STM32 Quest : 2024 University Developer Contest (GFX & Wireless)

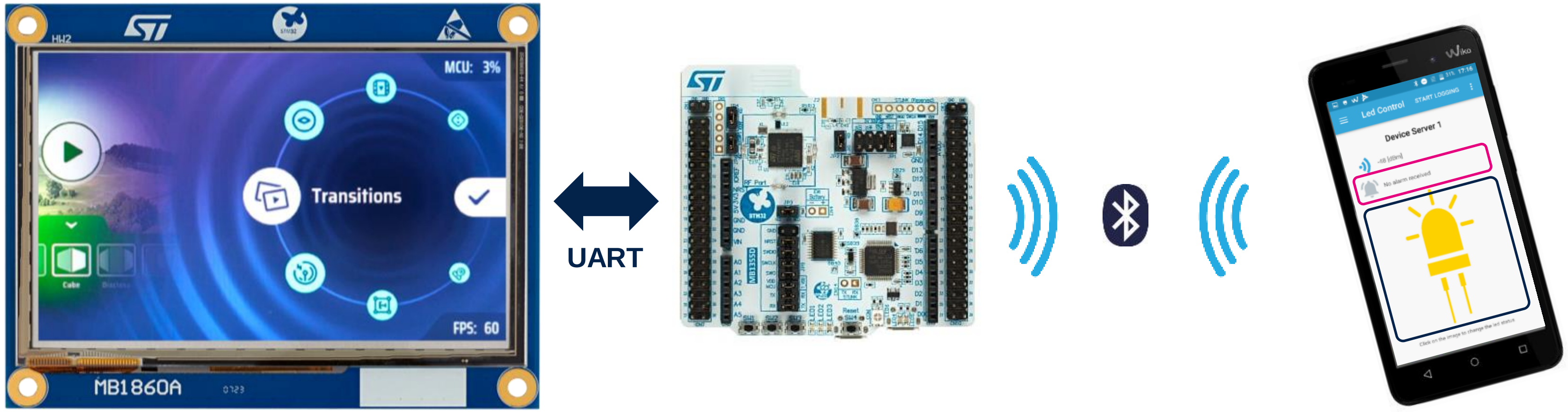
Mission 3 : Graphics + Wireless integration

STMicroelectronics

life.augmented

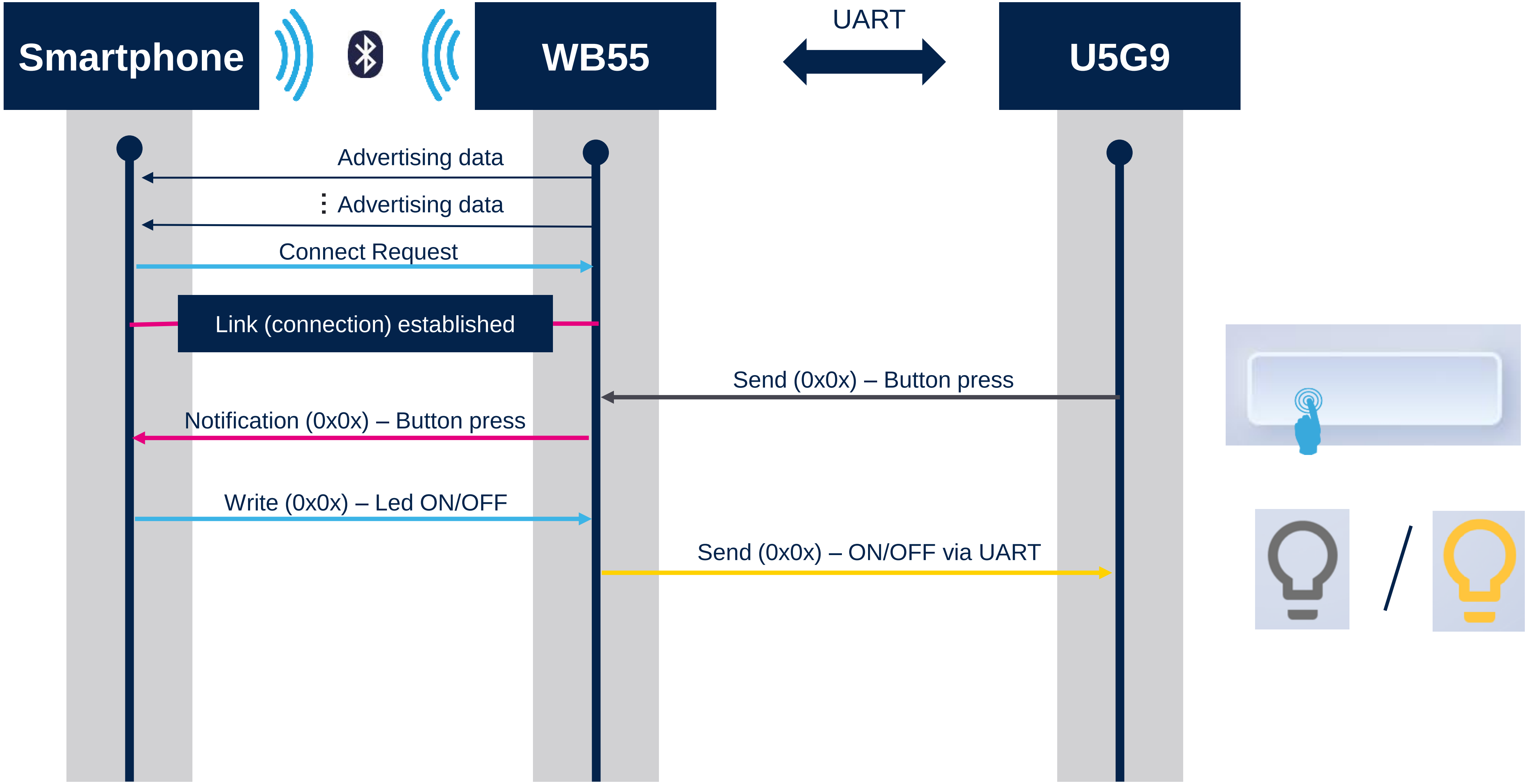
Mission overview

- Connect the STM32U5G9J-DK2 to STM32WB55RG-NUCLEO via UART
- When receiving data from BLE, send it to U5G9-DK2 over UART then pass it to the GUI task and update the GUI content.
- When pressing a button on the display, send data to WB via UART and WB should transmit the data to the smartphone over BLE.



life.augmented

Behavior graph



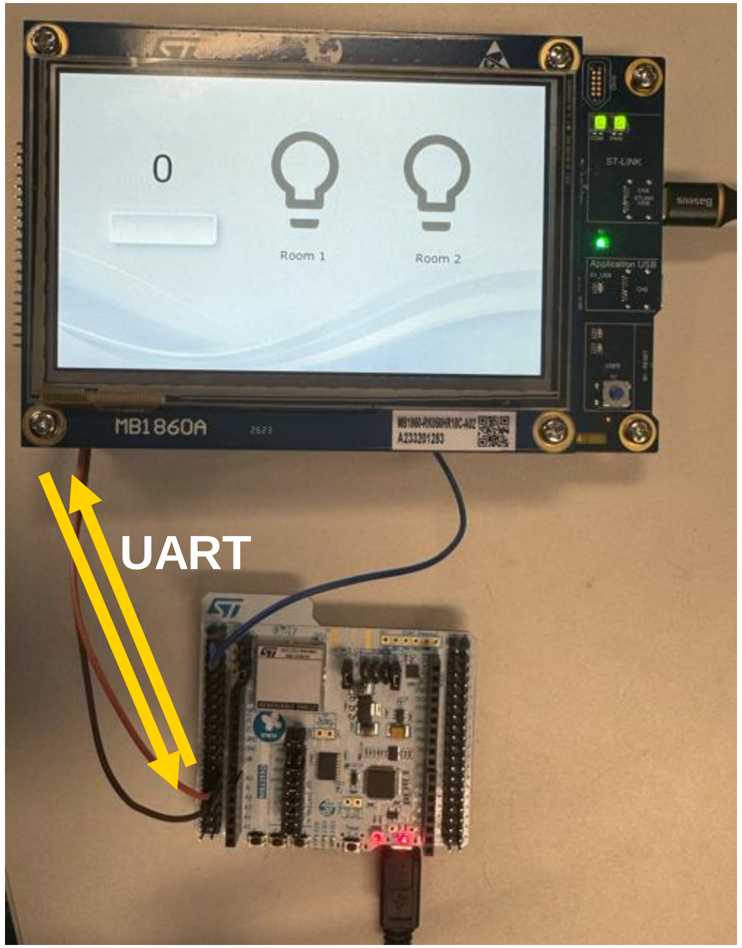
Missions' completion conditions

- Conditions
  - Submit each mission deliverable
    - Mission 1 : zip file with the GUI simulation
    - Mission 2 : video recording the expected behavior between the WB55 and the smartphone.
    - Mission 3 : video recording the expected behavior between the WB55, the U5G9 and the smartphone.

- Note :
  - You are expected to reuse projects created in mission 1 and 2 and add the UART communication between the 2 boards.

