Topic:	BLE-Module CubeIDE
Author:	PARA
Date:	30.11.2021

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_cubeide_gettingstarted”. Make sure you have read this document beforehand and executed all the steps to configure your STM32CubeIDE.

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

1. Introduction

In this example package the BLE module which can be found on the 4.3 – inch and 5.0 – inch display board will be put into 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 connection and the sending/receiving of data via Bluetooth you need the software *LightBlue* by PunchThrough, which is available for your smartphone (Android and iOS).

1.1 Using *LightBlue* and configuring the BMD-300

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

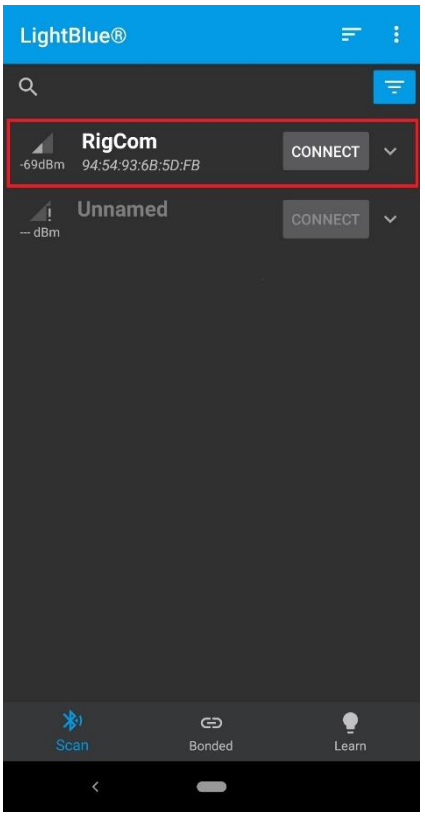
The screenshot in the middle is from the apple version of the app, the one on the right is from the android version.

Install the app from your App Store and open it. At the beginning you will need to scan (pull to refresh) for the available Bluetooth devices and search for the BMD-300.

Obviously, the display board has to be connected to its power.

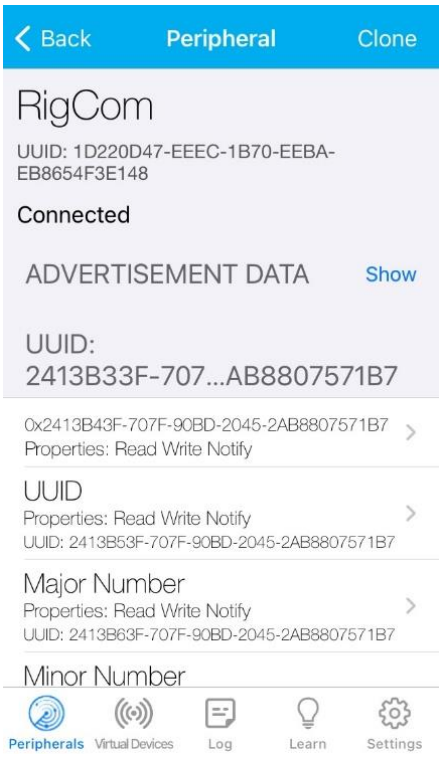
Note: By default, the BMD-300 device is called “RigCom”. Click on the entry in the list to connect to the device.

Screenshot 1.1

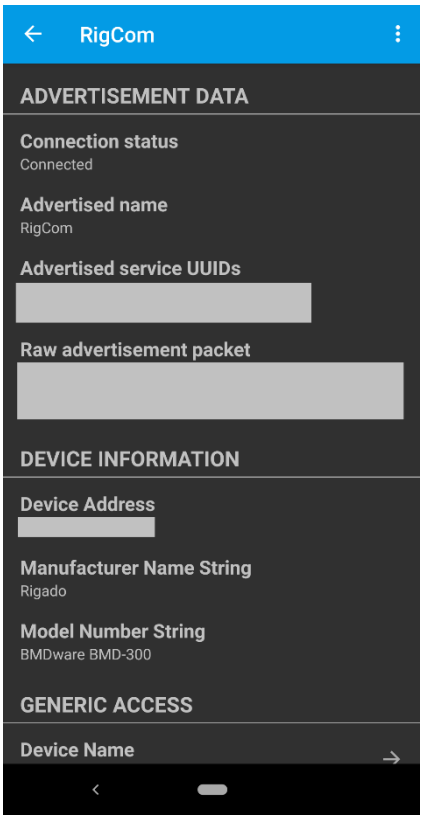


Screenshot 1.2

You should now see this overview page when successfully connected.



