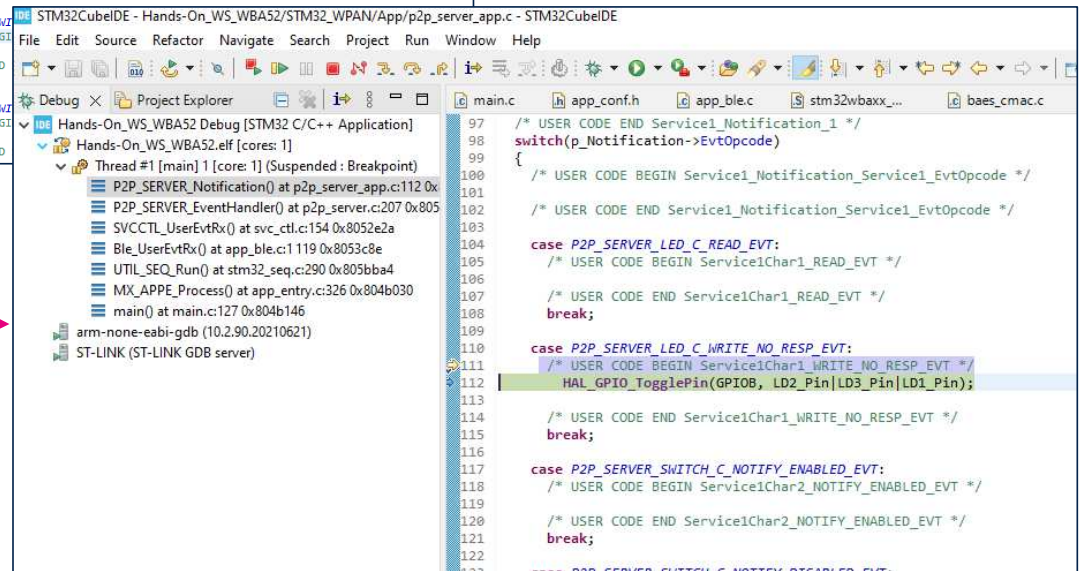
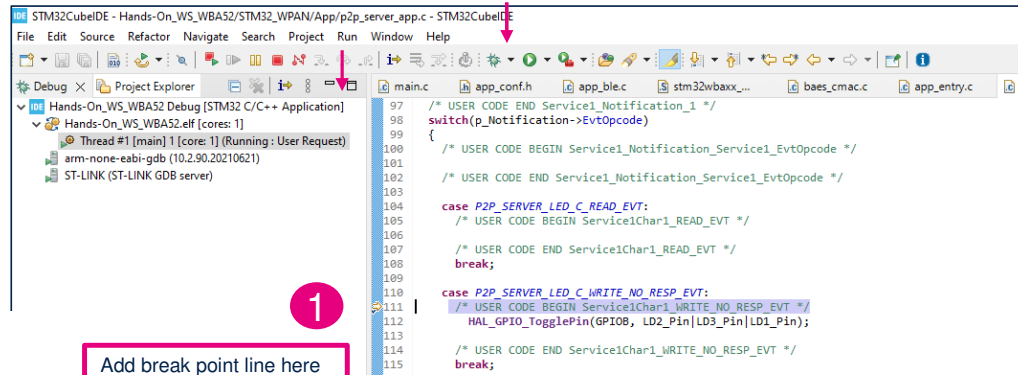
ACI_GATT_ATTRIBUTE_MODIFIED event received at application level

3

2

1

Add break point line here





Takeaways

What's next



Hands-on#2 – Build a BLE advertising device

Evaluate, prototype & customize your own project with your own BLE proprietary profile requirements.



Build and optimize you PCB and move to certification

HW guideline, what are the available resources what I should focus on.

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.