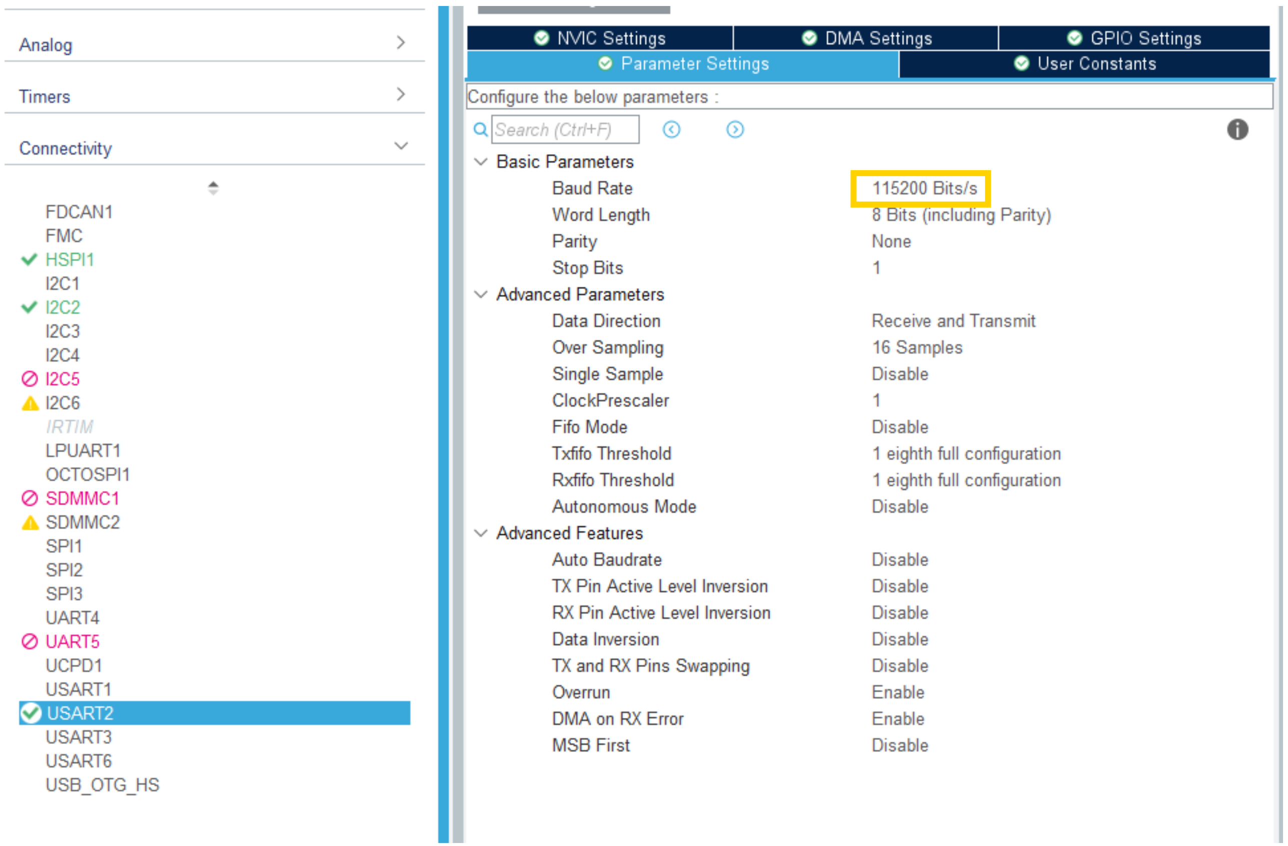
Expected result

- Necessary hardware
  - 1x STM32U5G9J-DK2
  - 1x STM32WB55-NUCLEO
  - 3x male-to-female cables (UART RX/TX, GND)
  - 1x USB-C cable to power and flash the STM32U5G9J-DK2
  - 1x USB micro B cable to power and flash the STM32WB55-NUCLEO



life.augmented

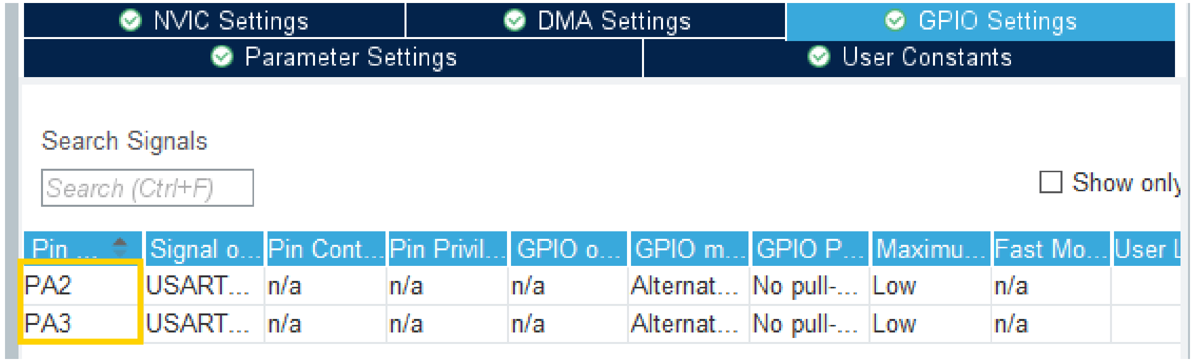
UART implementation on U5G9
CubeMX UART configuration

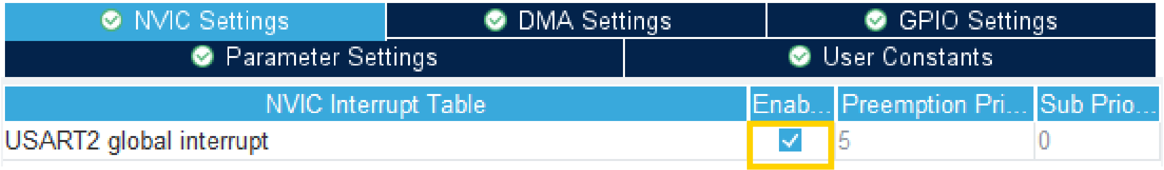


- Enable USART2
- Make sure the parameters settings are the same as on the left picture.
  - Especially, the baud rate must be 115200

life.augmented

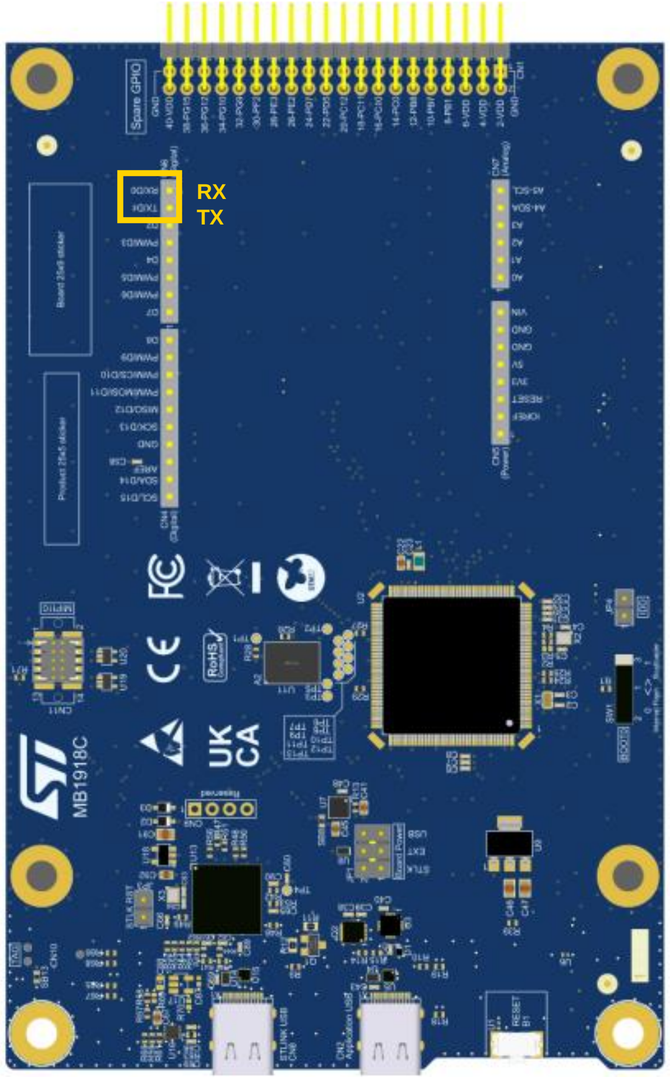
UART implementation on U5G9
CubeMX UART configuration





- Check that the pins used for the USART are PA2 for TX and PA3 for RX
- Enable the interrupt of the USART2 to be able to receive the data from the STM2WB55 on interrupt.
- Then, generate code.

UART implementation on U5G9
CubeMX UART configuration



- Why use USART2 with pin PA2 and PA3 ?
  - Because we want to easily connect the U5G9J-DK2 with the WB55-Nucleo board.
    For this, using the Arduino connectors is the easiest way.

life.augmented

UART implementation on U5G9
Declare a queue in app_freertos.c

app_freertos.c ×

Core > Src > app_freertos.c > ...

```c
/* Private variables ------------
/* USER CODE BEGIN Variables */
QueueHandle_t msgQueueUARTtoUI;
/* USER CODE END Variables */
```

app_freertos.c ×

Core > Src > app_freertos.c > defaultTask_attributes

```c
void MX_FREERTOS_Init(void) {

  /* USER CODE BEGIN RTOS_QUEUES */
  msgQueueUARTtoUI = xQueueCreate(1, sizeof(uint8_t));
  /* USER CODE END RTOS_QUEUES */
  /* creation of defaultTask */
  defaultTaskHandle = osThreadNew(StartDefaultTask, NULL, &defaultTask_attributes);

  /* creation of GUI_Task */
  GUI_TaskHandle = osThreadNew(TouchGFX_Task, NULL, &GUI_Task_attributes);
```

- In order to provide data to the TouchGFX task when running FreeRTOS, we need to create a queue.
  - First, we declare the queue.
  - Second, we initialize it to only store one byte.

