UART communication using LL libraries

rev1.0 24/03/2020

*Property of **BEOND s.r.l.** cannot be distributed or reproduced without authorization*

GOAL

Receive data via UART protocol

PREREQUISITES

Software needed:

- STM32IDE
- CoolTerm

Hardware used in this example:

- **NUCLEO-F446ZE**

UART communication using LL libraries

GOAL:

Receive data via UART protocol

UART communication using LL libraries

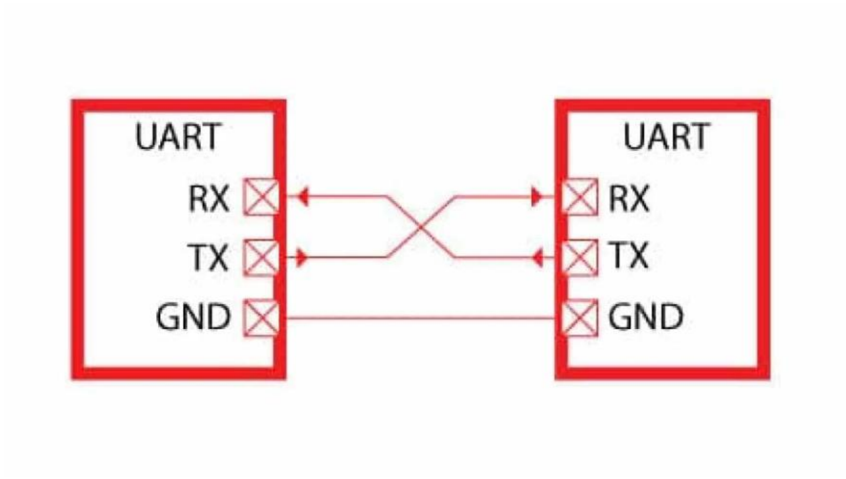
In this example:

- What’s UART
- Start a new project
- Configure the peripherals
- Generate code
- USART Configuration code
- Run the program and troubleshooting

What’s UART

“UART” stands for Universal Asynchronous receiver-transmitter. It is a peripheral that is present inside a microcontroller. The **function of UART** is to convert the incoming and outgoing data into the serial binary stream.

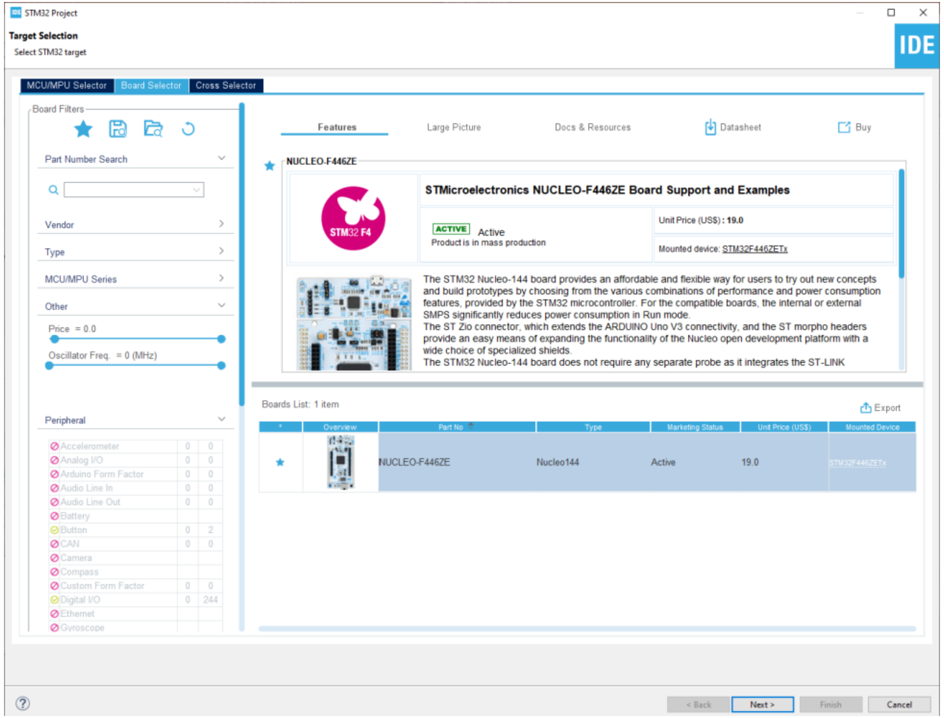
Notice that in UART communication the ***TX*** pin of the *transmitter* is connected to the ***RX*** pin of the *receiver* and viceversa.



Start a new project

From the stm32IDE software click on File -> New -> STM32 Project.

Select your board or your uC and click *next.*



Target Selection

Select STM32 target

Price = 0.0

Oscillator Freq. = 0 (MHz)

Accelerometer	0	0
Analog I/O	0	0
Arduino Form Factor	0	0
Audio Line In	0	0
Audio Line Out	0	0
Battery		
Button	0	2
CAN	0	0
Camera		
Compass		
Custom Form Factor	0	0
Digital I/O	0	244
Ethernet		
Gyroscope		

NUCLEO-F446ZE

STMicroelectronics NUCLEO-F446ZE Board Support and Examples

ACTIVE Active

Product is in mass production

Unit Price (US$) : **19.0**

Mounted device: STM32F446ZETx

The STM32 Nucleo-144 board provides an affordable and flexible way for users to try out new concepts and build prototypes by choosing from the various combinations of performance and power consumption features, provided by the STM32 microcontroller. For the compatible boards, the internal or external SMPS significantly reduces power consumption in Run mode.
The ST Zio connector, which extends the ARDUINO Uno V3 connectivity, and the ST morpho headers provide an easy means of expanding the functionality of the Nucleo open development platform with a wide choice of specialized shields.
The STM32 Nucleo-144 board does not require any separate probe as it integrates the ST-LINK

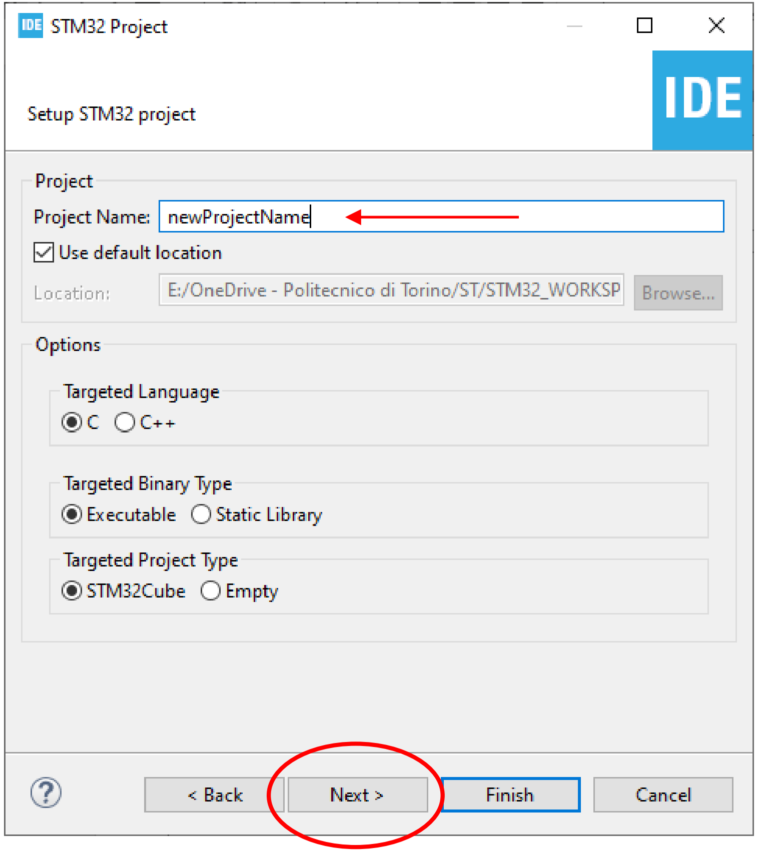
Boards List: 1 item

*	Overview	Part No	Type	Marketing Status	Unit Price (US$)	Mounted Device
★		NUCLEO-F446ZE	Nucleo144	Active	19.0	STM32F446ZETx

Start a new project

Type the name of your project and click next.

