Topic:	BLE-Module
Author:	JOKI
Date:	30.08.2021

0. Before you start

This document will give you an overview of the source code for the example package BLE-Module. Before you can work with it you need to set up your working environment as explained in the document “ExamplePackage_GettingStarted”. Make sure you have read this document beforehand and executed all the steps to configure your Workbench. Especially, that you have added all the paths to the *Source Location* and *Includes*.

For a more detailed explanation of how the microcontroller works, please refer to the STM32F429 Reference Manual (RM0090) and the Standard Peripheral Library documentation provided by STMicroelectronics.

1. Introduction

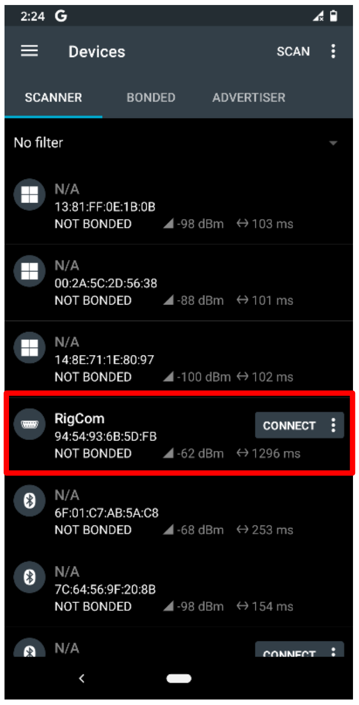
In this example package we will put the BLE-module, which is located on the 4.3- inch and 5.0-inch display boards, in operation. The used module is the BMD-300 manufactured by Rigado. The example application will let you send strings via Bluetooth to the display board. The string will then be displayed. You can also change color and position of the next string by using specific commands.
For the connecting and sending/receiving data via Bluetooth you need the software *nRF Connect* by NordicSemiconductor, which is available for desktop (Windows, Linux or macOS) and for your smartphone.

1.1 Using *nRF Connect for Mobile* and configuring the BMD-300

In this section you will learn, how to use the app *nRF Connect for Mobil* which will be needed for this example. Before the example code works, you need to configure the BMD-300 module, which is also done via the app.

Install the app from your App Store and open it. At the beginning you will need to scan for the available Bluetooth devices and search for the BMD-300. Obviously, the display board has to be connected to its power.

By default, the BMD-300 device is called “RigCom”. You can change this name later. Click *Connect*.



N/A
13:81:FF:0E:1B:0B
NOT BONDED -98 dBm ↔ 103 ms

N/A
00:2A:5C:2D:56:38
NOT BONDED -88 dBm ↔ 101 ms

N/A
14:8E:71:1E:80:97
NOT BONDED -100 dBm ↔ 102 ms

RigCom
94:54:93:6B:5D:FB
NOT BONDED -62 dBm ↔ 1296 ms

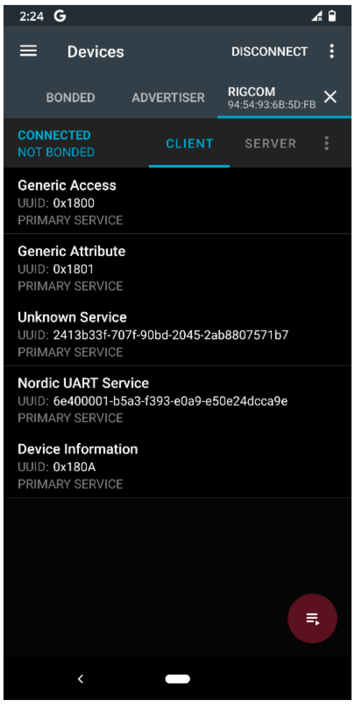
N/A
6F:01:C7:AB:5A:C8
NOT BONDED -68 dBm ↔ 253 ms

N/A
7C:64:56:9F:20:8B
NOT BONDED -98 dBm ↔ 154 ms

N/A

Screenshot 1

You should see this tab when successfully connected to the BMD-300.