UART implementation on U5G9 Receive/Send UART data from main.c

main.c 9+

Core > Src > main.c > MX_GPIO_Init(void)

```c
#include "FreeRTOS.h" // need to be included before queue.h
#include "queue.h" //For queues in FreeRTOS
/* USER CODE END Includes */
```

main.c 9+

Core > Src > main.c > ...

```c
/* USER CODE BEGIN PV */
static char pDataRx[2]; //Buffer used for receiving data from computer

extern QueueHandle_t msgQueueUARTtoUI;
/* USER CODE END PV */
```

main.c 9+

Core > Src > main.c > MX_DCACHE1_Init(void)

```c
int main(void)
  MX_USART2_UART_Init();
  MX_TouchGFX_Init();
  /* Call PreOsInit function */
  MX_TouchGFX_PreOSInit();
  /* USER CODE BEGIN 2 */
  HAL_UART_Receive_IT(&huart2, (uint8_t *)pDataRx, 1);
  /* USER CODE END 2 */
```

main.c 9+

Core > Src > main.c > ...

```c
/* USER CODE BEGIN 4 */
void Send_UART_Message(uint8_t *buf, uint8_t size)
{
  HAL_UART_Transmit(&huart2, (uint8_t *)buf, size, 5000);
}

void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart)
{
    //TODO: checks if the value received is correct or not
    xQueueSendFromISR(msgQueueUARTtoUI, &pDataRx[0], 0);

    HAL_UART_Receive_IT(&huart2, (uint8_t *)pDataRx, 1);
}
/* USER CODE END 4 */
```

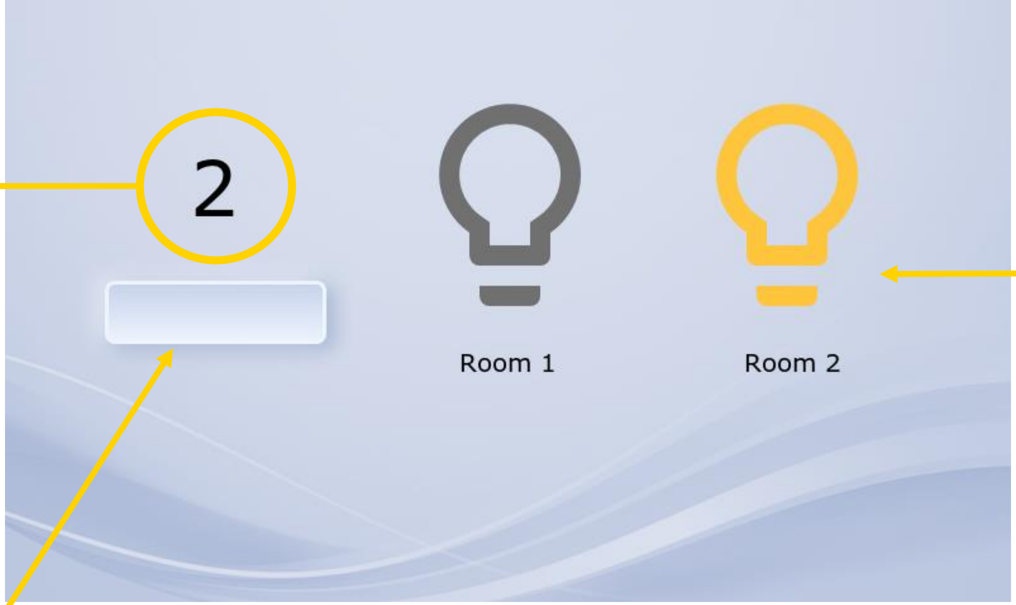
- Now that the queue is created, we need to use it when we receive data through UART
  1. First, we need to include some FreeRTOS files
  2. Then, we declare the queue and a buffer in main.c
  3. Enable the interruption trigger in the init() function
  4. Create the data reception callback.
- In order to send data to the WB55, we also create a function.

life.augmented

UART implementation on U5G9
GUI creation to use the UART feature – TouchGFX Designer

- Now, we need to create the GUI and the backend communication. The GUI should resemble the picture below.

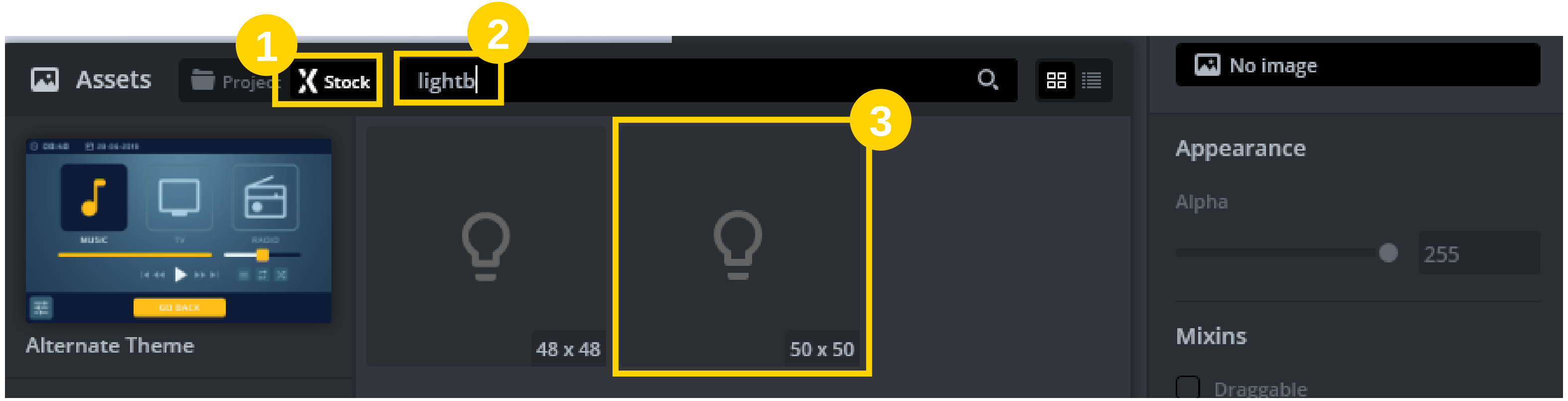
This number will correspond the last number received from the WB55 (see table on the right for more details)



Command	LIGHT1	LIGHT2
0x00	OFF	OFF
0x01	ON	OFF
0x02	OFF	ON
0x03	ON	ON

Pressing this button will send the current status value to the WB55, in this example it will be 2.

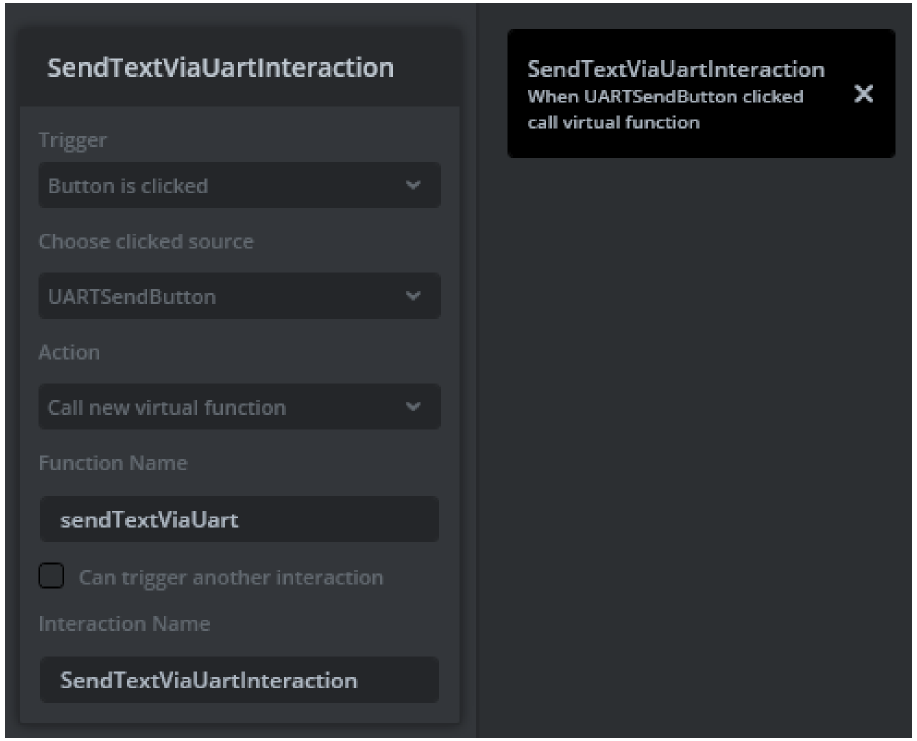
UART implementation on U5G9
GUI creation to use the UART feature - TouchGFX Designer



- You learned how to include a text and an image to a GUI during mission 1.
- Here, you will learn how to create an image based on a Google icon in SVG format. To do this, follow these steps :
  1. Create an image widget and in the image selector interface, click on “Stock”.
  2. Type “lightb” in the search bar
  3. Select the image in size 50x50
  4. Finally, create 2 lightbulb images in size 200x200 : one will have a value color of 0xFFC53D and the other will have 0x707070 as the color value. (see previous slide for reference)

life.augmented

UART implementation on U5G9
GUI creation to use the UART feature - TouchGFX Designer



- Create an interaction for the button to send data when pressed.

UART implementation on U5G9
GUI creation to use the UART feature - Code

MainView.hpp

TouchGFX > gui > include > gui > main_screen > MainView.hpp

```cpp
class MainView : public MainViewBase

    virtual void sendTextViaUart();
```

MainView.cpp

TouchGFX > gui > src > main_screen > MainView.cpp > setNewValue(uint8_t)

```cpp
void MainView::sendTextViaUart()
{
    // button is clicked, send text via UART
    uint8_t mes[2] = {(uint8_t)lightbulbs_status, 0x0};
    presenter->sendText(mes, 1);
}
```

MainPresenter.hpp

TouchGFX > gui > include > gui > main_screen > MainPresenter.hpp >

```cpp
class MainPresenter : public touchgfx::Presenter,

    void sendText(uint8_t* text, uint8_t size)
    {
        model->sendText(text, size);
    }
```

Model.hpp

TouchGFX > gui > include > gui > model > Model.hpp > Model >

```cpp
class Model

    void sendText(uint8_t* text, uint8_t size);
```

Model.cpp

TouchGFX > gui > src > model > Model.cpp > Model()

```cpp
void Model::sendText(uint8_t* text, uint8_t size)
{
    Send_UART_Message(text, size);
}
```

- Using the MVP design pattern, after button has been pressed, we send data by propagating the information from View → Presenter → Model → Backend

life.augmented

UART implementation on U5G9
GUI creation to use the UART feature - Code

Model.hpp ×

TouchGFX > gui > include > gui > model > Model.hpp

```cpp
class Model
protected:
    ModelListener* modelListener;

    uint8_t newValue;
};
```