By default the project will be created in the workspace folder.



STM32 Project

Setup STM32 project

Project

Project Name: newProjectName

☑ Use default location

Location: E:/OneDrive - Politecnico di Torino/ST/STM32_WORKSP

Options

Targeted Language

● C ○ C++

Targeted Binary Type

● Executable ○ Static Library

Targeted Project Type

● STM32Cube ○ Empty

Start a new project

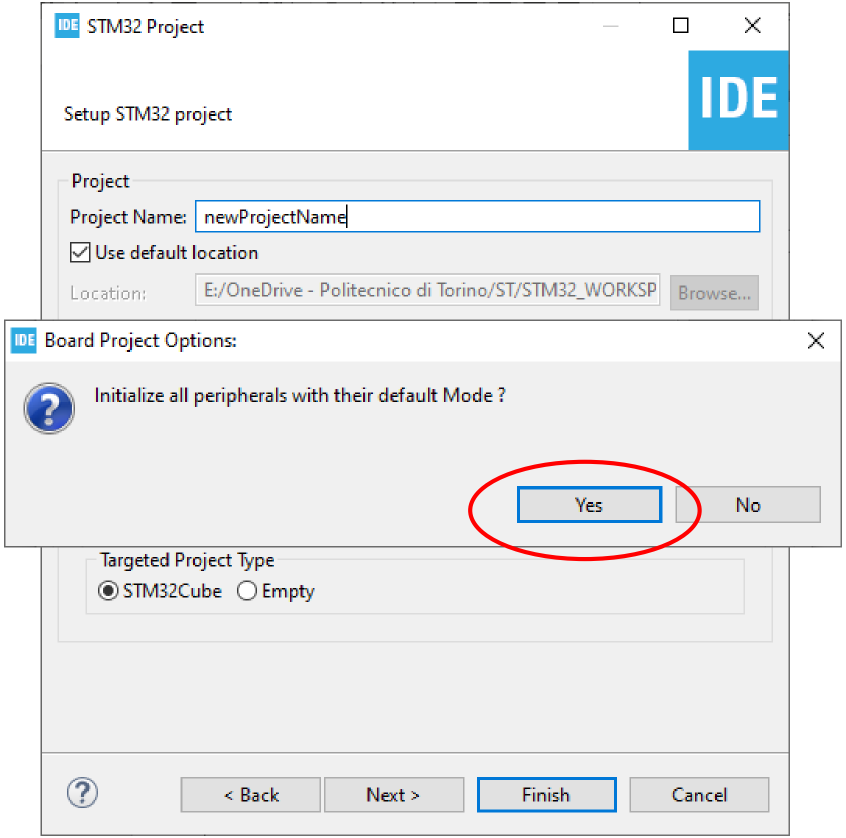
Type the name of your project and click next.

By default the project will be created in the *workspace* folder.

The *STM32IDE* has the option to initialize all the peripheral with their ***default*** mode:

Clicking *Yes* the *USART3*, all the *LEDs* and the blue *UserButton* will be configured as default.

Click *Yes*.



STM32 Project

Setup STM32 project

Project

Project Name: newProjectName

☑ Use default location

Location: E:/OneDrive - Politecnico di Torino/ST/STM32_WORKSP

Board Project Options:

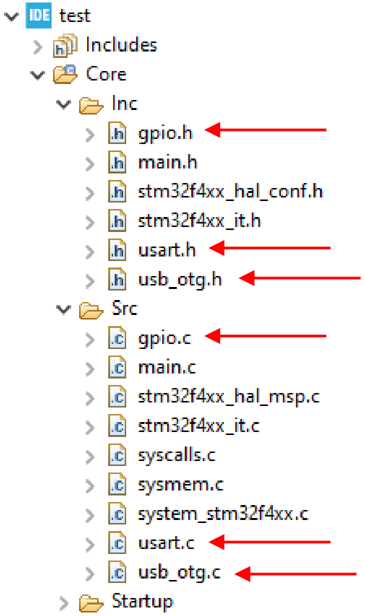
Initialize all peripherals with their default Mode ?

Targeted Project Type

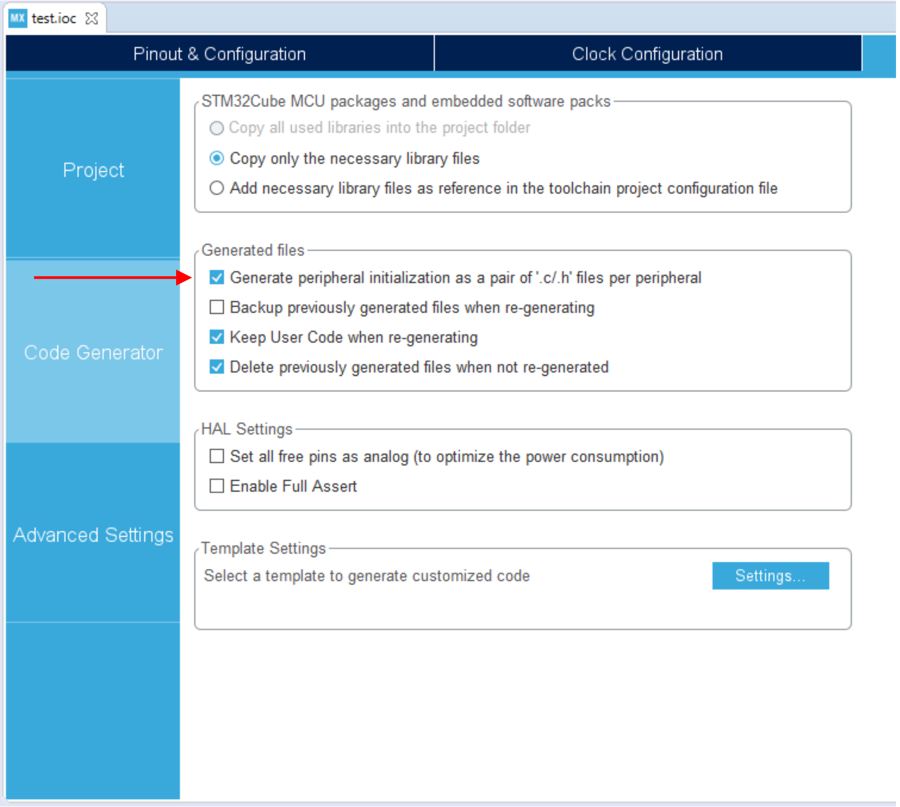
● STM32Cube ○ Empty

Project Manager

In the Code Generator Tab check the ***Generate peripheral initialization […]*** box: each periperhal will have a disting *periph.c* and *periph.h* files.



```
test
  Includes
  Core
    Inc
      gpio.h
      main.h
      stm32f4xx_hal_conf.h
      stm32f4xx_it.h
      usart.h
      usb_otg.h
    Src
      gpio.c
      main.c
      stm32f4xx_hal_msp.c
      stm32f4xx_it.c
      syscalls.c
      sysmem.c
      system_stm32f4xx.c
      usart.c
      usb_otg.c
  Startup
```



test.ioc

Pinout & Configuration | Clock Configuration

Project

Code Generator

Advanced Settings

STM32Cube MCU packages and embedded software packs
- ○ Copy all used libraries into the project folder
- ● Copy only the necessary library files
- ○ Add necessary library files as reference in the toolchain project configuration file

Generated files
- ☑ Generate peripheral initialization as a pair of '.c/.h' files per peripheral
- □ Backup previously generated files when re-generating
- ☑ Keep User Code when re-generating
- ☑ Delete previously generated files when not re-generated

HAL Settings
- □ Set all free pins as analog (to optimize the power consumption)
- □ Enable Full Assert

Template Settings

Select a template to generate customized code

Settings...