2:24 G

Devices DISCONNECT

BONDED ADVERTISER RIGCOM 94:54:93:6B:5D:FB

CONNECTED NOT BONDED CLIENT SERVER

Generic Access
UUID: 0x1800
PRIMARY SERVICE

Generic Attribute
UUID: 0x1801
PRIMARY SERVICE

Unknown Service
UUID: 2413b33f-707f-90bd-2045-2ab8807571b7
PRIMARY SERVICE

Nordic UART Service
UUID: 6e400001-b5a3-f393-e0a9-e50e24dcca9e
PRIMARY SERVICE

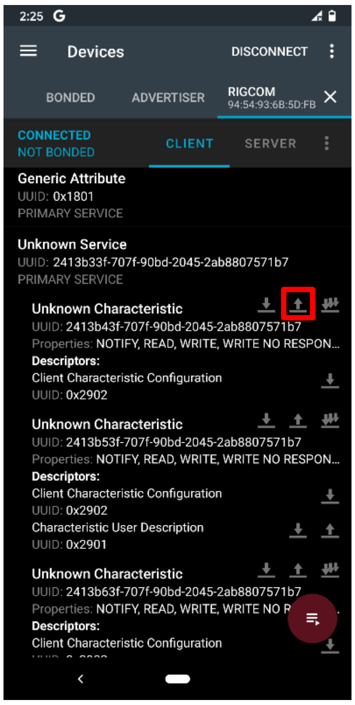
Device Information
UUID: 0x180A
PRIMARY SERVICE

Screenshot 2

We will at first change the name of our Bluetooth device. For this you need to click on *Unknown Service*. A submenu will show. Here you must change the *Unknown Characteristic* with the UUID:

2413B43F-707F-90BD-2045-2AB8807571B7

Click on the upload-button.


2:25 G

Devices DISCONNECT

BONDED ADVERTISER RIGCOM 94:54:93:6B:5D:FB

CONNECTED NOT BONDED CLIENT SERVER

Generic Attribute
UUID: 0x1801
PRIMARY SERVICE

Unknown Service
UUID: 2413b33f-707f-90bd-2045-2ab8807571b7
PRIMARY SERVICE

Unknown Characteristic
UUID: 2413b43f-707f-90bd-2045-2ab8807571b7
Properties: NOTIFY, READ, WRITE, WRITE NO RESPON...
Descriptors:
Client Characteristic Configuration
UUID: 0x2902

Unknown Characteristic
UUID: 2413b53f-707f-90bd-2045-2ab8807571b7
Properties: NOTIFY, READ, WRITE, WRITE NO RESPON...
Descriptors:
Client Characteristic Configuration
UUID: 0x2902
Characteristic User Description
UUID: 0x2901

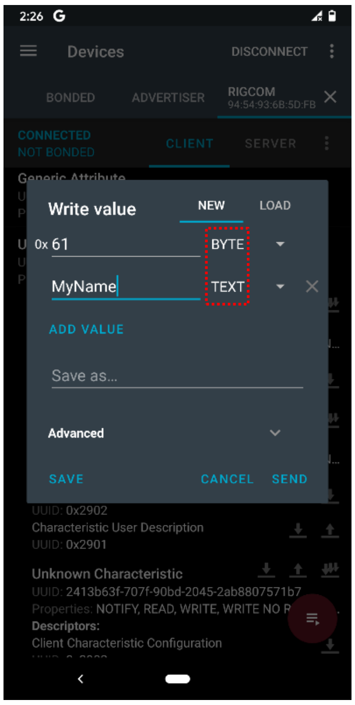
Unknown Characteristic
UUID: 2413b63f-707f-90bd-2045-2ab8807571b7
Properties: NOTIFY, READ, WRITE, WRITE NO R
Descriptors:
Client Characteristic Configuration

Screenshot 3

Example Package: BLE-Module_V200
Author: JOKI / 30.08.2021

A new window will pop up and ask you for the value, you want to write. You need to send two values. Therefore, you need to click *ADD VALUE* once. The first one is a byte with the value 0x61. This tells the BMD300 that the next parameter will be the device name. As the second value you need to choose text as type and type in your own device name. The device name must contain 1 to 8 ASCII characters. Then transmit your values by pressing *SEND*. After disconnecting and scanning for the available Bluetooth devices, you should find the BMD-300 module with your chosen name.

Make sure, that you select the correct types of the values. The first one is a BYTE and the second one a TEXT.