Model.cpp ×

TouchGFX > gui > src > model > Model.cpp > Model()

```cpp
extern "C"
{
    #include "FreeRTOS.h"
    #include "queue.h"
    extern QueueHandle_t msgQueueUARTtoUI;

    void Send_UART_Message(uint8_t *buf, uint8_t size);
}

Model::Model() :
    modelListener(0),
    newValue(0)
{

}

void Model::tick()
{
    if (uxQueueMessagesWaiting(msgQueueUARTtoUI) > 0)
    {
        xQueueReceive(msgQueueUARTtoUI, &newValue, 0);

        if(modelListener != 0)
        {
            modelListener->setNewValue(newValue);
        }
    }
}
```

- This code is used to receive the value available in the queue and send it the View.

life.augmented

UART implementation on U5G9
GUI creation to use the UART feature - Code

ModelListener.hpp

TouchGFX > gui > include > gui > model > ModelListener.hpp > ...

```cpp
class ModelListener

    virtual void setNewValue(uint8_t value) {}
protected:
```

MainPresenter.hpp

TouchGFX > gui > include > gui > main_screen > MainPresenter.h

```cpp
class MainPresenter : public touchgfx::Presen
    }

    virtual void setNewValue(uint8_t value);
```

MainPresenter.cpp

TouchGFX > gui > src > main_screen > MainPresenter.cpp > activate()

```cpp
void MainPresenter::setNewValue(uint8_t value)
{
    view.setNewValue(value);
}
```

MainView.hpp

TouchGFX > gui > include > gui > main_screen > MainV

```cpp
class MainView : public MainViewBase

    void setNewValue(uint8_t value);
```

- Once again, we use the [MVP design pattern](), after receiving data from the queue and we propagate the information in the following manner :
  Model → ModelListener → Presenter → View

life.augmented

UART implementation on U5G9
GUI creation to use the UART feature - Code

```
void MainView::setNewValue(uint8_t value)
{
    if(value <= LIGHTBULBS_ON) // verify the number received is correct
    {
        lightbulbs_status = value;
        // Update textArea according to the new value
        Unicode::snprintf(DataRXTextAreaBuffer, DATARXTEXTAREA_SIZE, "%d", value);
        DataRXTextArea.invalidate();

        switch(lightbulbs_status)
        {
            case LIGHTBULBS_OFF:
                lightbulb1_Img.setBitmap(touchgfx::Bitmap(BITMAP_ICON_THEME_IMAGES_ACTION_LIGHTBULB_200_200_707070_SVG_ID));
                lightbulb2_Img.setBitmap(touchgfx::Bitmap(BITMAP_ICON_THEME_IMAGES_ACTION_LIGHTBULB_200_200_707070_SVG_ID));
                break;
            case LIGHTBULB1_ON:
                lightbulb1_Img.setBitmap(touchgfx::Bitmap(BITMAP_ICON_THEME_IMAGES_ACTION_LIGHTBULB_200_200_FFC53D_SVG_ID));
                lightbulb2_Img.setBitmap(touchgfx::Bitmap(BITMAP_ICON_THEME_IMAGES_ACTION_LIGHTBULB_200_200_707070_SVG_ID));
                break;
            case LIGHTBULB2_ON:
                lightbulb1_Img.setBitmap(touchgfx::Bitmap(BITMAP_ICON_THEME_IMAGES_ACTION_LIGHTBULB_200_200_707070_SVG_ID));
                lightbulb2_Img.setBitmap(touchgfx::Bitmap(BITMAP_ICON_THEME_IMAGES_ACTION_LIGHTBULB_200_200_FFC53D_SVG_ID));
                break;
            case LIGHTBULBS_ON:
                lightbulb1_Img.setBitmap(touchgfx::Bitmap(BITMAP_ICON_THEME_IMAGES_ACTION_LIGHTBULB_200_200_FFC53D_SVG_ID));
                lightbulb2_Img.setBitmap(touchgfx::Bitmap(BITMAP_ICON_THEME_IMAGES_ACTION_LIGHTBULB_200_200_FFC53D_SVG_ID));
                break;
            default:
                break;
        }
        lightbulb1_Img.invalidate();
        lightbulb2_Img.invalidate();
    }
}
```

- Here is one way to turn on/off the lightbulbs according to the data received through UART.
    - Don’t forget to declare the different lightbulb states in an enum declaration. And initialize the lightbulb_status variable.
    - Note : The application should start with both lightbulbs turned off.

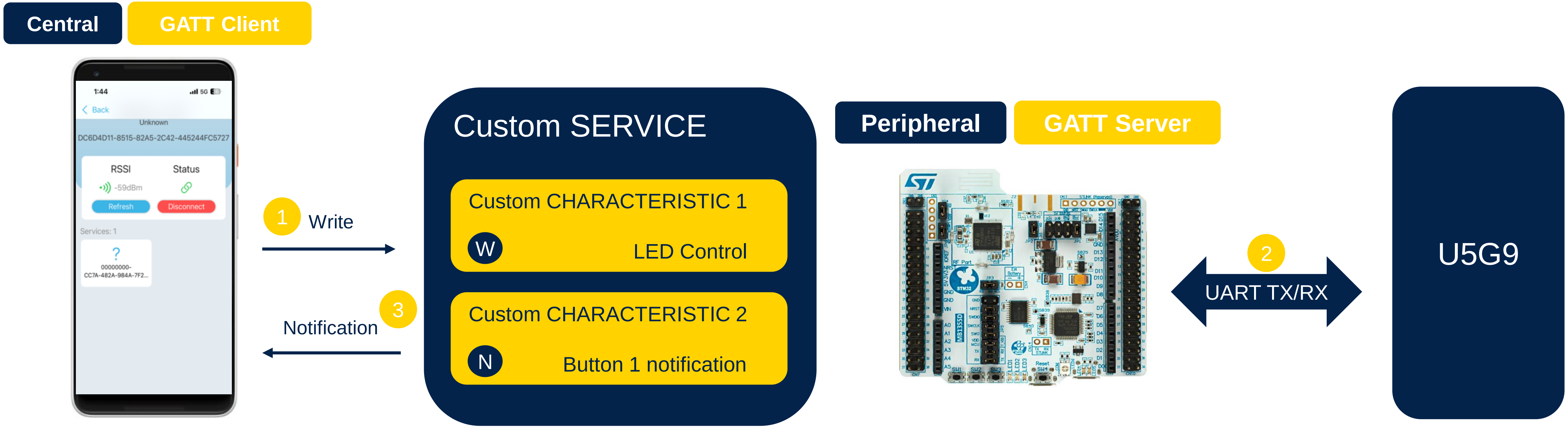
life.augmented

UART implementation on STM32WB55

life.augmented

Adding UART function
UART scenario

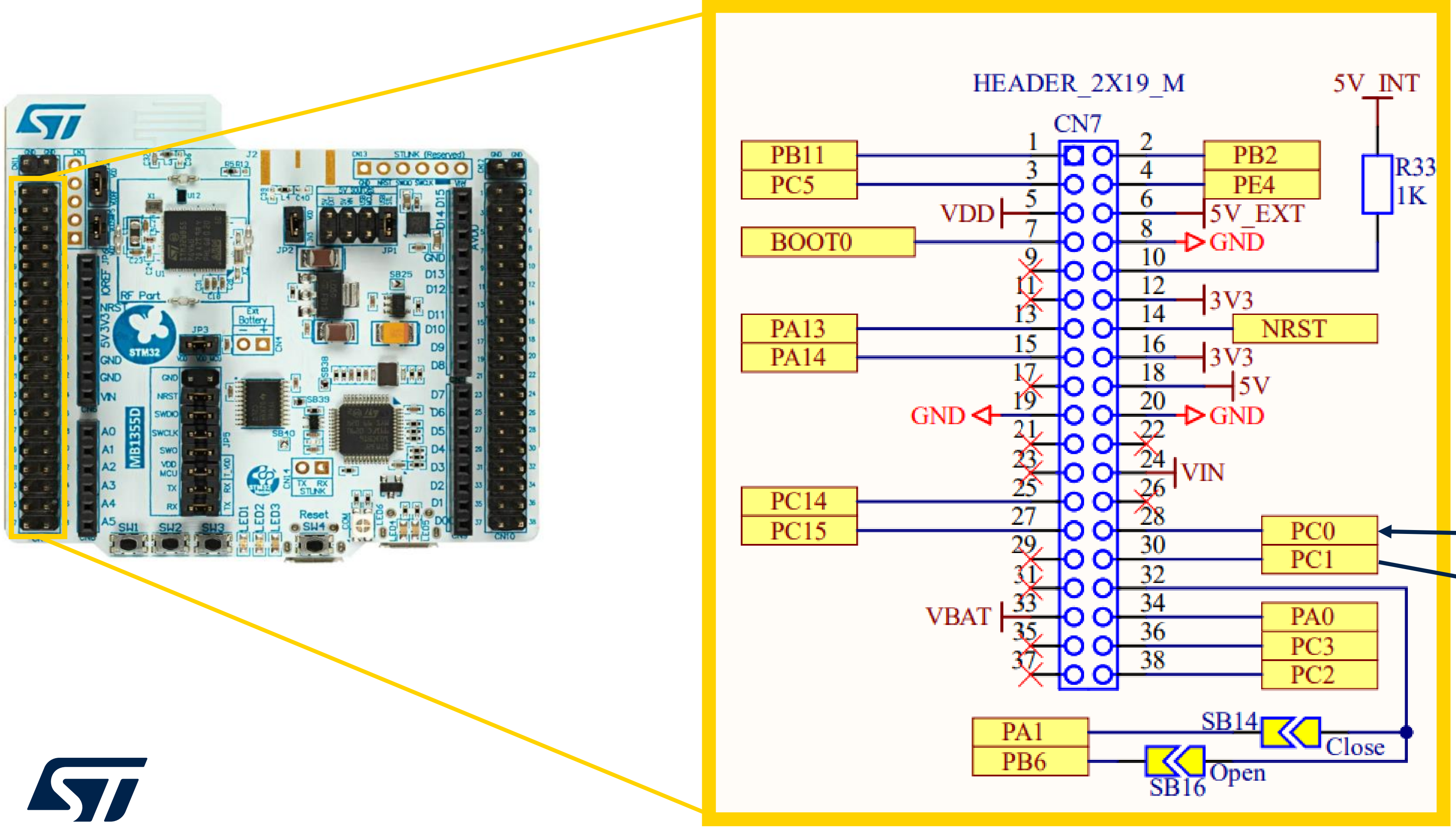
- If phone sends 1 byte write command to the device, it will be transmitted to the UART TX.
- If UART RX interrupt (1byte) occurs, the received data (1byte) will be transmitted to the phone.