Configure the peripherals

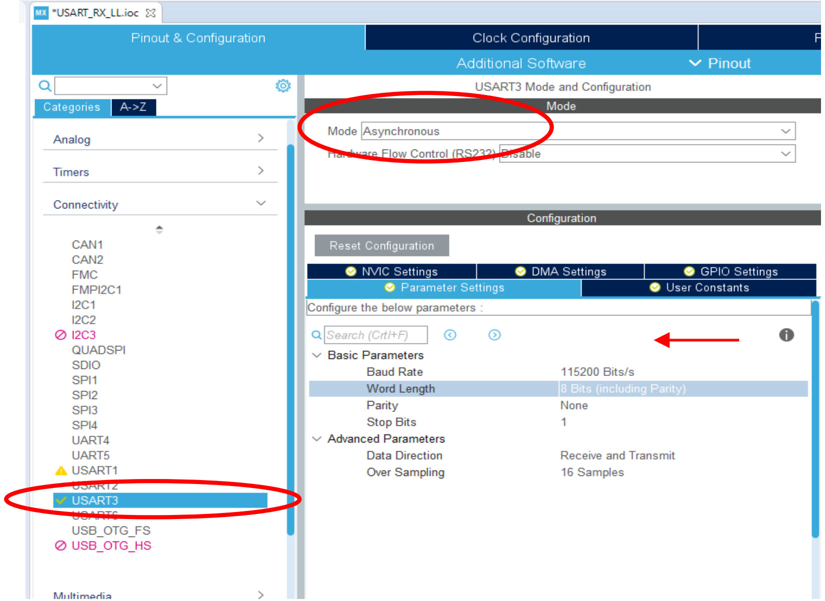
The main core of this project is the USART peripheral, let’s have a look in the USART page configuration.

For this example we will use the USART3:

- *Mode*: Asynchronous
- *Baud Rate*: 115200 Bits/s

The *Baud Rate* is the speed of the serial communication, 115200 Bits/s is a reasonable speed.

Don’t forget to enable the *interrupt* under the **NVIC** tab.



Mode: Asynchronous

Hardware Flow Control (RS232): Disable

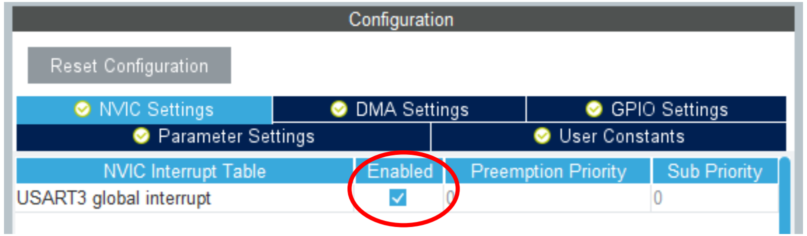
Configure the below parameters :

Basic Parameters

Baud Rate	115200 Bits/s
Word Length	8 Bits (including Parity)
Parity	None
Stop Bits	1

Advanced Parameters

Data Direction	Receive and Transmit
Over Sampling	16 Samples



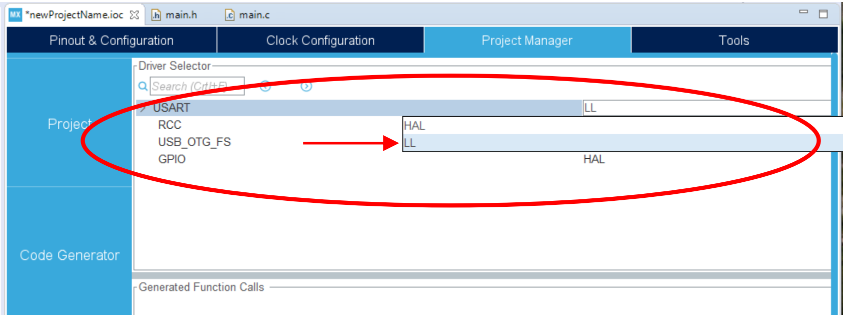
NVIC Interrupt Table	Enabled	Preemption Priority	Sub Priority
USART3 global interrupt	☑	0	0

Generate code

The last step before generating the code is to select LL libraries for manage the USART peripheral.

Go in *Project Manager -> Advanced Configuration* and select ***LL*** (USART and GPIO).

Then generate the code (click on the generate icon)



USART	LL
RCC	HAL LL
USB_OTG_FS	
GPIO	HAL

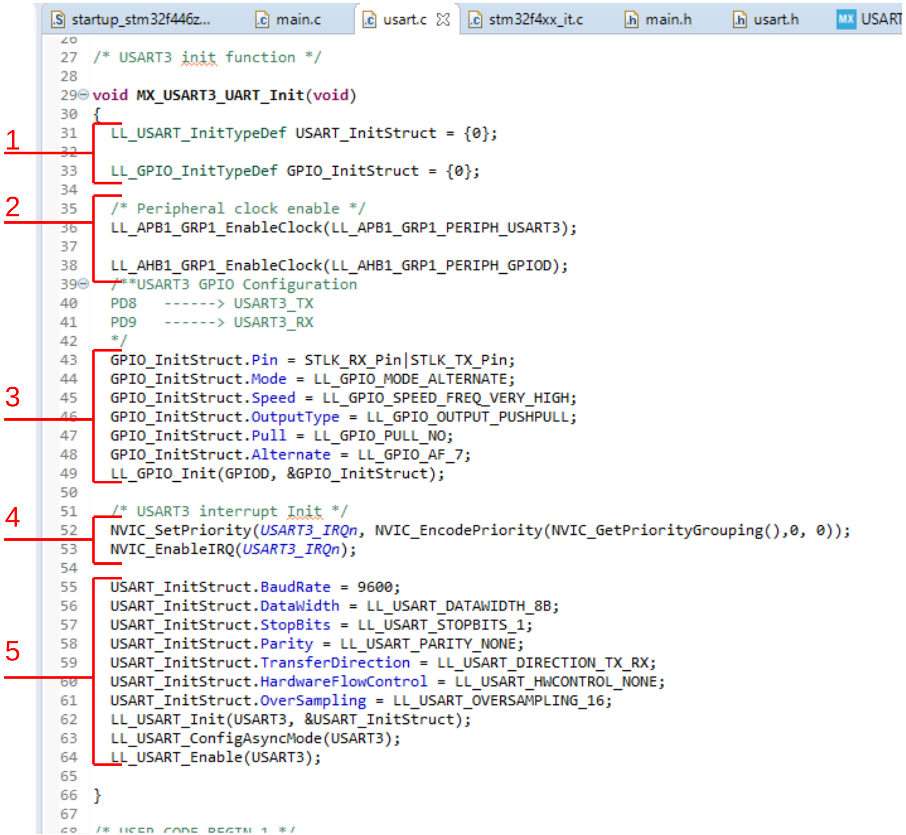
USART Configuration

Let’s have a look in the **USART** configuration function made by *CubeMX*.

As always we can find it in the **usart.c** file

We can subdivide this section in 5 parts:

1. Inizialization of the GPIO and USART structures, they contains all the configuration info for the peripherals.
2. Enable of the clock sources for the GPIO pins and for the USART bus.