Write value

0x 61 BYTE

MyName TEXT

UUID: 0x2902
Characteristic User Description
UUID: 0x2901

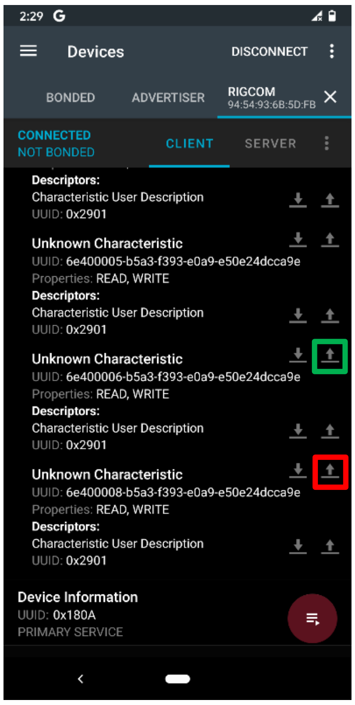
Unknown Characteristic
UUID: 2413b63f-707f-90bd-2045-2ab8807571b7
Properties: NOTIFY, READ, WRITE, WRITE NO R
Descriptors:
Client Characteristic Configuration

Screenshot 4

Next, we need to change some configurations, so that the BMD300 can communicate properly with the STM32-microcontroller. You will have to repeat these steps on every reconnect with the BLE-Modul

At first, we will enable the *pass-through mode*. This means, that the BMD300 will transmit all received data via UART to the connected STM32. For enabling this you need to open the submenu of *Nordic UART Service* (see Screenshot 2). Go to the end of the list and change the value of the characteristic with the UUID:

6E400008-B5A3-F393-E0A9-E50E24DCCA9E
(see red square in Screenshot 5)



Descriptors:
Characteristic User Description
UUID: 0x2901

Unknown Characteristic
UUID: 6e400005-b5a3-f393-e0a9-e50e24dcca9e
Properties: READ, WRITE
Descriptors:
Characteristic User Description
UUID: 0x2901

Unknown Characteristic
UUID: 6e400006-b5a3-f393-e0a9-e50e24dcca9e
Properties: READ, WRITE
Descriptors:
Characteristic User Description
UUID: 0x2901

Unknown Characteristic
UUID: 6e400008-b5a3-f393-e0a9-e50e24dcca9e
Properties: READ, WRITE
Descriptors:
Characteristic User Description
UUID: 0x2901

Device Information
UUID: 0x180A
PRIMARY SERVICE

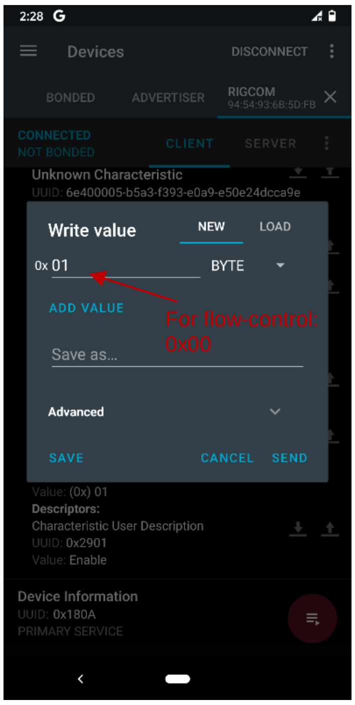
Screenshot 5

Change the value to 0x01 and press *SEND*. (See Screenshot 6)

As a second parameter we need to configure the hardware flow-control, for a correct UART communication between STM32 and BMD300. This must be disabled. Normally, this should be disabled by default. But it's better to do it manually, so you can be certain. This is done by setting the characteristic with UUID:

6E400006-B5A3-F393-E0A9-E50E24DCCA9E (see green square in Screenshot 5)

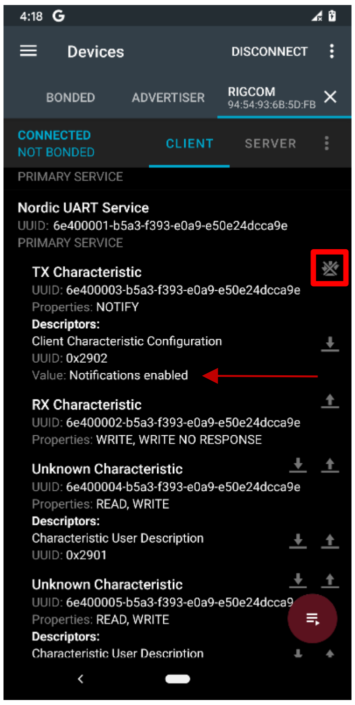
The new value must also be 0x00.