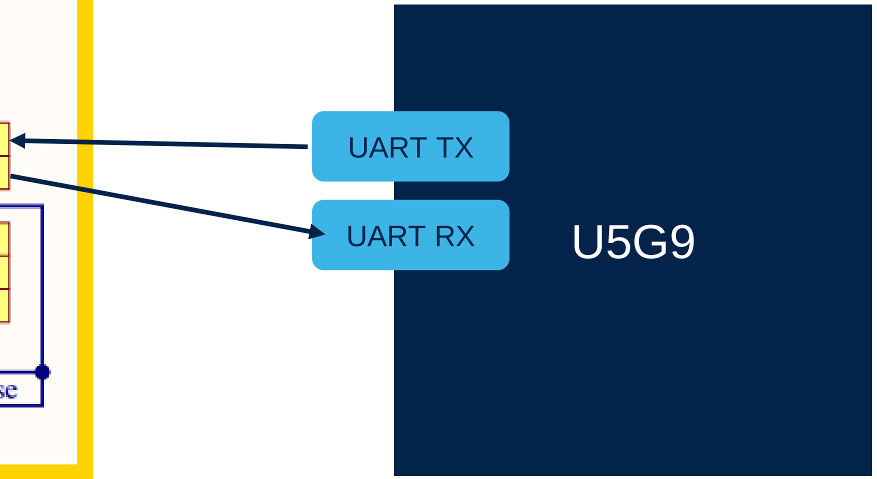
life.augmented

Adding UART function Step #1

1. Wire setting with U5G9 board

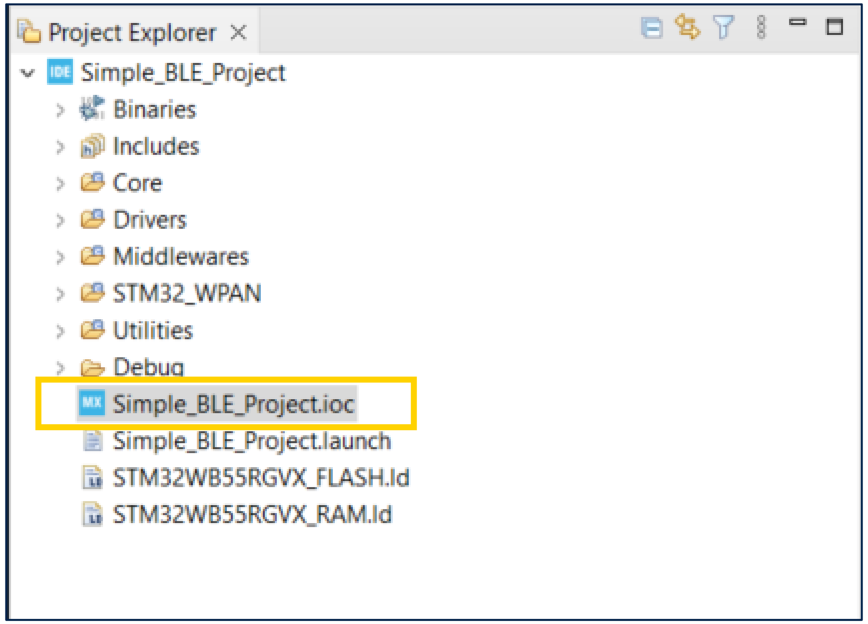


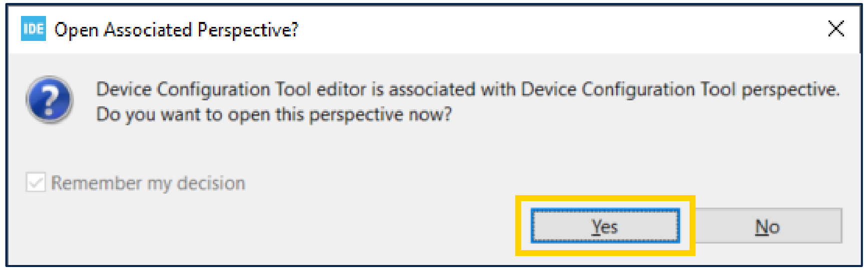
PC0 : UART RX
PC1 : UART TX



Adding UART function
Step #2

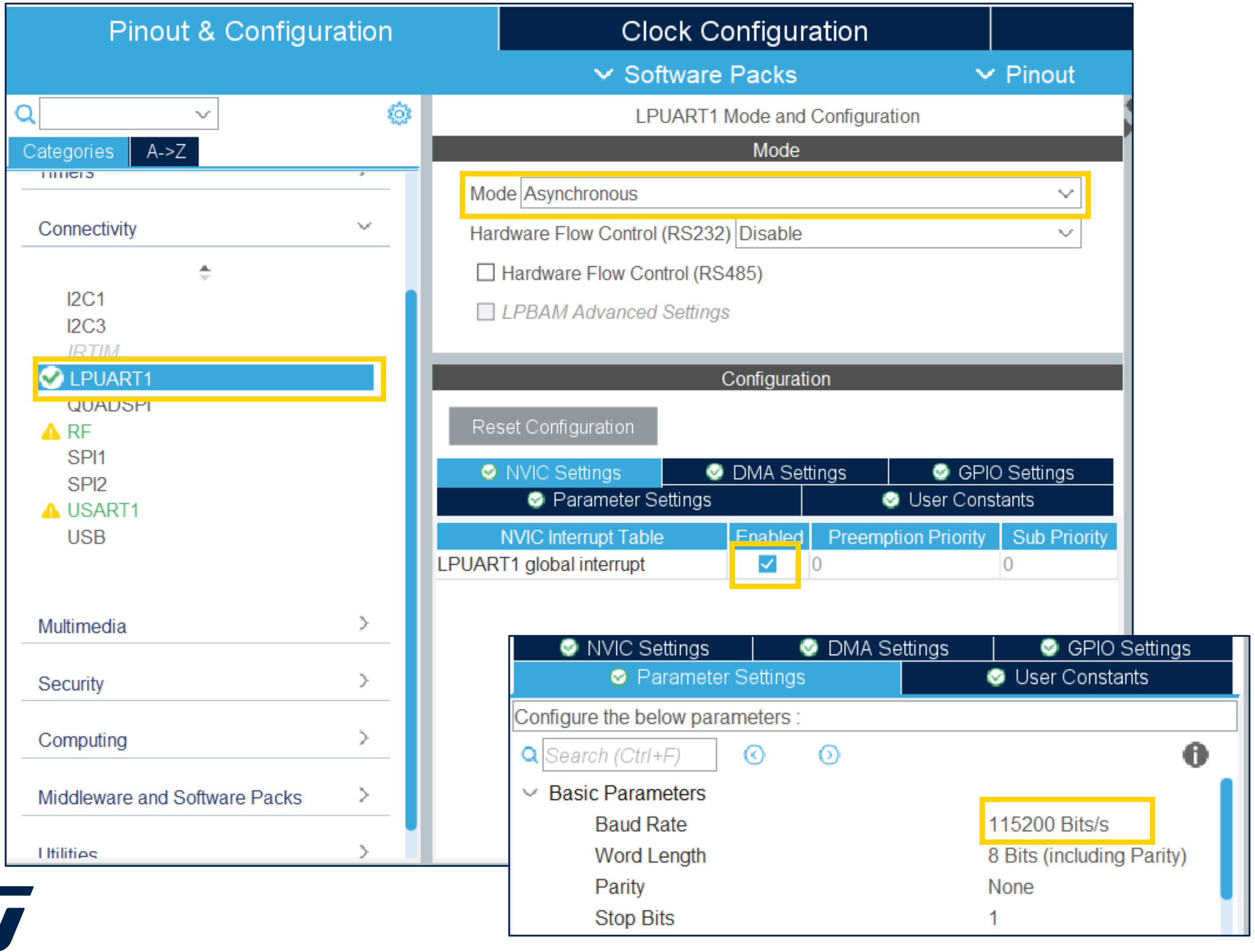
1. Open Simple_BLE_Project project by CubeIDE
2. Double click “Simple_BLE_Project.ioc” on CubeIDE