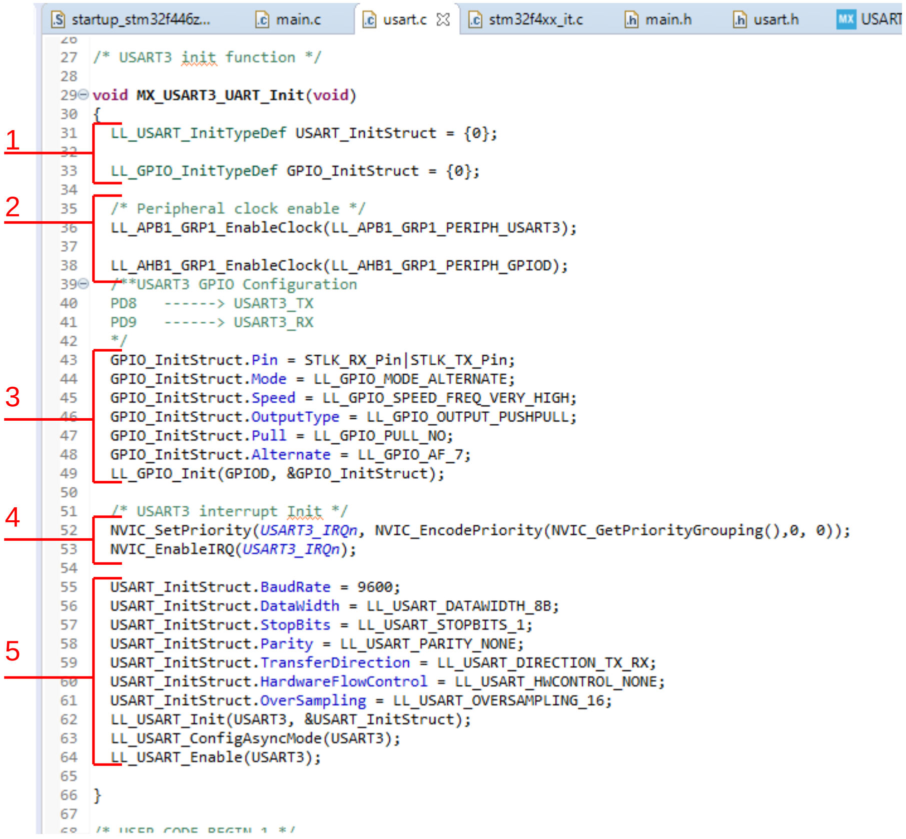


```
/* USART3 init function */

void MX_USART3_UART_Init(void)
{
  LL_USART_InitTypeDef USART_InitStruct = {0};

  LL_GPIO_InitTypeDef GPIO_InitStruct = {0};

  /* Peripheral clock enable */
  LL_APB1_GRP1_EnableClock(LL_APB1_GRP1_PERIPH_USART3);

  LL_AHB1_GRP1_EnableClock(LL_AHB1_GRP1_PERIPH_GPIOD);
  /**USART3 GPIO Configuration
  PD8   ------> USART3_TX
  PD9   ------> USART3_RX
  */
  GPIO_InitStruct.Pin = STLK_RX_Pin|STLK_TX_Pin;
  GPIO_InitStruct.Mode = LL_GPIO_MODE_ALTERNATE;
  GPIO_InitStruct.Speed = LL_GPIO_SPEED_FREQ_VERY_HIGH;
  GPIO_InitStruct.OutputType = LL_GPIO_OUTPUT_PUSHPULL;
  GPIO_InitStruct.Pull = LL_GPIO_PULL_NO;
  GPIO_InitStruct.Alternate = LL_GPIO_AF_7;
  LL_GPIO_Init(GPIOD, &GPIO_InitStruct);

  /* USART3 interrupt Init */
  NVIC_SetPriority(USART3_IRQn, NVIC_EncodePriority(NVIC_GetPriorityGrouping(),0, 0));
  NVIC_EnableIRQ(USART3_IRQn);

  USART_InitStruct.BaudRate = 9600;
  USART_InitStruct.DataWidth = LL_USART_DATAWIDTH_8B;
  USART_InitStruct.StopBits = LL_USART_STOPBITS_1;
  USART_InitStruct.Parity = LL_USART_PARITY_NONE;
  USART_InitStruct.TransferDirection = LL_USART_DIRECTION_TX_RX;
  USART_InitStruct.HardwareFlowControl = LL_USART_HWCONTROL_NONE;
  USART_InitStruct.OverSampling = LL_USART_OVERSAMPLING_16;
  LL_USART_Init(USART3, &USART_InitStruct);
  LL_USART_ConfigAsyncMode(USART3);
  LL_USART_Enable(USART3);

}

/* USER CODE BEGIN 1 */
```

USART Configuration

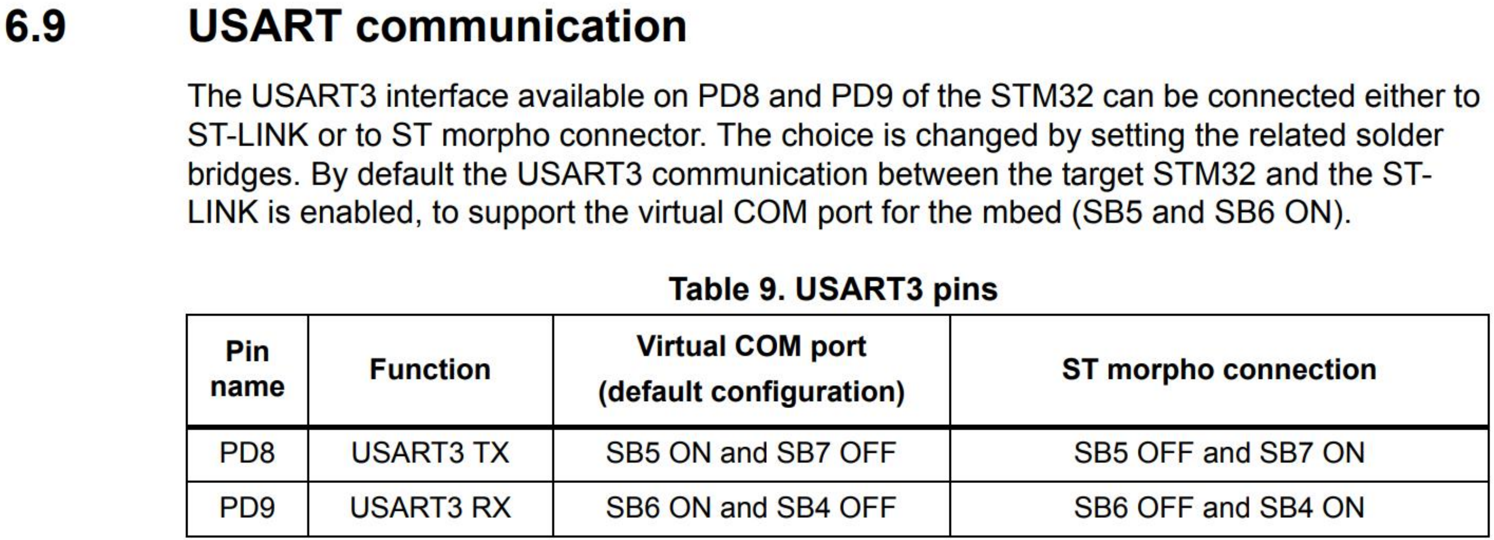
3. Configuration of the GPIO pins required for the communication. In this example we use USART3 since the RX and TX pins PD8 and PD9 are directly connected to the Nucleo ST-Link (the upper part of the board that allows the communication between the uC and the PC) by default. For other configurations have a look in the manual here.

4. Initialization of the interrupts and priorities.



```c
/* USART3 init function */

void MX_USART3_UART_Init(void)
{
  LL_USART_InitTypeDef USART_InitStruct = {0};

  LL_GPIO_InitTypeDef GPIO_InitStruct = {0};

  /* Peripheral clock enable */
  LL_APB1_GRP1_EnableClock(LL_APB1_GRP1_PERIPH_USART3);

  LL_AHB1_GRP1_EnableClock(LL_AHB1_GRP1_PERIPH_GPIOD);
  /**USART3 GPIO Configuration
  PD8   ------> USART3_TX
  PD9   ------> USART3_RX
  */
  GPIO_InitStruct.Pin = STLK_RX_Pin|STLK_TX_Pin;
  GPIO_InitStruct.Mode = LL_GPIO_MODE_ALTERNATE;
  GPIO_InitStruct.Speed = LL_GPIO_SPEED_FREQ_VERY_HIGH;
  GPIO_InitStruct.OutputType = LL_GPIO_OUTPUT_PUSHPULL;
  GPIO_InitStruct.Pull = LL_GPIO_PULL_NO;
  GPIO_InitStruct.Alternate = LL_GPIO_AF_7;
  LL_GPIO_Init(GPIOD, &GPIO_InitStruct);

  /* USART3 interrupt Init */
  NVIC_SetPriority(USART3_IRQn, NVIC_EncodePriority(NVIC_GetPriorityGrouping(),0, 0));
  NVIC_EnableIRQ(USART3_IRQn);

  USART_InitStruct.BaudRate = 9600;
  USART_InitStruct.DataWidth = LL_USART_DATAWIDTH_8B;
  USART_InitStruct.StopBits = LL_USART_STOPBITS_1;
  USART_InitStruct.Parity = LL_USART_PARITY_NONE;
  USART_InitStruct.TransferDirection = LL_USART_DIRECTION_TX_RX;
  USART_InitStruct.HardwareFlowControl = LL_USART_HWCONTROL_NONE;
  USART_InitStruct.OverSampling = LL_USART_OVERSAMPLING_16;
  LL_USART_Init(USART3, &USART_InitStruct);
  LL_USART_ConfigAsyncMode(USART3);
  LL_USART_Enable(USART3);

}

/* USER CODE BEGIN 1 */
```

Usart configuration



6.9 USART communication

The USART3 interface available on PD8 and PD9 of the STM32 can be connected either to ST-LINK or to ST morpho connector. The choice is changed by setting the related solder bridges. By default the USART3 communication between the target STM32 and the ST-LINK is enabled, to support the virtual COM port for the mbed (SB5 and SB6 ON).