Screenshot 2.1



Screenshot 2.2

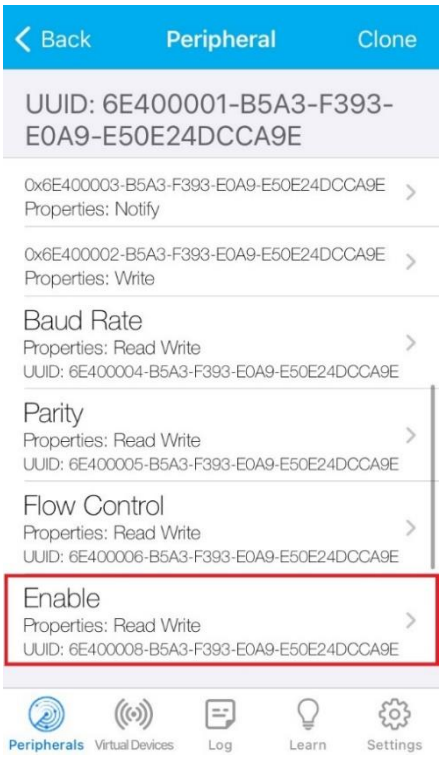
Now we need to make some changes in the configurations, so that the BMD300 can communicate properly with the STM32-Microcontroller.

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.

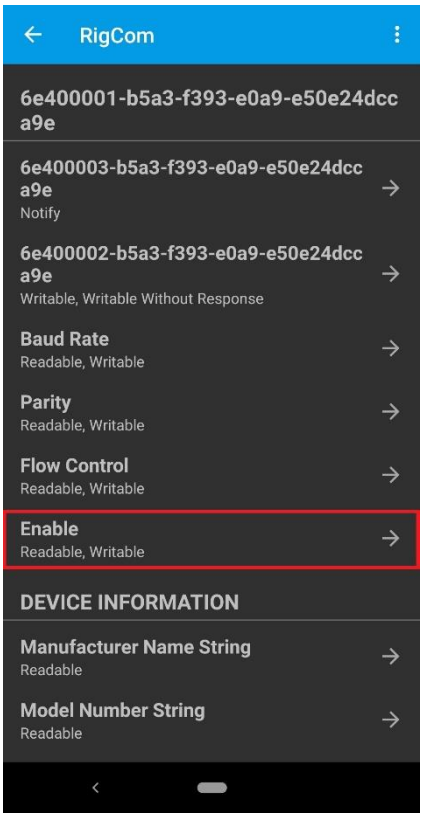
To enable this, you need to scroll down to the UUID:

6E400008-B5A3-F393-E0A9-E50E24DCCA9E

(Notice the last number before the first minus sign)



Screenshot 3.1

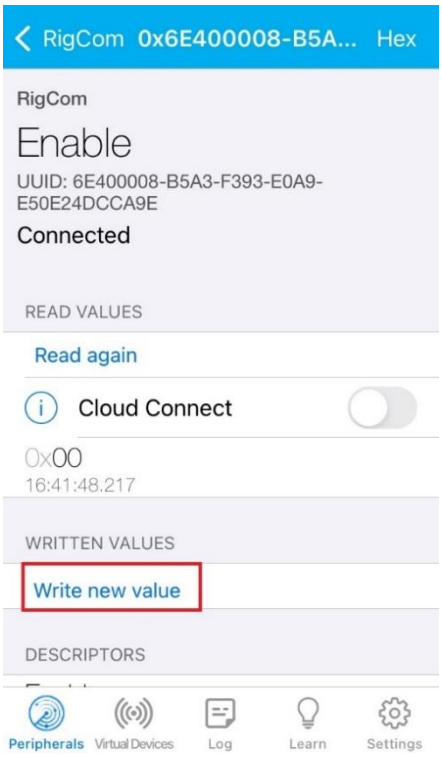


Screenshot 3.2

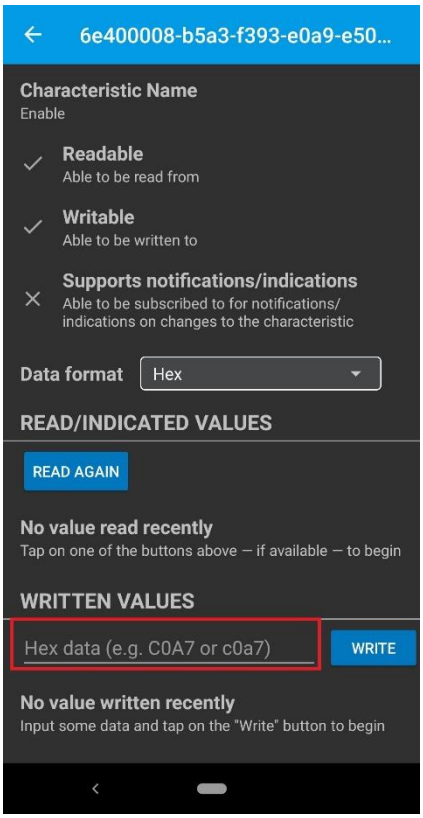
In this menu on the iPhone, you have to click on the *Write new value* button.