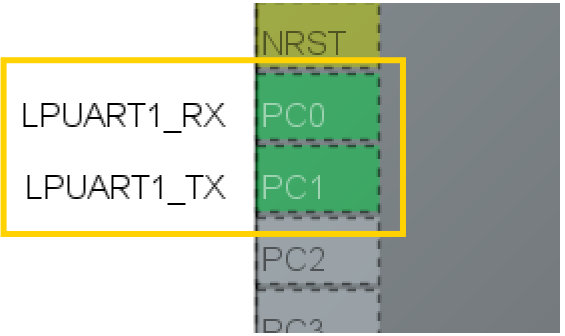
life.augmented

Adding UART function Step #3



1. Select LPUART1
2. Set Mode to Asynchronous
3. Enable LPUART1 global interrupt
4. Set Baud rate to 115200

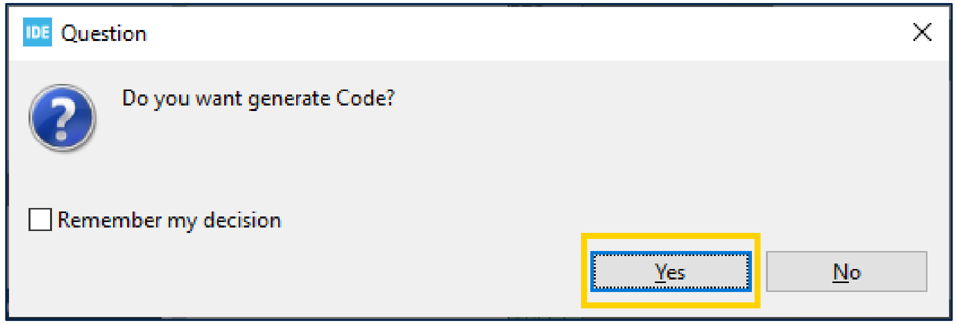
IO will be assigned as thisa

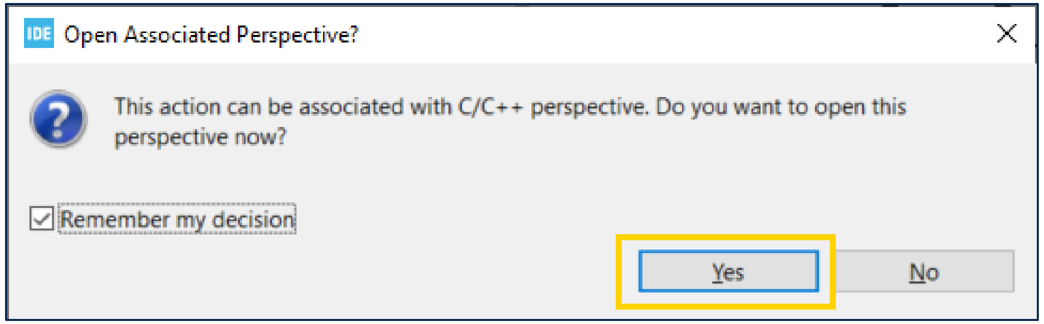


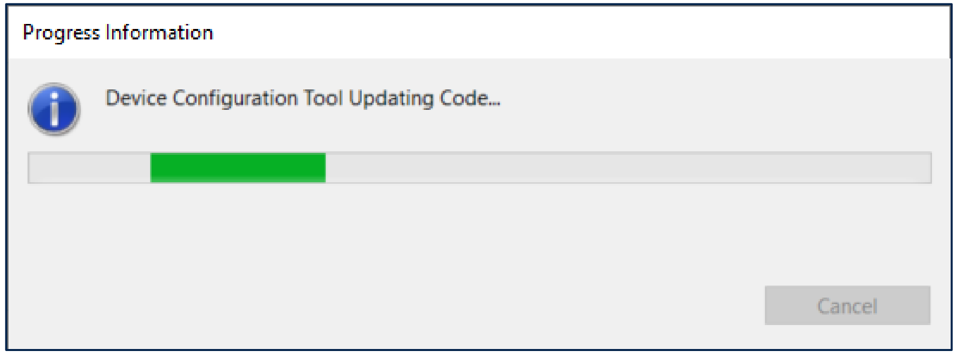
life.augmented

Adding UART function
Step #4

- Save all (Ctrl+Shift+S) 
- Code will be updated



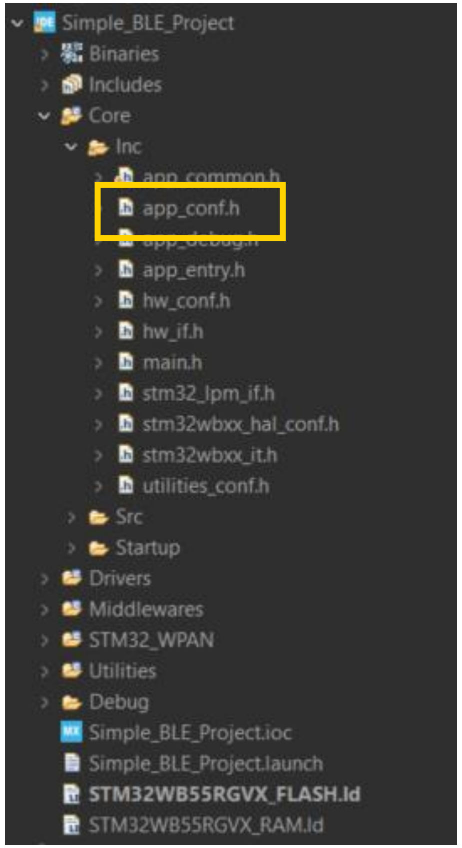




life.augmented

Adding UART function
Step #5

- Update app_conf.h as follows (location : Simple_BLE_Project/Core/Inc/)



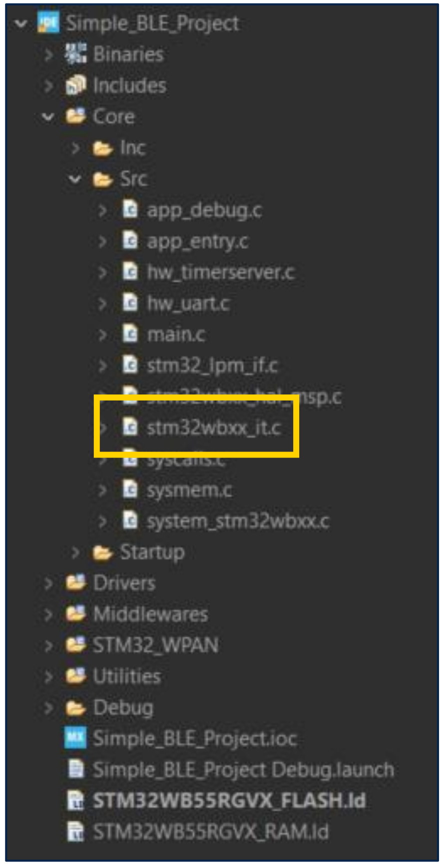

```
/* USER CODE BEGIN Defines */
void task_button1(void);

void task_uarttx(void);
void task_uartrx(void);
/* USER CODE END Defines */
```

```
/* USER CODE BEGIN CFG_Task_Id_With_HCI_Cmd_t */
CFG_TASK_BUTTON1_ID,

CFG_TASK_UARTTX_ID,
CFG_TASK_UARTRXCALLBACK_ID,
/* USER CODE END CFG_Task_Id_With_HCI_Cmd_t */
```

life.augmented

Adding UART function Step #6

- Update stm32wbxx_it.c as follows (location : Simple_BLE_Project/Core/Src/)

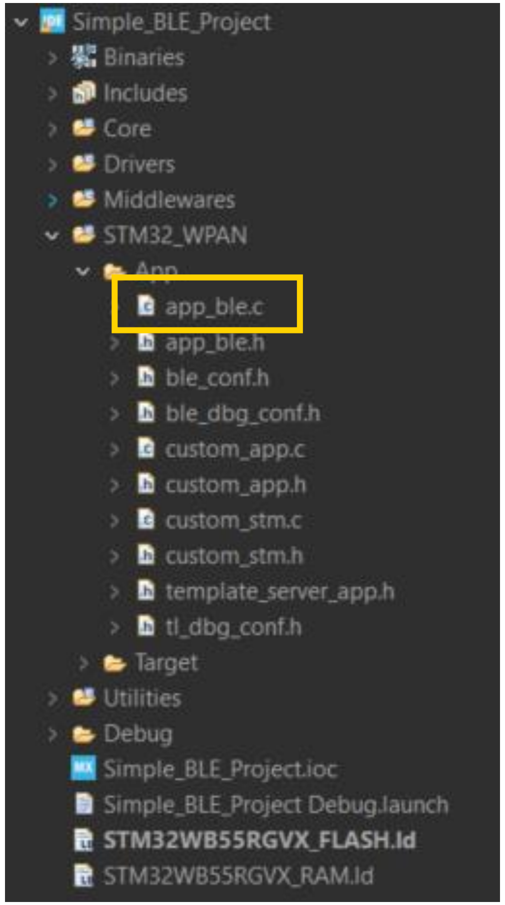



```c
/* USER CODE BEGIN Includes */
#include "custom_stm.h"
/* USER CODE END Includes */
```

```c
void LPUART1_IRQHandler(void)
{
  /* USER CODE BEGIN LPUART1_IRQn 0 */
  Uart_Rx_Callback();
  /* USER CODE END LPUART1_IRQn 0 */
  HAL_UART_IRQHandler(&hlpuart1);
  /* USER CODE BEGIN LPUART1_IRQn 1 */

  /* USER CODE END LPUART1_IRQn 1 */
}
```

Adding UART function
Step #7

- Update app_ble.c as follows
  (location : Simple_BLE_Project/STM32_WPAN/App/)

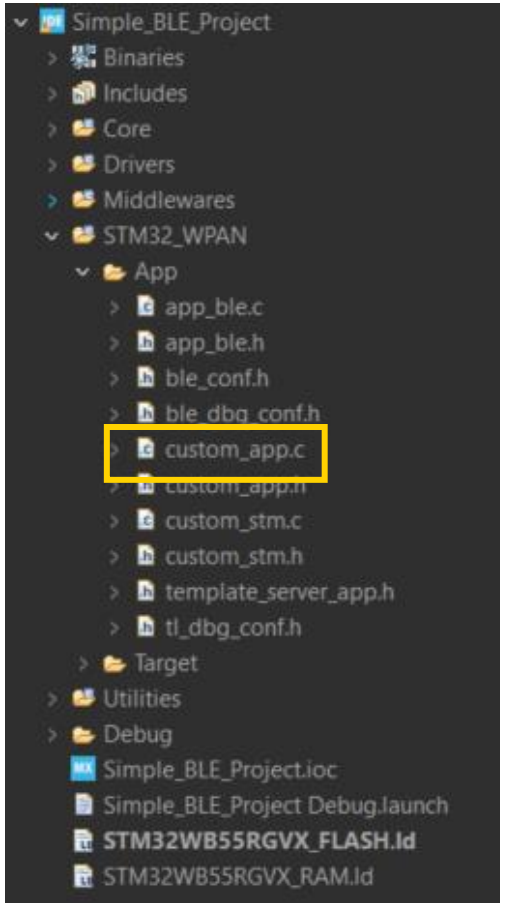



```
/* USER CODE BEGIN APP_BLE_Init_1 */
UTIL_SEQ_RegTask(1<<CFG_TASK_BUTTON1_ID, UTIL_SEQ_RFU, task_button1);
UTIL_SEQ_SetTask(1<<CFG_TASK_BUTTON1_ID, CFG_SCH_PRIO_0);

UTIL_SEQ_RegTask(1<<CFG_TASK_UARTTX_ID, UTIL_SEQ_RFU, task_uarttx);
UTIL_SEQ_RegTask(1<<CFG_TASK_UARTRXCALLBACK_ID, UTIL_SEQ_RFU, task_uartrx);
/* USER CODE END APP_BLE_Init_1 */
```


life.augmented

Adding UART function
Step #8

- Update custom_app.c as follows
(location : Simple_BLE_Project/STM32_WPAN/App/)