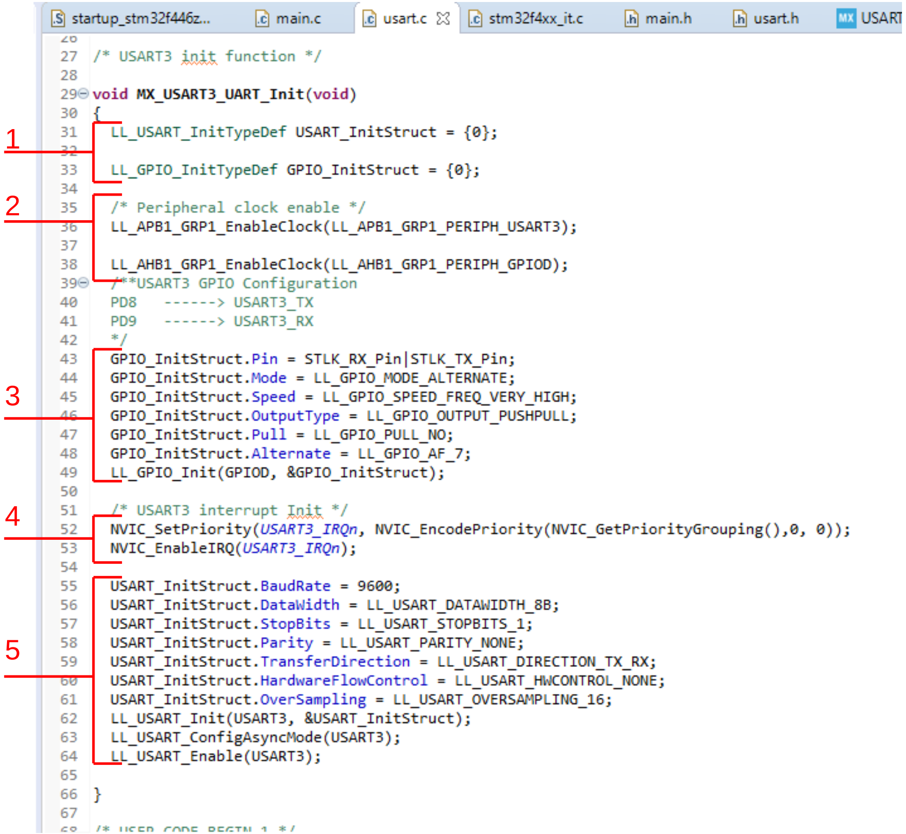
Table 9. USART3 pins

Pin name	Function	Virtual COM port (default configuration)	ST morpho connection
PD8	USART3 TX	SB5 ON and SB7 OFF	SB5 OFF and SB7 ON
PD9	USART3 RX	SB6 ON and SB4 OFF	SB6 OFF and SB4 ON

USART Configuration

5. Configuration of the USART peripheral. Here it’s possible to see some values like the *BaudRate*, *StopBits*, *Parity, ecc*.

The last step is to enable the USART interrupt: with the function ***LL_USART_EnableIT_RXNE()*** we enable the *RX Not Empty Interrupt*, so the interrupt triggers only if the received message via UART is not empty.

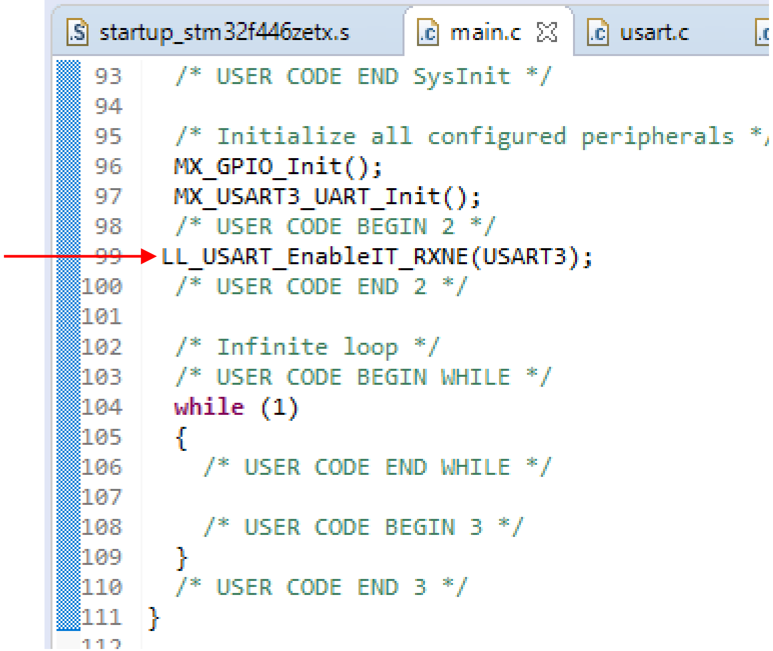


```c
/* USART3 init function */

void MX_USART3_UART_Init(void)
{
  LL_USART_InitTypeDef USART_InitStruct = {0};

  LL_GPIO_InitTypeDef GPIO_InitStruct = {0};

  /* Peripheral clock enable */
  LL_APB1_GRP1_EnableClock(LL_APB1_GRP1_PERIPH_USART3);

  LL_AHB1_GRP1_EnableClock(LL_AHB1_GRP1_PERIPH_GPIOD);
  /**USART3 GPIO Configuration
  PD8   ------> USART3_TX
  PD9   ------> USART3_RX
  */
  GPIO_InitStruct.Pin = STLK_RX_Pin|STLK_TX_Pin;
  GPIO_InitStruct.Mode = LL_GPIO_MODE_ALTERNATE;
  GPIO_InitStruct.Speed = LL_GPIO_SPEED_FREQ_VERY_HIGH;
  GPIO_InitStruct.OutputType = LL_GPIO_OUTPUT_PUSHPULL;
  GPIO_InitStruct.Pull = LL_GPIO_PULL_NO;
  GPIO_InitStruct.Alternate = LL_GPIO_AF_7;
  LL_GPIO_Init(GPIOD, &GPIO_InitStruct);

  /* USART3 interrupt Init */
  NVIC_SetPriority(USART3_IRQn, NVIC_EncodePriority(NVIC_GetPriorityGrouping(),0, 0));
  NVIC_EnableIRQ(USART3_IRQn);

  USART_InitStruct.BaudRate = 9600;
  USART_InitStruct.DataWidth = LL_USART_DATAWIDTH_8B;
  USART_InitStruct.StopBits = LL_USART_STOPBITS_1;
  USART_InitStruct.Parity = LL_USART_PARITY_NONE;
  USART_InitStruct.TransferDirection = LL_USART_DIRECTION_TX_RX;
  USART_InitStruct.HardwareFlowControl = LL_USART_HWCONTROL_NONE;
  USART_InitStruct.OverSampling = LL_USART_OVERSAMPLING_16;
  LL_USART_Init(USART3, &USART_InitStruct);
  LL_USART_ConfigAsyncMode(USART3);
  LL_USART_Enable(USART3);

}

/* USER CODE BEGIN 1 */
```

Enable interrupt

The last step is to enable the USART interrupt: with the function ***LL_USART_EnableIT_RXNE()*** we enable the *RX Not Empty Interrupt*, so the interrupt triggers only if the received message via UART is not empty.