In the Android Version, click on the marked area.

(Screenshot 4.2)



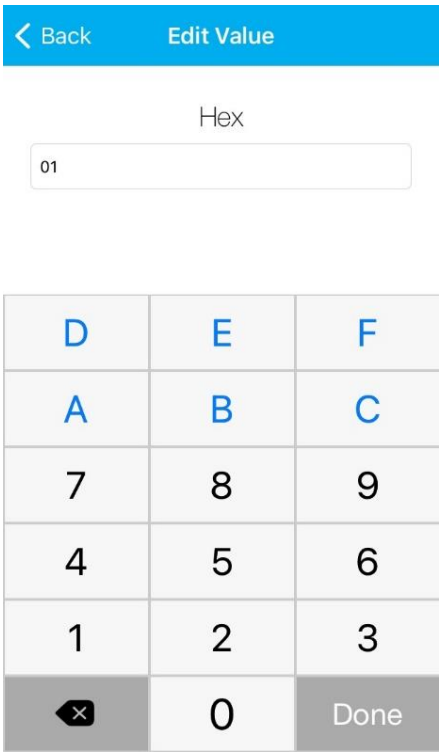
Screenshot 4.1



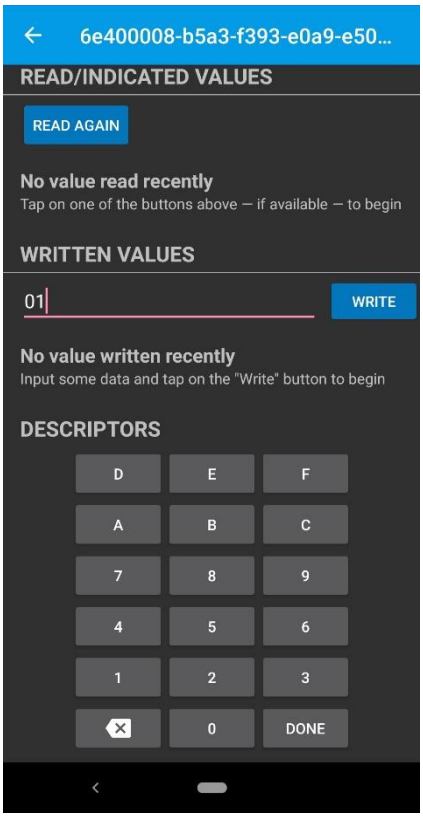
Screenshot 4.2

Now the following editor opens, where you have to enter the value 01 and click on the *Done* button in the lower right corner.

In the Android Version enter the Value 01 and click on WRITE.



Screenshot 5.1



Screenshot 5.2

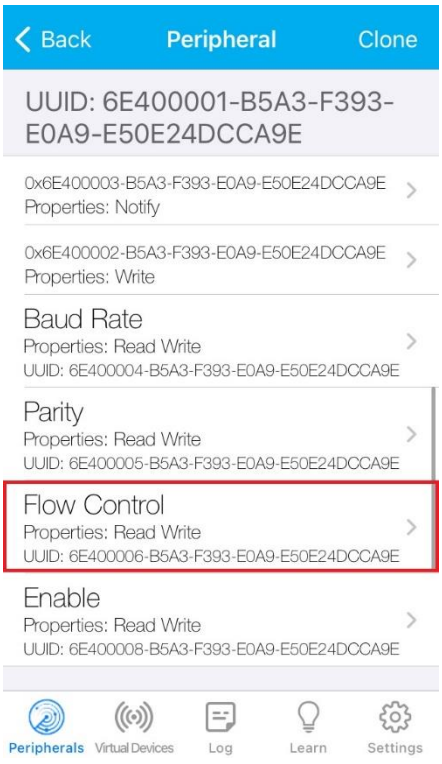
As a second parameter we need to configure the hardware flow-control, for a correct UART communication between STM32 and BMD300. This must be disabled.