2:28 G
Devices DISCONNECT
BONDED ADVERTISER RIGCOM 94:54:93:6B:5D:FB
CONNECTED NOT BONDED CLIENT SERVER
Unknown Characteristic
UUID: 6e400005-b5a3-f393-e0a9-e50e24dcca9e
Write value NEW LOAD
0x 01 BYTE
ADD VALUE
For flow-control: 0x00
Save as...
Advanced
SAVE CANCEL SEND
Value: (0x) 01
Descriptors:
Characteristic User Description
UUID: 0x2901
Value: Enable
Device Information
UUID: 0x180A
PRIMARY SERVICE

Screenshot 6

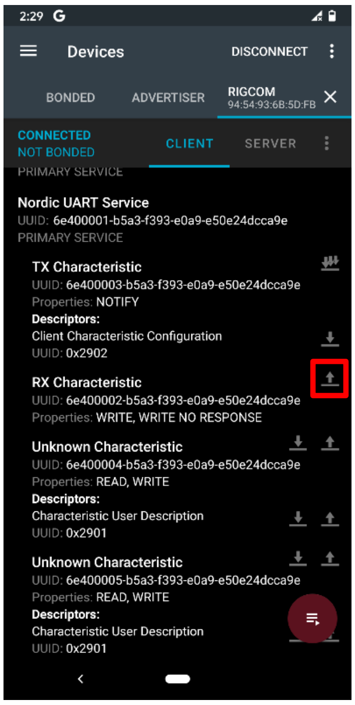
Finally, you need to enable notifications for *TX Characteristics*. This is done by clicking on the symbol with three arrows looking down. Every time you click this button, its state will change (crossed-out or not crossed-out). When the symbol is crossed-out, everything is correct and you should see that the value is “Notifications enabled”.

Now, everything is configured properly, so that you can start to use the example package.


4:18 G
Devices DISCONNECT
BONDED ADVERTISER RIGCOM 94:54:93:6B:5D:FB
CONNECTED NOT BONDED CLIENT SERVER
PRIMARY SERVICE
Nordic UART Service
UUID: 6e400001-b5a3-f393-e0a9-e50e24dcca9e
PRIMARY SERVICE
TX Characteristic
UUID: 6e400003-b5a3-f393-e0a9-e50e24dcca9e
Properties: NOTIFY
Descriptors:
Client Characteristic Configuration
UUID: 0x2902
Value: Notifications enabled
RX Characteristic
UUID: 6e400002-b5a3-f393-e0a9-e50e24dcca9e
Properties: WRITE, WRITE NO RESPONSE
Unknown Characteristic
UUID: 6e400004-b5a3-f393-e0a9-e50e24dcca9e
Properties: READ, WRITE
Descriptors:
Characteristic User Description
UUID: 0x2901
Unknown Characteristic
UUID: 6e400005-b5a3-f393-e0a9-e50e24dcca9
Properties: READ, WRITE
Descriptors:
Characteristic User Description

Screenshot 7

The core function of this example package is to send a string via Bluetooth to the display board. This string will be painted on the display. You send a string by writing it to the *RX Characteristic*.



2:29 G

Devices DISCONNECT

BONDED ADVERTISER RIGCOM 94:54:93:6B:5D:FB

CONNECTED
NOT BONDED CLIENT SERVER

PRIMARY SERVICE

Nordic UART Service
UUID: 6e400001-b5a3-f393-e0a9-e50e24dcca9e
PRIMARY SERVICE

TX Characteristic
UUID: 6e400003-b5a3-f393-e0a9-e50e24dcca9e
Properties: NOTIFY
Descriptors:
Client Characteristic Configuration
UUID: 0x2902

RX Characteristic
UUID: 6e400002-b5a3-f393-e0a9-e50e24dcca9e
Properties: WRITE, WRITE NO RESPONSE

Unknown Characteristic
UUID: 6e400004-b5a3-f393-e0a9-e50e24dcca9e
Properties: READ, WRITE
Descriptors:
Characteristic User Description
UUID: 0x2901