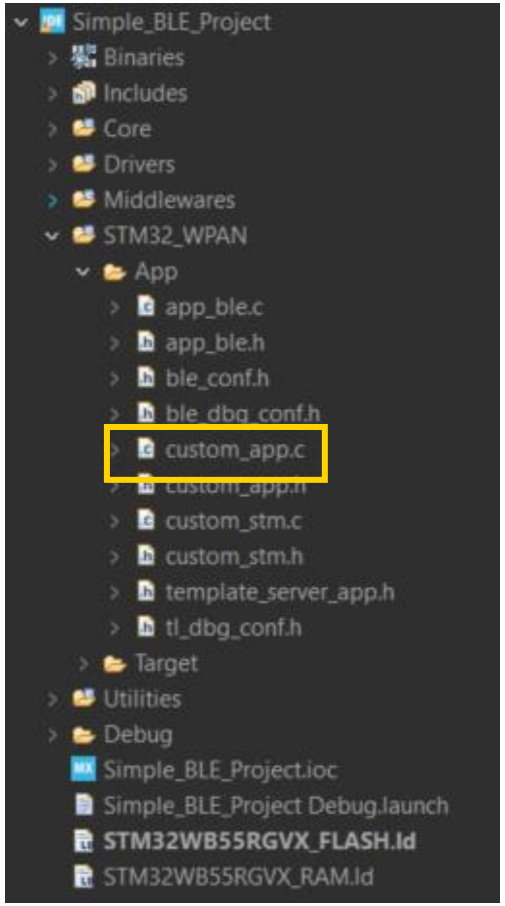

```c
/* USER CODE BEGIN PV */
uint8_t notidata = 0;
uint8_t notiflag = 0;
uint8_t keystate = 0;

extern uint8_t uart_rxdata;
extern uint8_t uart_txdata;
extern UART_HandleTypeDef hlpuart1;
/* USER CODE END PV */
```

```c
void task_uarttx(void)
{
APP_DBG_MSG("[UART] Write CMD --> task_uart tx --> U5G9 (0x%x)\n\r", uart_txdata);
HAL_UART_Transmit_IT(&hlpuart1, &uart_txdata, 1);
}

void task_uartrx(void)
{
APP_DBG_MSG("[UART] task_uart rx --> Notification --> Phone (0x%x)\n\r", uart_rxdata);
NotifyCharData[0] = uart_rxdata;
Custom_St_nchar_Send_Notification();
HAL_UART_Receive_IT(&hlpuart1, &uart_rxdata, 1);    //1byte interrupt
}
/* USER CODE END PFP */
```

life.augmented

Adding UART function
Step #9

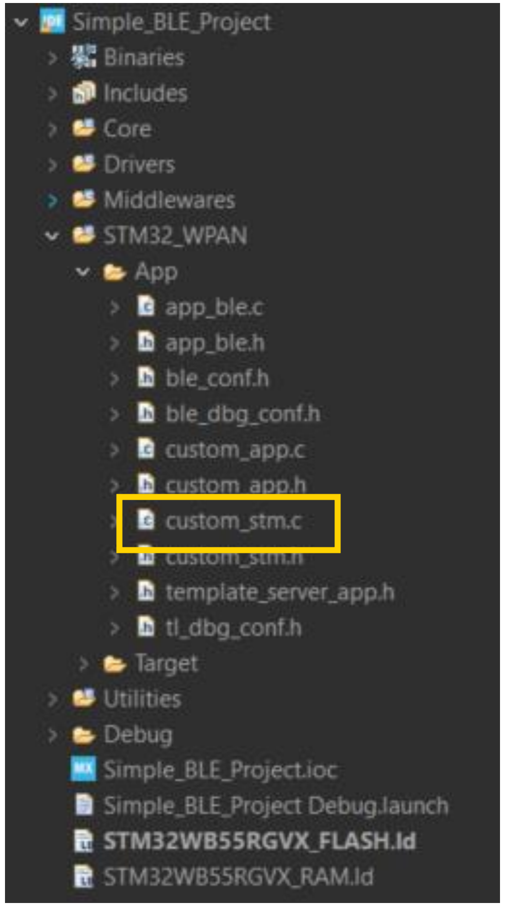
- Update custom_app.c as follows
  (location : Simple_BLE_Project/STM32_WPAN/App/)




```c
case CUSTOM_STM_ST_NCHAR_NOTIFY_ENABLED_EVT:
  /* USER CODE BEGIN CUSTOM_STM_ST_NCHAR_NOTIFY_ENABLED_EVT */
  APP_DBG_MSG("notification enabled!\n\r");
  notiflag = 1;

  HAL_UART_Receive_IT(&hlpuart1, &uart_rxdata, 1); //1byte interrupt
  /* USER CODE END CUSTOM_STM_ST_NCHAR_NOTIFY_ENABLED_EVT */
  break;
```

Adding UART function
Step #10

- Update custom_stm.c as follows
  (location : Simple_BLE_Project/STM32_WPAN/App/)




```c
/* USER CODE BEGIN Includes */
#include "main.h"
#include "stm32_seq.h"
/* USER CODE END Includes */
```

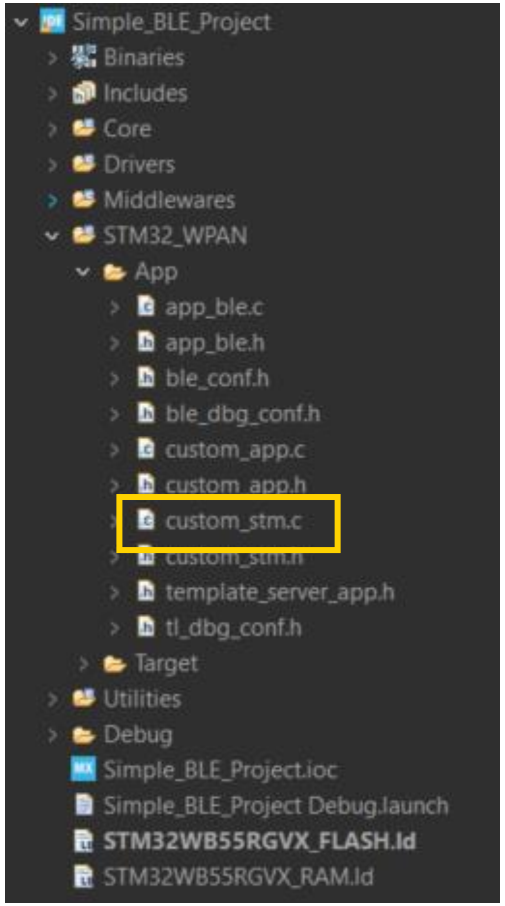
```c
/* USER CODE BEGIN PV */
extern UART_HandleTypeDef hlpuart1;
extern uint8_t NotifyCharData[247];

uint8_t uart_rxdata = 0x00;
uint8_t uart_txdata = 0x00;
/* USER CODE END PV */
```

```c
/* USER CODE BEGIN PF */
void Uart_Rx_Callback(void)
{
    UTIL_SEQ_SetTask(1<<CFG_TASK_UARTRXCALLBACK_ID, CFG_SCH_PRIO_0);
}
/* USER CODE END PF */
```

life.augmented

Adding UART function
Step #11

- Update custom_stm.c as follows
  (location : Simple_BLE_Project/STM32_WPAN/App/)

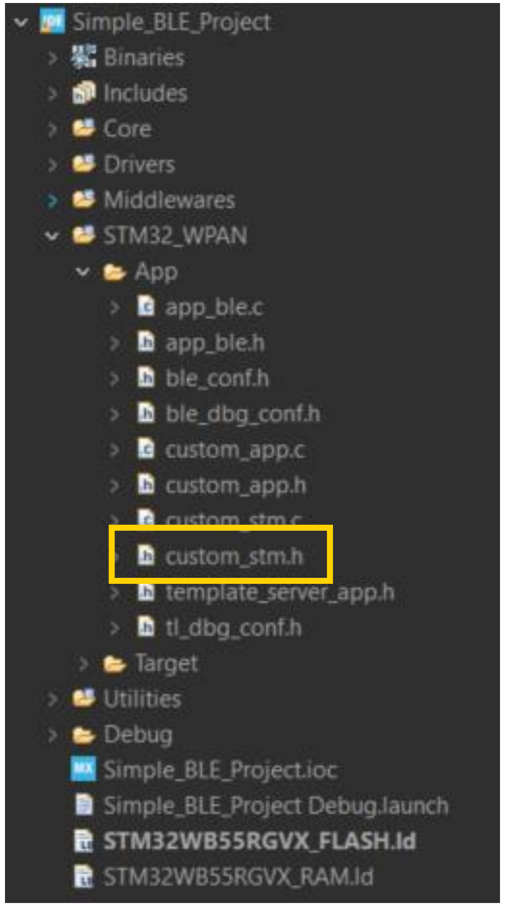



```c
else if (attribute_modified->Attr_Handle == (CustomContext.CustomSt_WcharHdle + CHARACTERISTIC_VALUE_ATTRIBUTE_OFFSET))
{
  return_value = SVCCTL_EvtAckFlowEnable;
  /* USER CODE BEGIN CUSTOM_STM_Service_1_Char_1_ACI_GATT_ATTRIBUTE_MODIFIED_VSEVT_CODE */
  APP_DBG_MSG(" Write_Cmd : 0x%02x \n\r", attribute_modified->Attr_Data[0]);
  if(attribute_modified->Attr_Data[0] == 0x00)
  {
    HAL_GPIO_WritePin(LD2_GPIO_Port, LD2_Pin, GPIO_PIN_RESET);
  }
  else if(attribute_modified->Attr_Data[0] == 0x01)
  {
    HAL_GPIO_WritePin(LD2_GPIO_Port, LD2_Pin, GPIO_PIN_SET);
  }

  if(attribute_modified->Attr_Data_Length != 0x00)
  {
    //Transmit write data through uart tx
    uart_txdata = attribute_modified->Attr_Data[0];
    UTIL_SEQ_SetTask(1<<CFG_TASK_UARTTX_ID, CFG_SCH_PRIO_0);
  }

  /* USER CODE END CUSTOM_STM_Service_1_Char_1_ACI_GATT_ATTRIBUTE_MODIFIED_VSEVT_CODE */
```

life.augmented

Adding UART function
Step #12

- Update custom_stm.h as follows
  (location : Simple_BLE_Project/STM32_WPAN/App/)