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:

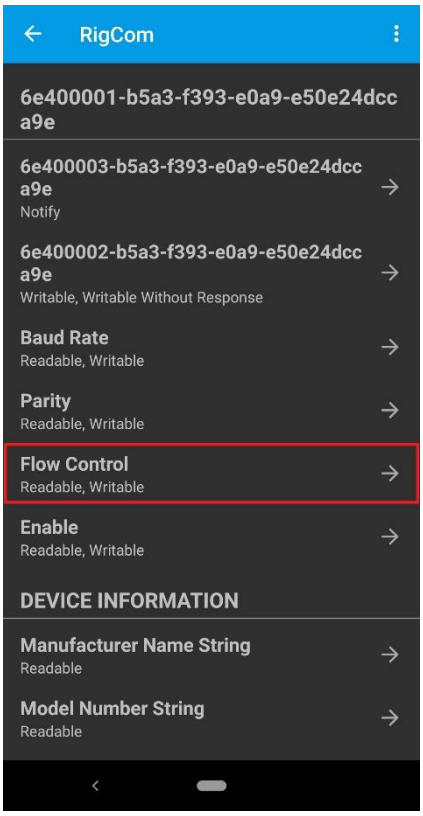
6E400006-B5A3-F393-E0A9-E50E24DCCA9E

(Notice the last number before the first minus sign)

Write the value 00 exactly as you have done above and click on Done.



Screenshot 6.1

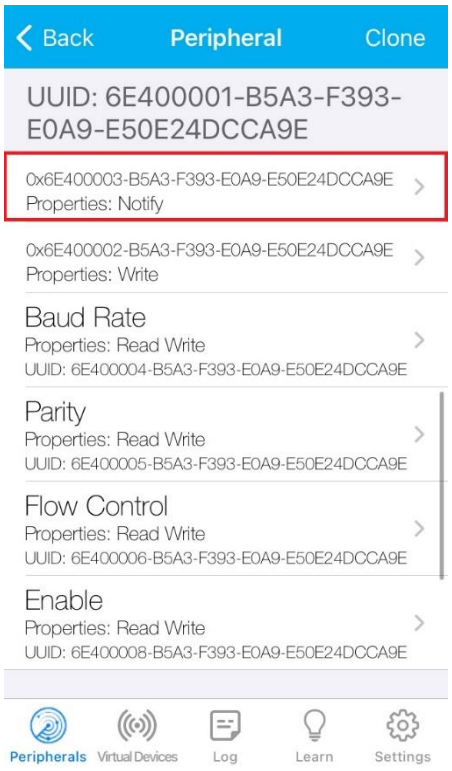


Screenshot 6.2

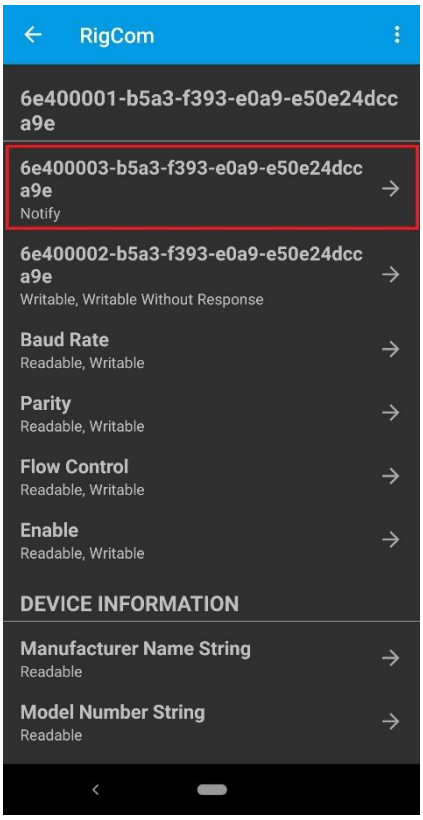
Finally, you need to enable notifications. This is done by clicking on the option with the UUID:

6E400003-B5A3-F393-E0A9-E50E24DCCA9E

(Notice the last number before the first minus sign)