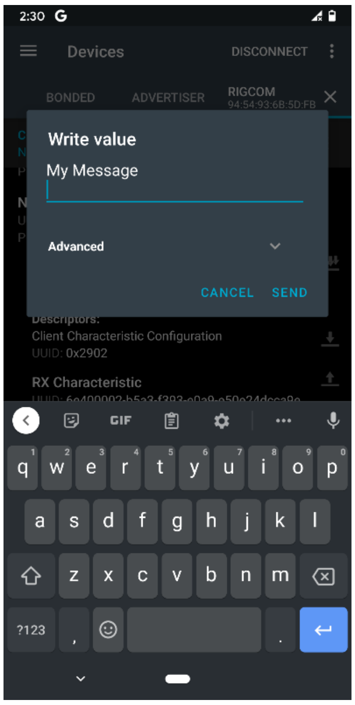
Unknown Characteristic
UUID: 6e400005-b5a3-f393-e0a9-e50e24dcca9e
Properties: READ, WRITE
Descriptors:
Characteristic User Description
UUID: 0x2901

Screenshot 8

Just type in your text and press *SEND*.

Note, that you need to **finish your string with a new line (CR)**. Otherwise, the example code is not able to distinguish between subsequent inputs.

Also note, that each message can only contain 20 bytes, including the newline-character at the end and all spaces. Therefore, your string can only consist of 19 characters. If your string consists of more than 19 characters +CR, the connection with the BLE-Module is terminated.



2:30 G

Devices DISCONNECT

BONDED ADVERTISER RIGCOM 94:54:93:6B:5D:FB

Write value
My Message

Advanced

CANCEL SEND

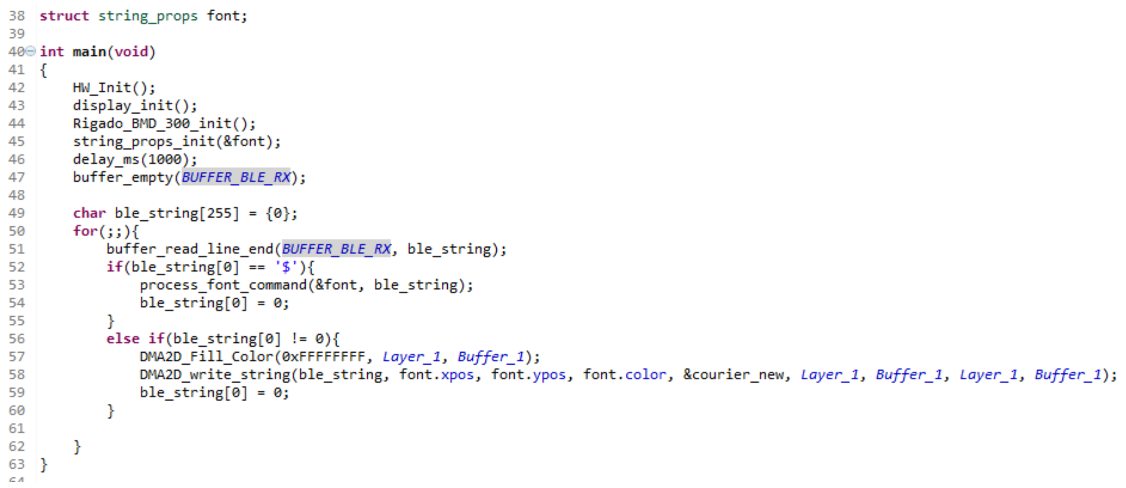
Descriptors:
Client Characteristic Configuration
UUID: 0x2902

RX Characteristic

Screenshot 9

2. Explanation of Example Code

2.1 Main function



```c
struct string_props font;

int main(void)
{
    HW_Init();
    display_init();
    Rigado_BMD_300_init();
    string_props_init(&font);
    delay_ms(1000);
    buffer_empty(BUFFER_BLE_RX);

    char ble_string[255] = {0};
    for(;;){
        buffer_read_line_end(BUFFER_BLE_RX, ble_string);
        if(ble_string[0] == '$'){
            process_font_command(&font, ble_string);
            ble_string[0] = 0;
        }
        else if(ble_string[0] != 0){
            DMA2D_Fill_Color(0xFFFFFFFF, Layer_1, Buffer_1);
            DMA2D_write_string(ble_string, font.xpos, font.ypos, font.color, &courier_new, Layer_1, Buffer_1, Layer_1, Buffer_1);
            ble_string[0] = 0;
        }

    }
}
```

We will start our walkthrough of the code at the main function since this gives a perfect overview of the steps that we will take. At the beginning, the used peripherals are initialized. *HW_Init()* initializes and turns on the display backlight. Also, it sets up the different system clocks and the timer, which counts every microsecond.