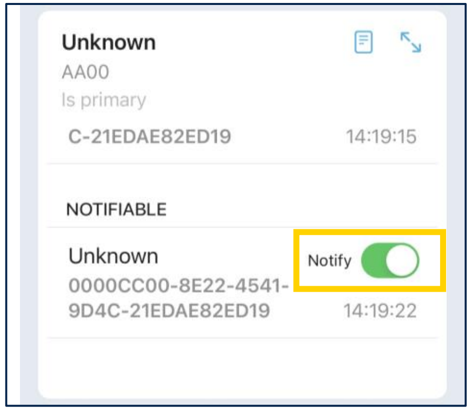



```c
/* Includes ------------------------------------------------------------------*/
/* USER CODE BEGIN Includes */
#include "ble_types.h"
/* USER CODE END Includes */
```

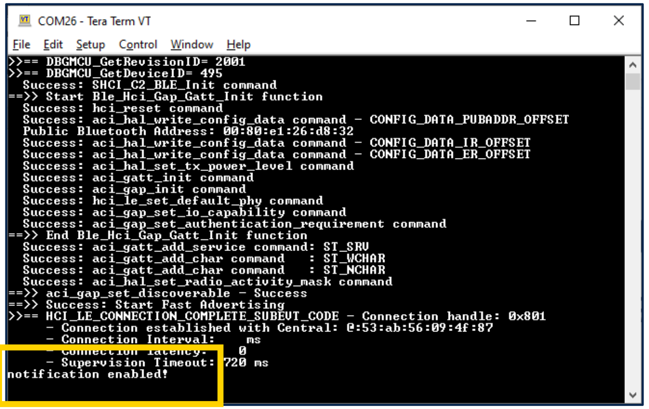
```c
/* USER CODE BEGIN EF */
void Uart_Rx_Callback(void);
/* USER CODE END EF */
```

Adding UART loopback Step #13

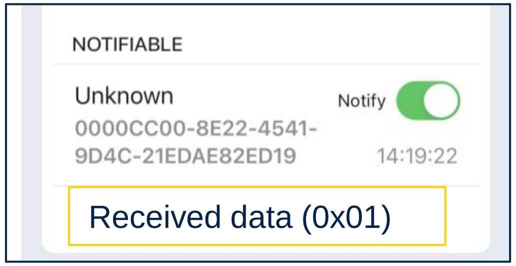
1. Build Project.
2. Programming new firmware again.
3. Connect your device through BLE
4. Enable notify by phone
5. Write 0x01.
6. Checking the received data.



Enable Notify



UART

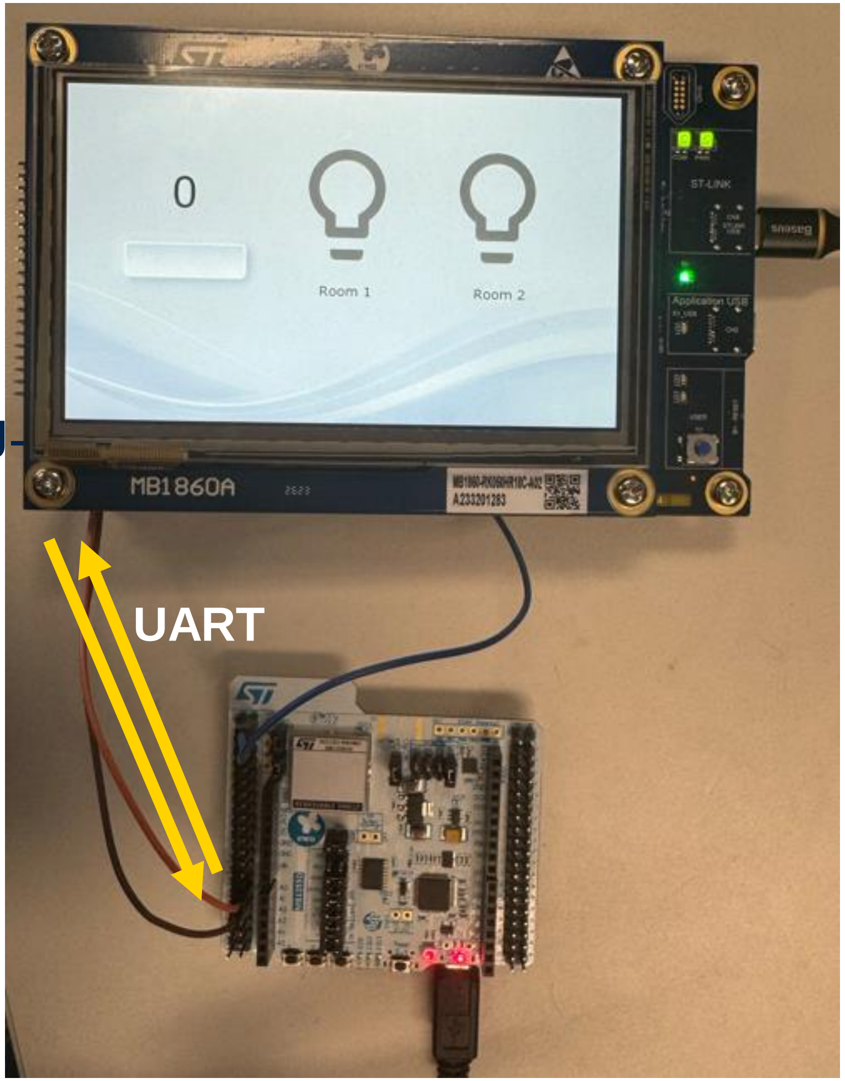


UART

life.augmented

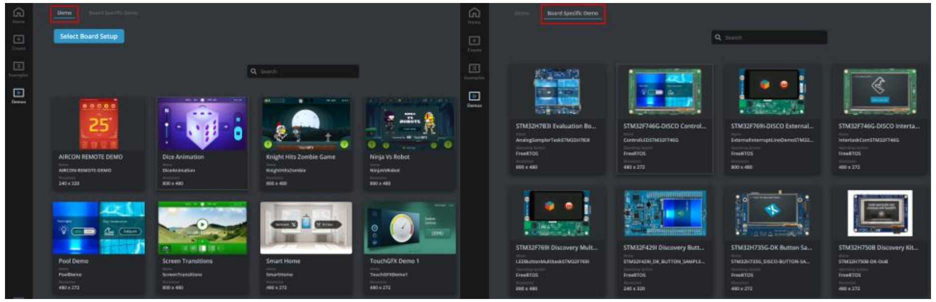
Expected result

- Necessary hardware
  - 1x STM32U5G9J-DK2
  - 1x STM32WB55-NUCLEO
  - 3x male-to-female cables (UART RX/TX, GND)
  - 1x USB-C cable to power and flash the STM32U5G9J-DK2
  - 1x USB micro B cable to power and flash the STM32WB55-NUCLEO

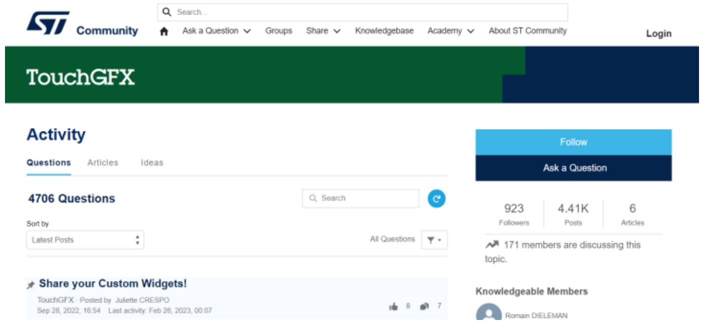


Useful resources for Graphics

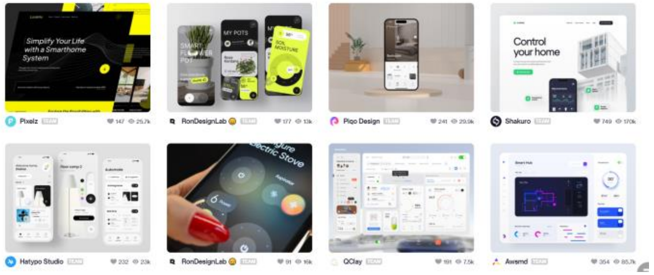
- Demos in TouchGFX Designer
- [TouchGFX Technical videos YouTube playlist]
- [Videos in Korean YouTube playlist]
- [ST Community]
- [TouchGFX documentation]
- [Dribble] for UI inspiration
- [st.com] for information on ST products
- Free image creation tool : [Paint.NET]



Demos in TouchGFX Designer



TouchGFX – ST Community



Dribble website

life.augmented

Useful resources for Wireless

- STM32WB Wiki page :
  https://wiki.st.com/stm32mcu/wiki/Category:STM32WB_Series

- STM32WB Online training:

  https://www.st.com/content/st_com/en/support/learning/stm32-education/stm32-online-training/stm32wb-online-training.html

- STM32WB Online training video session (YouTube)

  https://www.youtube.com/playlist?list=PLnMKNibPkDnGkMxFkRArr9uOq_Es_a7vG

life.augmented

Our technology starts with You

Find out more at www.st.com

life.augmented