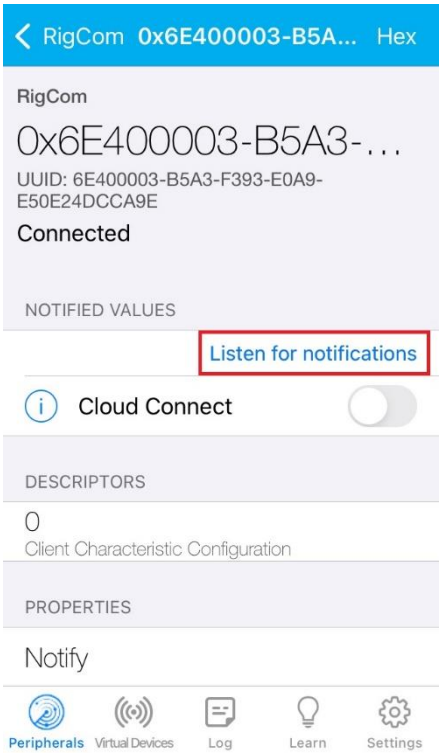
Screenshot 7.1



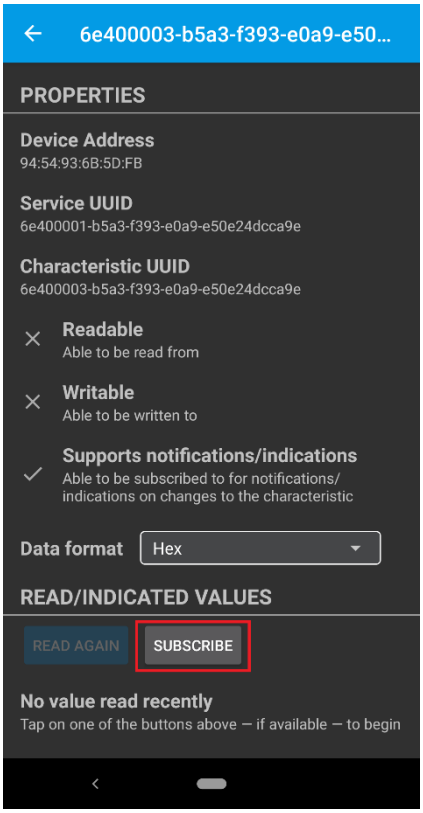
Screenshot 7.2

In this menu, click on the *Listen for notifications* button.

On Android, simply click on *SUBSCRIBE.*



Screenshot 8.1

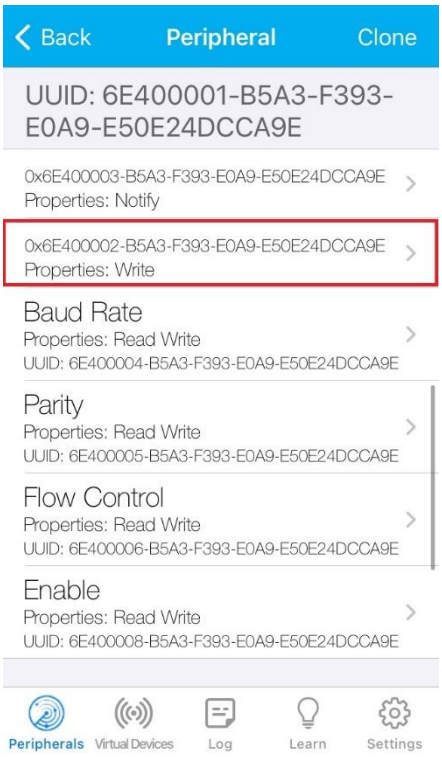


Screenshot 8.2

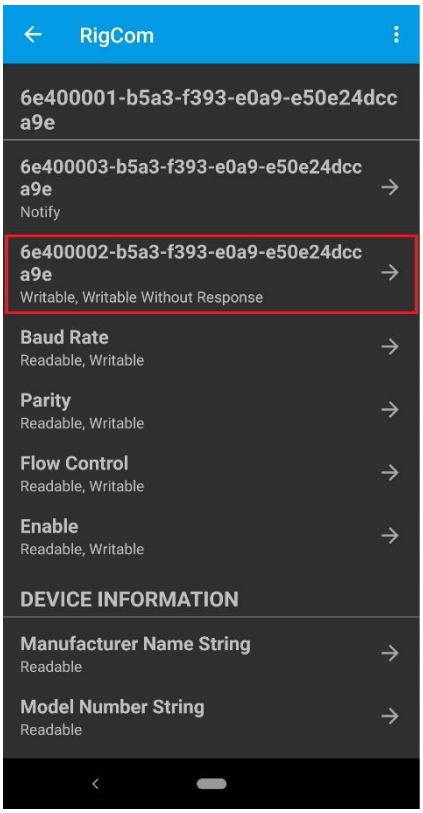
The core function of this example package is to send a string via Bluetooth to the display board. This string will be shown on the display. You can send a string by clicking on the UUID:

6E400002-B5A3-F393-E0A9-E50E24DCCA9E

(Notice the last number before the first minus sign)