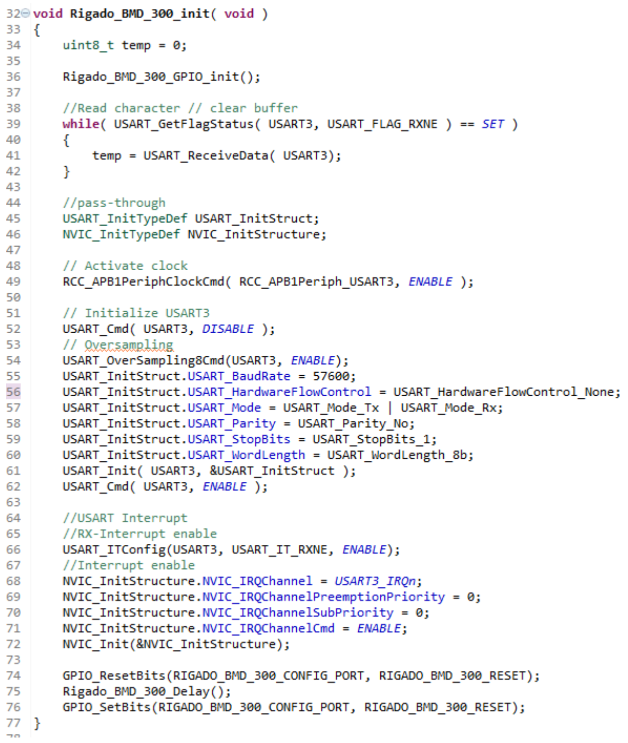
Afterwards we initialize the display which also includes the initialization of the SDRAM, because this is where the buffer for the displayed images will be located. The DMA2D is used to send the image data directly from the buffer to the LCD, which speeds up the image manipulation. Part of the display initialization is the drawing of the EBS-SYSTART logo. The function that executes this task is explained in the Example Package-documentation for Drawing Text and Images.

The function *Rigado_BMD_300_init* initializes the UART communication and the interrupt function for the connection between the BMD300 module and the STM32-chip.

At the end of the initialization the struct *font* is filled with default values by *string_props_init*. We need a struct to store the values for the position and the color of the string, so that other functions can change these depending on a transmitted command. Then we wait a second and clear the buffer for the UART data. This is necessary, because the ESP32-WROOM-32 module, which is connected to the same UART lines, sends a message on each restart, which contains its configuration. So, we wait a second to make sure, we got the whole message and delete it afterwards.

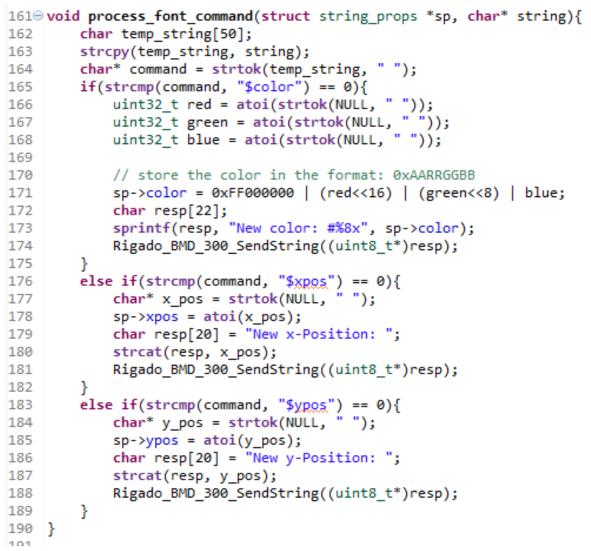
In the main loop, the UART buffer is periodically checked for a new line of input. If the new line contains a command, which is marked by beginning with a $-symbol, a function to process the command in called. Otherwise, the string will be displayed on the screen on the position and in the color, that is stored in *font*.