Do it in the *main.c* file, after the peripherals initializations.



```
  /* USER CODE END SysInit */

  /* Initialize all configured peripherals *
  MX_GPIO_Init();
  MX_USART3_UART_Init();
  /* USER CODE BEGIN 2 */
 LL_USART_EnableIT_RXNE(USART3);
  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
    /* USER CODE END WHILE */

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

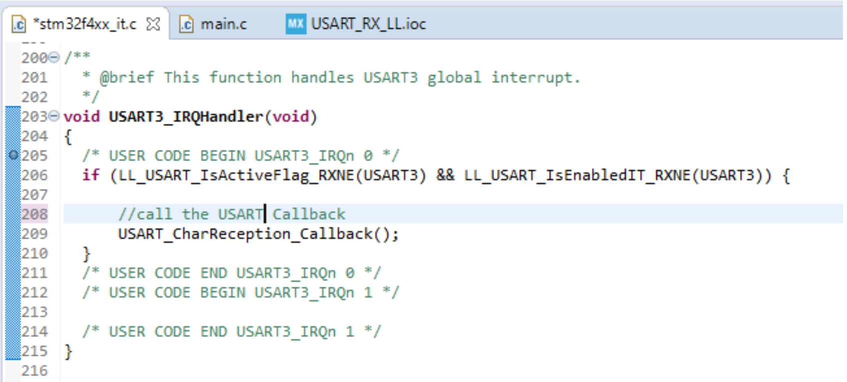
USART Handler

When the interrupt is triggered, the program calls the *USART handler*, defined in the ***stm32f4xx_it.c*** file.

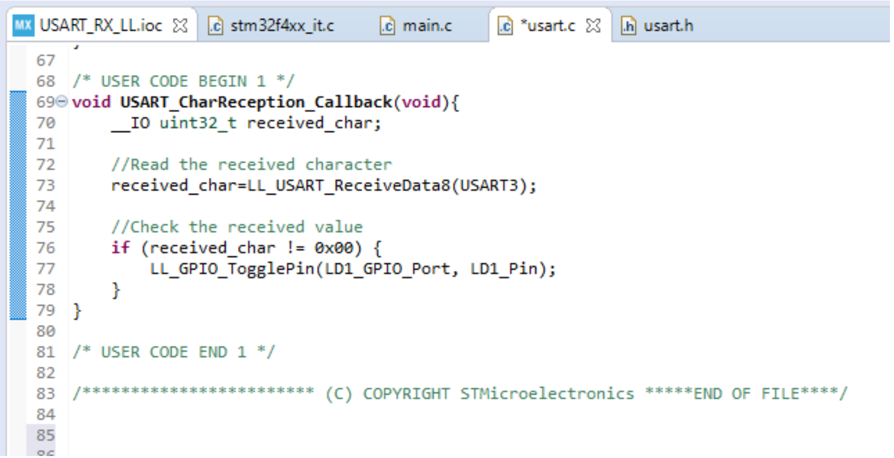
At this point we check if the interrupt has been generated correctly and eventually calls the ***USART_CharReception_Callback()*** that we now define in the *usart.c* file.

Via the ***LL_USART_ReceiveData8()*** we read 1byte of data from the UART bus and put it the *received_char* variable.

For this example when we receive some data, the LED will toggle. In this case we are comparing the received value with an exadecimal value but take in mind that you can impose any condition you want.



```c
/**
  * @brief This function handles USART3 global interrupt.
  */
void USART3_IRQHandler(void)
{
  /* USER CODE BEGIN USART3_IRQn 0 */
  if (LL_USART_IsActiveFlag_RXNE(USART3) && LL_USART_IsEnabledIT_RXNE(USART3)) {

      //call the USART Callback
      USART_CharReception_Callback();
  }
  /* USER CODE END USART3_IRQn 0 */
  /* USER CODE BEGIN USART3_IRQn 1 */

  /* USER CODE END USART3_IRQn 1 */
}

```

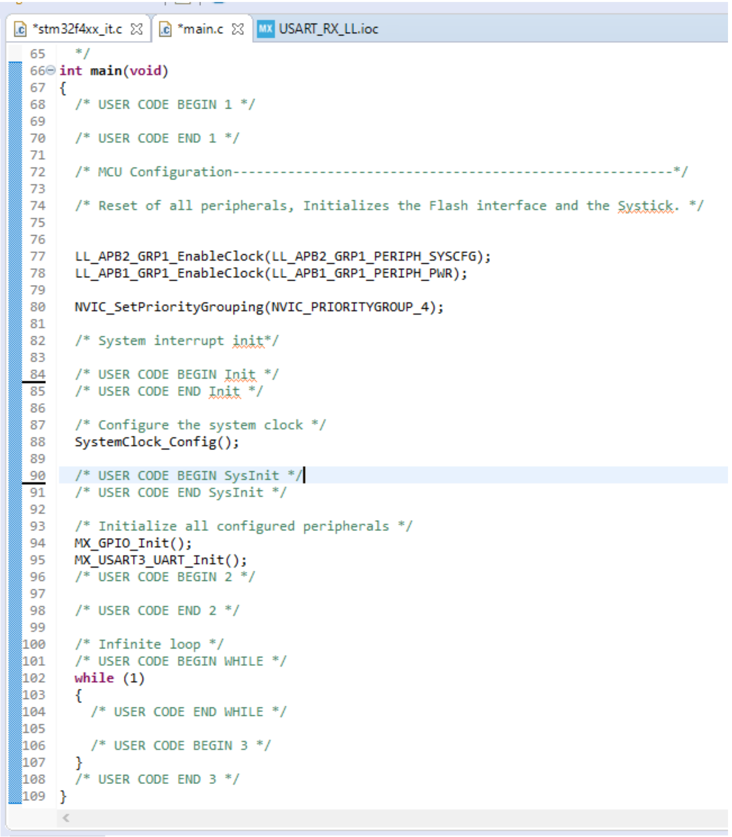


```c

/* USER CODE BEGIN 1 */
void USART_CharReception_Callback(void){
    __IO uint32_t received_char;

    //Read the received character
    received_char=LL_USART_ReceiveData8(USART3);

    //Check the received value
    if (received_char != 0x00) {
        LL_GPIO_TogglePin(LD1_GPIO_Port, LD1_Pin);
    }
}