Screenshot 9.1

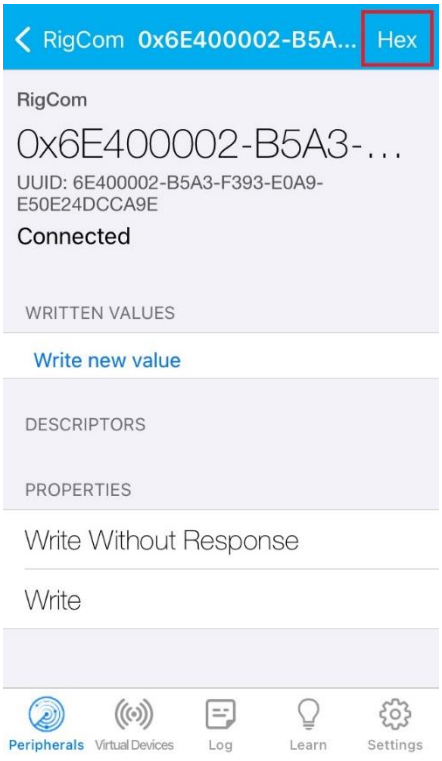


Screenshot 9.2

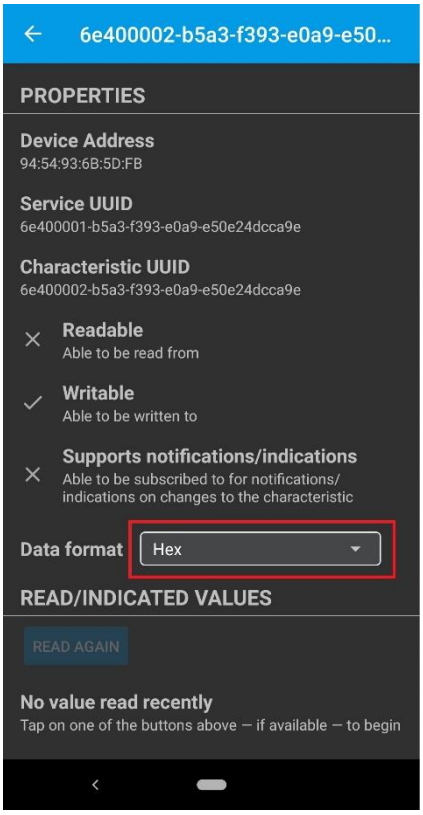
To send a string you have to change the **Characteristic Format** to *UTF-8 String.*

To do that click on the Button in the upper right corner of the screen and change it to the desired input format.

On Android open the *Data format* option.