2.2 Rigado_BMD_300_init


```c
void Rigado_BMD_300_init( void )
{
    uint8_t temp = 0;

    Rigado_BMD_300_GPIO_init();

    //Read character // clear buffer
    while( USART_GetFlagStatus( USART3, USART_FLAG_RXNE ) == SET )
    {
        temp = USART_ReceiveData( USART3);
    }

    //pass-through
    USART_InitTypeDef USART_InitStruct;
    NVIC_InitTypeDef NVIC_InitStructure;

    // Activate clock
    RCC_APB1PeriphClockCmd( RCC_APB1Periph_USART3, ENABLE );

    // Initialize USART3
    USART_Cmd( USART3, DISABLE );
    // Oversampling
    USART_OverSampling8Cmd(USART3, ENABLE);
    USART_InitStruct.USART_BaudRate = 57600;
    USART_InitStruct.USART_HardwareFlowControl = USART_HardwareFlowControl_None;
    USART_InitStruct.USART_Mode = USART_Mode_Tx | USART_Mode_Rx;
    USART_InitStruct.USART_Parity = USART_Parity_No;
    USART_InitStruct.USART_StopBits = USART_StopBits_1;
    USART_InitStruct.USART_WordLength = USART_WordLength_8b;
    USART_Init( USART3, &USART_InitStruct );
    USART_Cmd( USART3, ENABLE );

    //USART Interrupt
    //RX-Interrupt enable
    USART_ITConfig(USART3, USART_IT_RXNE, ENABLE);
    //Interrupt enable
    NVIC_InitStructure.NVIC_IRQChannel = USART3_IRQn;
    NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
    NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;
    NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
    NVIC_Init(&NVIC_InitStructure);

    GPIO_ResetBits(RIGADO_BMD_300_CONFIG_PORT, RIGADO_BMD_300_RESET);
    Rigado_BMD_300_Delay();
    GPIO_SetBits(RIGADO_BMD_300_CONFIG_PORT, RIGADO_BMD_300_RESET);
}
```

At the beginning of this function the GPIOs for the UART connection to the BMD300 are initialized. Afterwards, the parameters for the UART communication are configured. Note that the baud rate is 57600 this time. This is the default value of the BMD300. If you want another baud rate, you need to change this on the Bluetooth module as well.

2.3 process_font_command


```c
void process_font_command(struct string_props *sp, char* string){
    char temp_string[50];
    strcpy(temp_string, string);
    char* command = strtok(temp_string, " ");
    if(strcmp(command, "$color") == 0){
        uint32_t red = atoi(strtok(NULL, " "));
        uint32_t green = atoi(strtok(NULL, " "));
        uint32_t blue = atoi(strtok(NULL, " "));

        // store the color in the format: 0xAARRGGBB
        sp->color = 0xFF000000 | (red<<16) | (green<<8) | blue;
        char resp[22];
        sprintf(resp, "New color: #%8x", sp->color);
        Rigado_BMD_300_SendString((uint8_t*)resp);
    }
    else if(strcmp(command, "$xpos") == 0){
        char* x_pos = strtok(NULL, " ");
        sp->xpos = atoi(x_pos);
        char resp[20] = "New x-Position: ";
        strcat(resp, x_pos);
        Rigado_BMD_300_SendString((uint8_t*)resp);
    }
    else if(strcmp(command, "$ypos") == 0){
        char* y_pos = strtok(NULL, " ");
        sp->ypos = atoi(y_pos);
        char resp[20] = "New y-Position: ";
        strcat(resp, y_pos);
        Rigado_BMD_300_SendString((uint8_t*)resp);
    }
}
```

This function is called, after we already know, that the string contains a command. The command and its parameters are separated by spaces and therefore we use the C-library function *strtok* to get each individually. The application knows only three different commands; one to change the color of the strings, one to change the x-position and one to change the y-position.

The commands must have the following format:

Color Command:	$color *R G B*	With *R*, *G* and *B* being numbers in the range of 0 to 255 representing the red, green and blue values
x-Position Command:	$xpos *value*	With *value* being the pixel number of the horizontal start position. The range must not succeed the display size.
y-Position Command:	$ypos *value*	With *value* being the pixel number of the vertical start position. The range must not succeed the display size.
Note, that each command has to end with a new line (CR) just like every message transmitted to the display board via Bluetooth. Otherwise, the STM32 can't distinguish between subsequent messages.		