/* USER CODE END 1 */

/************************ (C) COPYRIGHT STMicroelectronics *****END OF FILE****/

```

Empty Loop

Since our programs manage all the functions via the interrupts, the main loop is empty.

The coding part ends here, now simply download the program to your board.



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

  /* USER CODE END 1 */

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

  /* Reset of all peripherals, Initializes the Flash interface and the Systick. */


  LL_APB2_GRP1_EnableClock(LL_APB2_GRP1_PERIPH_SYSCFG);
  LL_APB1_GRP1_EnableClock(LL_APB1_GRP1_PERIPH_PWR);

  NVIC_SetPriorityGrouping(NVIC_PRIORITYGROUP_4);

  /* System interrupt 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_USART3_UART_Init();
  /* USER CODE BEGIN 2 */

  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
    /* USER CODE END WHILE */

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

Send data via UART via PC

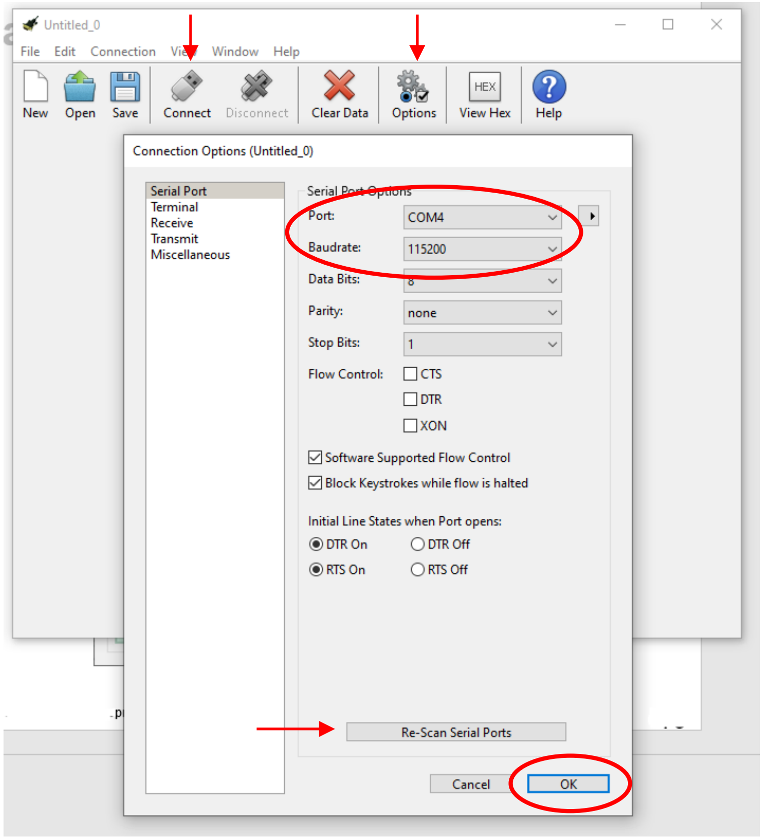
In this example we send the UART data via PC. You can use any Serial interface program you want, in this case we will use CoolTerm (link).

Run the program and select the correct settings in order to communicate with the board.

Remeber to select the correct *COM Port* and *BaudRate.*

Once done, click on *Connect.*

If you have troubles finding your board COM Port in the COM Port list try to click *Re-Scan Serial Ports* and check again.



Connection Options (Untitled_0)

Serial Port Options

Port: COM4

Baudrate: 115200

Data Bits: 8

Parity: none

Stop Bits: 1

Flow Control: □ CTS □ DTR □ XON

☑ Software Supported Flow Control

☑ Block Keystrokes while flow is halted

Initial Line States when Port opens:

● DTR On ○ DTR Off

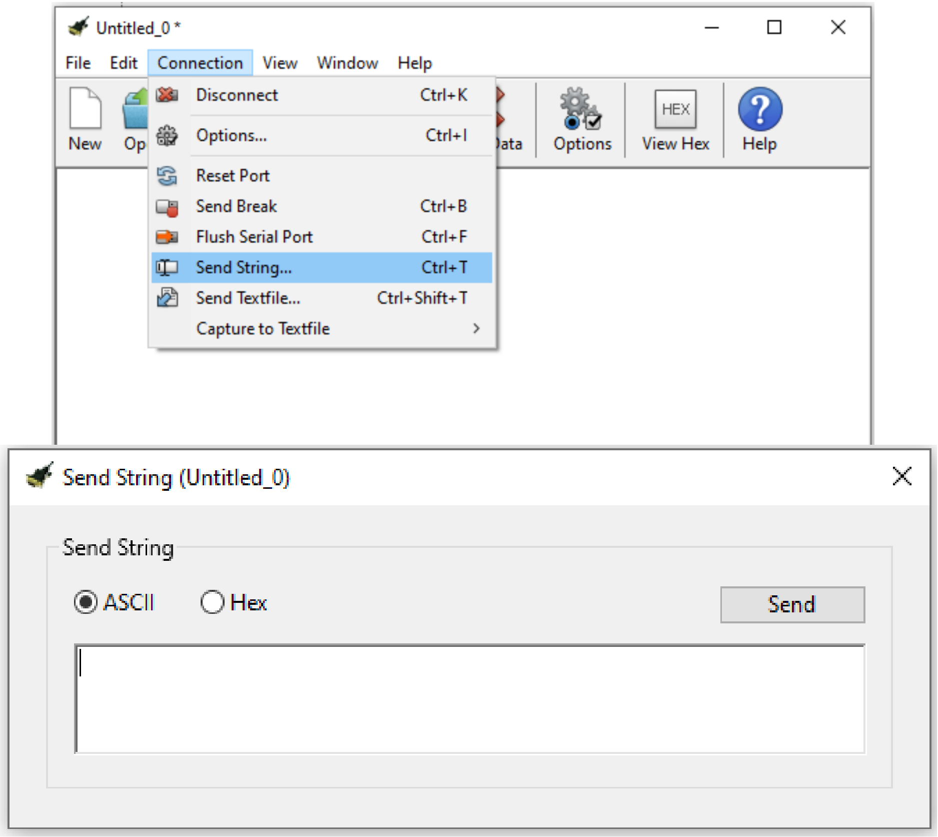
● RTS On ○ RTS Off

Send data via UART

To open the send window click on *connection->Send String.*

From here you can send any string you want: you can choose to send ascii o hexadecimal formatted message.

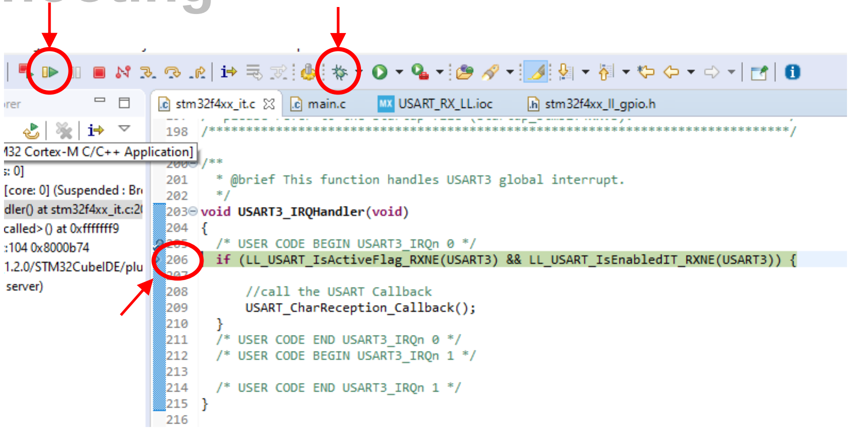
The result is that when you send any message the led will toggle.



Troubleshooting

If you have any trouble getting the program to run correctly try to debug your application in the IDE:

1. Download the program to your board in debug mode, the debug view will open.
2. Place a breakpoint where the interrupt should be triggered (to place a breakpoint double click on the number of line where you want to place it).
3. Click on the resume icon.
4. Try to send a message via CoolTerm.



```
/*****************************************************************************/

/**
  * @brief This function handles USART3 global interrupt.
  */
void USART3_IRQHandler(void)
{
  /* USER CODE BEGIN USART3_IRQn 0 */
  if (LL_USART_IsActiveFlag_RXNE(USART3) && LL_USART_IsEnabledIT_RXNE(USART3)) {

      //call the USART Callback
      USART_CharReception_Callback();
  }
  /* USER CODE END USART3_IRQn 0 */
  /* USER CODE BEGIN USART3_IRQn 1 */