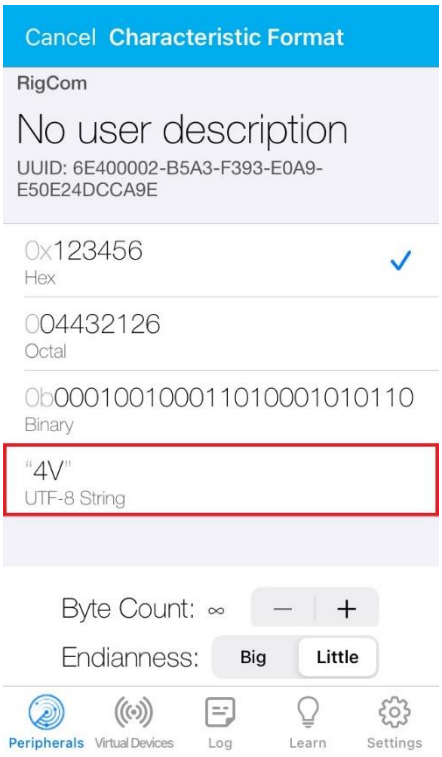


Screenshot 10.1

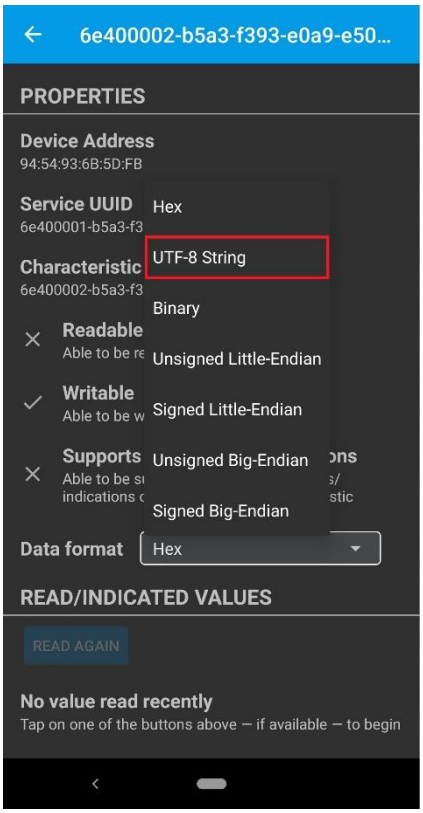


Screenshot 10.2

In the menu click on *UTF-8 String.*



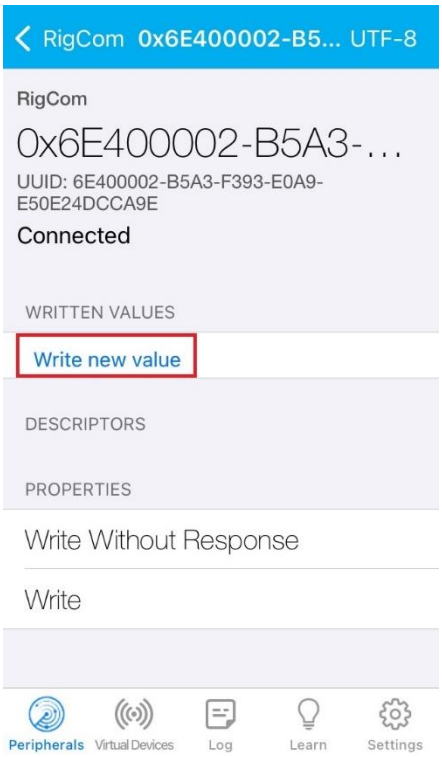
Screenshot 11.1



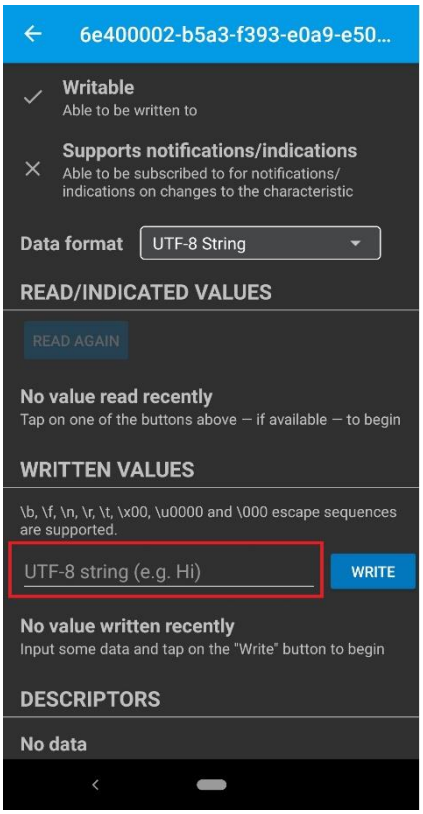
Screenshot 11.2

To finally send a string, click on the **Write new value** button.

On Android click on the marked area on the screen.



Screenshot 12.1



Screenshot 12.2

Just input your text, add the characters “\n” to the end and press *Enter.*

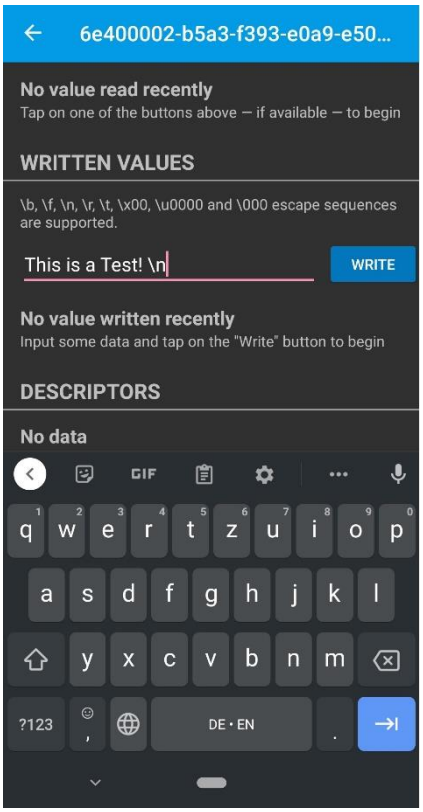
You need to finish your string with a new line symbol! (\n)

Otherwise, the STM32 code is not able to distinguish between subsequent inputs.

Also note, that your message should not exceed 20 characters, including the newline symbol.

If everything has worked you should now see the text on the display.

Screenshot 13.1



Screenshot 13.2

2. Explanation of Example Code

2.1 Main function

We will start our walk through the code at the main function since this gives a perfect overview of the steps that we will take. At the beginning, the hardware abstraction layer (HAL) and the system clocks are initialized. Then all the peripherals are initialized. A few variables are declared for later use.

The 5.0-inch display need the pin PH6 to be set to high in order to enable the backlight. This display is recognized by the define BACKLIGHT_EN which is defined in the global_Display_Touch_HAL.h file. After that, the display startup sequence, consisting of filling the buffers/layers with the color white and displaying our 2 logos, is started.

Following the short delay of the display sequence the Rigado BMD is initialized.

Then the active display buffer is filled with the color 0xFFFFFFFF (ARGB8888) as background. Now the string with the instructions is drawn to the display.

The font struct is initialized and the ringbuffer is reset.

```
int main(void)
{
  /* USER CODE BEGIN 1 */

  /* USER CODE END 1 */

  /* MCU Configuration--------------------------------------------------------*/

  /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
  HAL_Init();

  /* USER CODE BEGIN Init */

  /* USER CODE END Init */

  /* Configure the system clock */
  SystemClock_Config();

  /* USER CODE BEGIN SysInit */

  /* USER CODE END SysInit */

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_DMA2D_Init();
  MX_FMC_Init();
  MX_LTDC_Init();
  MX_I2C1_Init();
  MX_USART1_UART_Init();
  MX_USART3_UART_Init();
  MX_USART6_UART_Init();
  MX_TIM2_Init();
  /* USER CODE BEGIN 2 */
  /* Declaration of variables */
  Buffer_e activeBuffer;
  char ble_string[255] = {0};

  /* Initialize the uart ringbuffer */
  uart_init();

  /* Enable backlight */
  HAL_GPIO_WritePin(GPIOG, GPIO_PIN_9, GPIO_PIN_SET);
  if (BACKLIGHT_EN) {
    HAL_Delay(100);
    HAL_GPIO_WritePin(GPIOH, GPIO_PIN_6, GPIO_PIN_SET);
  }

  /* Display the logos and save active buffer */
  activeBuffer = Display_Startup_Sequence();

  /* Enable the rigado bluetooth module */
  Rigado_BMD_300_init();

  /* Display the instructions to recveive bluetooth messages */
  DMA2D_Fill_Color(0xFFFFFFFF, Layer_2, activeBuffer, false);
  DMA2D_write_string("Bluetooth waiting", 25, 20, 0xFF000000, &courier_new_23, Layer_2, activeBuffer, false);
  DMA2D_write_string("for Connection.", 25, 50, 0xFF000000, &courier_new_23, Layer_2, activeBuffer, false);
  DMA2D_write_string("Please enter your String", 25, 80, 0xFF000000, &courier_new_23, Layer_2, activeBuffer, false);
  DMA2D_write_string("via Bluetooth.", 25, 110, 0xFF000000, &courier_new_23, Layer_2, activeBuffer,  false);

  /* Initialize the string property structure */
  string_props_init(&font);

  /* Wait for rigado */
  HAL_Delay(1000);

  /* Empty out ringbuffer */
  reset_ringbuffer(COM3);
  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
    /* Clear ble_string */
    memset(ble_string, 0, 255);
    /* Receive string through uart ringbuffer */
    uart_ReceiveString(COM3, ble_string);
    /* Check if command string received and process it */
    if(ble_string[0] == '$'){
      process_font_command(&font, ble_string);
      ble_string[0] = 0;
    }
    /* Check if string is not empty and display it */
    else if(ble_string[0] != 0){
      DMA2D_Fill_Color(0xFFFFFFFF, Layer_2, activeBuffer, true);
      DMA2D_write_string(ble_string, font.xpos, font.ypos, font.color, &courier_new_23, Layer_2, activeBuffer, false);
      ble_string[0] = 0;
    }
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
  }
  /* USER CODE END 3 */
}
```

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 is called. Otherwise, the string will be displayed on the screen on the position and in the color, that is stored in the *font* struct.

2.2 Rigado_BMD_300_init

This function disables the Rigado BMD 300, waits for half a second and enables it again to be sure its enabled.

```
void Rigado_BMD_300_init( void )
{
  GPIOB -> ODR &= ~GPIO_PIN_15;
  HAL_Delay(500);
  GPIOB -> ODR |= GPIO_PIN_15;

}
```

2.3 process_font_command

```
79  void process_font_command(struct string_props *sp, char* string){
80    char temp_string[50];
81    strcpy(temp_string, string);
82    char* command = strtok(temp_string, " ");
83    if(strcmp(command, "$color") == 0){
84      uint32_t red = atoi(strtok(NULL, " "));
85      uint32_t green = atoi(strtok(NULL, " "));
86      uint32_t blue = atoi(strtok(NULL, " "));
87
88      // store the color in the format: 0xAARRGGBB
89      sp->color = 0xFF000000 | (red<<16) | (green<<8) | blue;
90      char resp[22];
91      sprintf(resp, "New color: #%8lx", sp->color);
92      Rigado_BMD_300_SendString((uint8_t*)resp);
93    }
94    else if(strcmp(command, "$xpos") == 0){
95      char* x_pos = strtok(NULL, " ");
96      sp->xpos = atoi(x_pos);
97      char resp[20] = "New x-Position: ";
98      strcat(resp, x_pos);
99      Rigado_BMD_300_SendString((uint8_t*)resp);
100   }
101   else if(strcmp(command, "$ypos") == 0){
102     char* y_pos = strtok(NULL, " ");
103     sp->ypos = atoi(y_pos);
104     char resp[20] = "New y-Position: ";
105     strcat(resp, y_pos);
106     Rigado_BMD_300_SendString((uint8_t*)resp);
107   }
108 }
```

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* \n	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* \n	With *value* being the number of pixel of the horizontal start position. The range must not exceed the display size.
y-Position Command:	$ypos *value* \n	With *value* being the pixel number of the vertical start position. The range must not exceed the display size.
Note, that each command has to end with the UTF-8 ASCII sequence "\" and "n" or the hex value 0x0A like every message transmitted to the display board via Bluetooth. Otherwise, the STM32 can't distinguish between subsequent messages.		