  /* USER CODE END USART3_IRQn 1 */
}
```

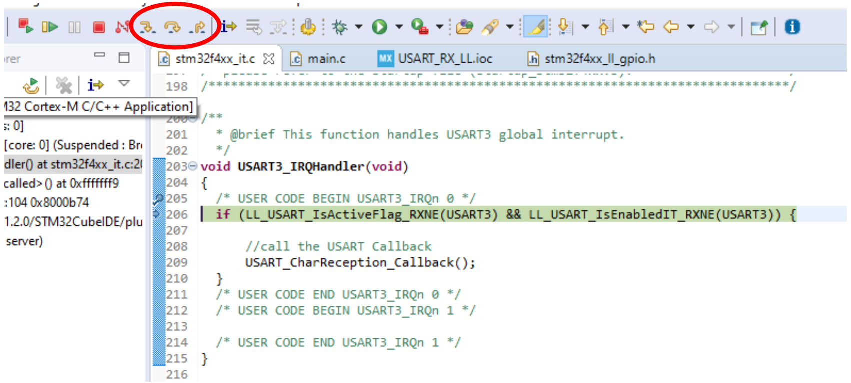
Troubleshooting

5. If all it’s correct your program will jump to the breakpoint and the line will be highlighted.

6. At this point you can resume again or move inside your program with the arrows

In this example we sent 0x87 via CoolTerm, as you can see in the next slide.

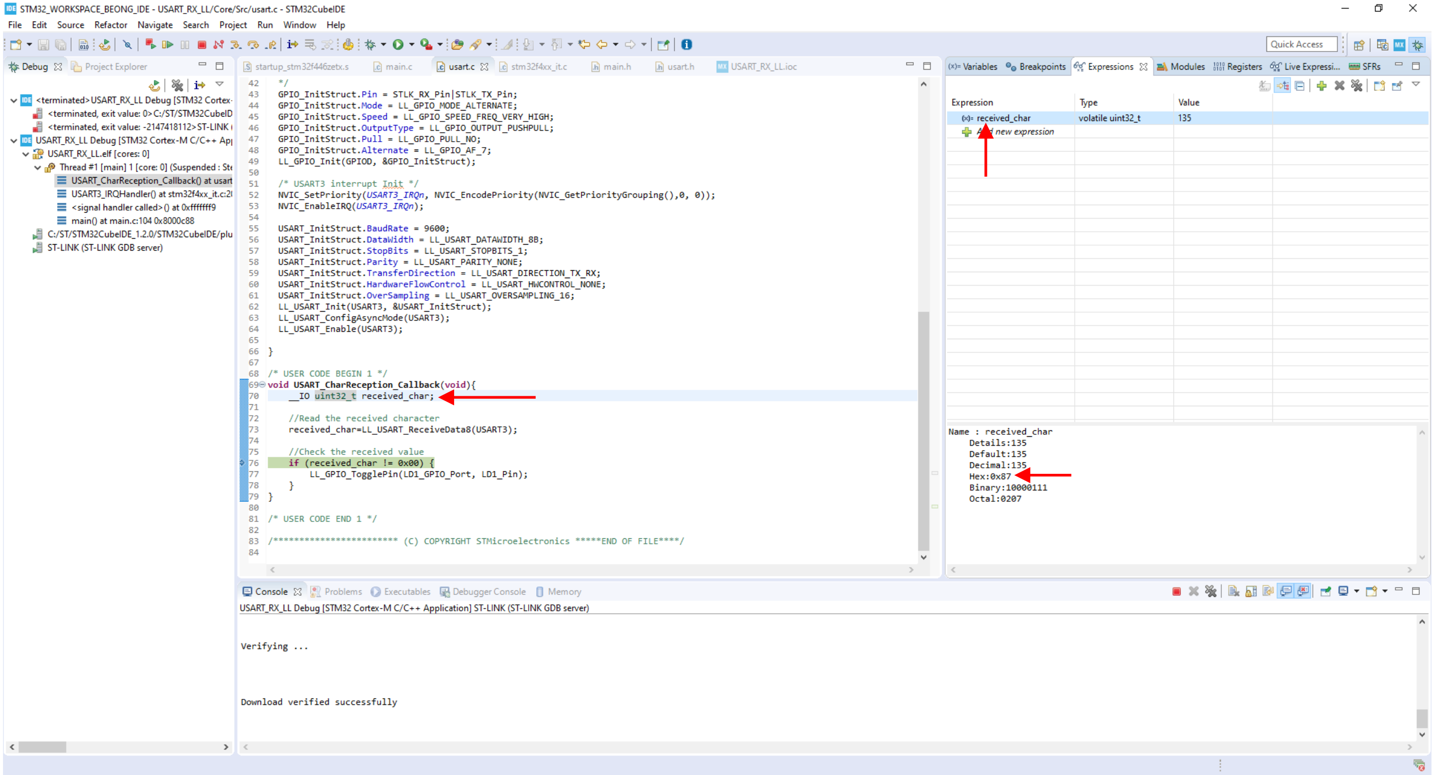
For more info in how the UART bus works see: https://www.youtube.com/watch?v=ZzRXKDkMBhA



```
132 Cortex-M C/C++ Application]
s: 0]
[core: 0] (Suspended : Br
dler() at stm32f4xx_it.c:2
called> () at 0xfffffff9
:104 0x8000b74
1.2.0/STM32CubeIDE/plu
server)
```

```
/*******************************************************************************/

/**
  * @brief This function handles USART3 global interrupt.
  */
void USART3_IRQHandler(void)
{
  /* USER CODE BEGIN USART3_IRQn 0 */
  if (LL_USART_IsActiveFlag_RXNE(USART3) && LL_USART_IsEnabledIT_RXNE(USART3)) {

      //call the USART Callback
      USART_CharReception_Callback();
  }
  /* USER CODE END USART3_IRQn 0 */
  /* USER CODE BEGIN USART3_IRQn 1 */

  /* USER CODE END USART3_IRQn 1 */
}
```

Troubleshooting



```
<terminated>USART_RX_LL Debug [STM32 Cortex-
    <terminated, exit value: 0>C:/ST/STM32CubeID
    <terminated, exit value: -2147418112>ST-LINK (
USART_RX_LL Debug [STM32 Cortex-M C/C++ Ap|
  USART_RX_LL.elf [cores: 0]
    Thread #1 [main] 1 [core: 0] (Suspended : Ste
      USART_CharReception_Callback() at usart
      USART3_IRQHandler() at stm32f4xx_it.c:2(
      <signal handler called>() at 0xfffffff9
      main() at main.c:104 0x8000c88
  C:/ST/STM32CubeIDE_1.2.0/STM32CubeIDE/plu
  ST-LINK (ST-LINK GDB server)
```

```c
  */
  GPIO_InitStruct.Pin = STLK_RX_Pin|STLK_TX_Pin;
  GPIO_InitStruct.Mode = LL_GPIO_MODE_ALTERNATE;
  GPIO_InitStruct.Speed = LL_GPIO_SPEED_FREQ_VERY_HIGH;
  GPIO_InitStruct.OutputType = LL_GPIO_OUTPUT_PUSHPULL;
  GPIO_InitStruct.Pull = LL_GPIO_PULL_NO;
  GPIO_InitStruct.Alternate = LL_GPIO_AF_7;
  LL_GPIO_Init(GPIOD, &GPIO_InitStruct);

  /* USART3 interrupt Init */
  NVIC_SetPriority(USART3_IRQn, NVIC_EncodePriority(NVIC_GetPriorityGrouping(),0, 0));
  NVIC_EnableIRQ(USART3_IRQn);

  USART_InitStruct.BaudRate = 9600;
  USART_InitStruct.DataWidth = LL_USART_DATAWIDTH_8B;
  USART_InitStruct.StopBits = LL_USART_STOPBITS_1;
  USART_InitStruct.Parity = LL_USART_PARITY_NONE;
  USART_InitStruct.TransferDirection = LL_USART_DIRECTION_TX_RX;
  USART_InitStruct.HardwareFlowControl = LL_USART_HWCONTROL_NONE;
  USART_InitStruct.OverSampling = LL_USART_OVERSAMPLING_16;
  LL_USART_Init(USART3, &USART_InitStruct);
  LL_USART_ConfigAsyncMode(USART3);
  LL_USART_Enable(USART3);

}

/* USER CODE BEGIN 1 */
void USART_CharReception_Callback(void){
    __IO uint32_t received_char;

    //Read the received character
    received_char=LL_USART_ReceiveData8(USART3);

    //Check the received value
    if (received_char != 0x00) {
        LL_GPIO_TogglePin(LD1_GPIO_Port, LD1_Pin);
    }
}

/* USER CODE END 1 */

/************************ (C) COPYRIGHT STMicroelectronics *****END OF FILE****/
```

Expression	Type	Value
received_char	volatile uint32_t	135
Add new expression		

```
Name : received_char
    Details:135
    Default:135
    Decimal:135
    Hex:0x87
    Binary:10000111
    Octal:0207
```

USART_RX_LL Debug [STM32 Cortex-M C/C++ Application] ST-LINK (ST-LINK GDB server)

```
Verifying ...


Download verified successfully
```

Select received_char and *rightClick -> add to watch expression* to monitor it.