PYSERIAL:

```
import serial
import time

# ------------------------
# Configure your port
# ------------------------
PORT = "COM15"        # Change to your STM32's COM port
BAUD = 115200

# Four 9-byte frames (8 chars + '\n')
frames = [
    b"+200+200\n",
    b"+150+100\n",
    b"+100-050\n",
    b"-200+000\n"
]

# ------------------------
# Open serial connection
# ------------------------
ser = serial.Serial(PORT, BAUD, timeout=1)
time.sleep(2)  # give STM32 time to reset after opening port

print(f"[OPEN] {PORT} @ {BAUD} baud")
print("Transmitting continuously... Press Ctrl+C to stop.\n")

try:
    i = 0
    while True:
        # Send one frame
```

```
        frame = frames[i % len(frames)]
        ser.write(frame)
        ser.flush()
        print(f"Sent: {frame.decode().strip()}")
        i += 1

        # Adjust rate — 10 ms between frames (~100 Hz)
        time.sleep(1)

except KeyboardInterrupt:
    print("\nStopped by user.")
finally:
    ser.close()
    print("Port closed.")
```

CAN_TX OR PC SIDE:

```
/* USER CODE BEGIN Header */

/**

 ******************************************************************************

 * @file : main.c

 * @brief : Main program body

 ******************************************************************************

 * @attention
```

```
*
* Copyright (c) 2025 STMicroelectronics.
* All rights reserved.
*
* This software is licensed under terms that can be found in the LICENSE file
* in the root directory of this software component.
* If no LICENSE file comes with this software, it is provided AS-IS.
*
******************************************************************************
*/
/* USER CODE END Header */
/* Includes ------------------------------------------------------------------*/
#include "main.h"


/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */
```

```
/* USER CODE END Includes */

/* Private typedef -----------------------------------------------------------*/

/* USER CODE BEGIN PTD */

/* USER CODE END PTD */

/* Private define ------------------------------------------------------------*/

/* USER CODE BEGIN PD */

/* USER CODE END PD */

/* Private macro -------------------------------------------------------------*/

/* USER CODE BEGIN PM */
```

```
/* USER CODE END PM */



/* Private variables ---------------------------------------------------------*/

CAN_HandleTypeDef hcan;



UART_HandleTypeDef huart1;

DMA_HandleTypeDef hdma_usart1_rx;



/* USER CODE BEGIN PV */

#define RX_BUF_SIZE 18 // two 9B frames: [0..8] and [9..17]

uint8_t rx_buf[RX_BUF_SIZE];



static CAN_TxHeaderTypeDef tx;

static uint32_t canMailbox;
```

```
/* USER CODE END PV */

/* Private function prototypes -----------------------------------------------*/

void SystemClock_Config(void);

static void MX_GPIO_Init(void);

static void MX_DMA_Init(void);

static void MX_USART1_UART_Init(void);

static void MX_CAN_Init(void);

/* USER CODE BEGIN PFP */

/* USER CODE END PFP */

/* Private user code ---------------------------------------------------------*/

/* USER CODE BEGIN 0 */
```

```
/* USER CODE END 0 */



/**

 * @brief The application entry point.

 * @retval int

 */

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_DMA_Init();

 MX_USART1_UART_Init();

 MX_CAN_Init();

/* USER CODE BEGIN 2 */




 HAL_UART_Receive_DMA(&huart1, rx_buf, RX_BUF_SIZE);



// Accept-all CAN filter to FIFO0 (needed even in loopback)




// Start CAN and enable RX FIFO0 notification (for loopback receive)

 HAL_CAN_Start(&hcan);



 tx.StdId = 0x100; // your CAN ID
```

```
tx.ExtId = 0;

tx.IDE = CAN_ID_STD;

tx.RTR = CAN_RTR_DATA;

tx.DLC = 8;

tx.TransmitGlobalTime = DISABLE;




/* USER CODE END 2 */



/* Infinite loop */

/* USER CODE BEGIN WHILE */

while (1)

{

/* USER CODE END WHILE */
```

```
/* USER CODE BEGIN 3 */

// nothing; all work happens in IRQ/DMAs

 }

/* USER CODE END 3 */

}



/**

 * @brief System Clock Configuration

 * @retval None

 */

void SystemClock_Config(void)

{

RCC_OscInitTypeDef RCC_OscInitStruct = {0};

RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};



/** Initializes the RCC Oscillators according to the specified parameters
```

```
* in the RCC_OscInitTypeDef structure.

*/

RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSE;

RCC_OscInitStruct.HSEState = RCC_HSE_ON;

RCC_OscInitStruct.HSEPredivValue = RCC_HSE_PREDIV_DIV1;

RCC_OscInitStruct.HSIState = RCC_HSI_ON;

RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;

RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;

RCC_OscInitStruct.PLL.PLLMUL = RCC_PLL_MUL9;

if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)

{

Error_Handler();

}



/** Initializes the CPU, AHB and APB buses clocks
```

```
 */

 RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK

 |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;

 RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;

 RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;

 RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV2;

 RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;



if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_2) != HAL_OK)

 {

 Error_Handler();

 }

}



/**

 * @brief CAN Initialization Function
```

```
 * @param None

 * @retval None

 */

static void MX_CAN_Init(void)

{



/* USER CODE BEGIN CAN_Init 0 */

/* USER CODE END CAN_Init 0 */



/* USER CODE BEGIN CAN_Init 1 */

/* USER CODE END CAN_Init 1 */

 hcan.Instance = CAN1;

 hcan.Init.Prescaler = 2;

 hcan.Init.Mode = CAN_MODE_NORMAL;

 hcan.Init.SyncJumpWidth = CAN_SJW_1TQ;
```

```
 hcan.Init.TimeSeg1 = CAN_BS1_15TQ;

 hcan.Init.TimeSeg2 = CAN_BS2_2TQ;

 hcan.Init.TimeTriggeredMode = DISABLE;

 hcan.Init.AutoBusOff = ENABLE;

 hcan.Init.AutoWakeUp = ENABLE;

 hcan.Init.AutoRetransmission = ENABLE;

 hcan.Init.ReceiveFifoLocked = DISABLE;

 hcan.Init.TransmitFifoPriority = DISABLE;

if (HAL_CAN_Init(&hcan) != HAL_OK)

 {

 Error_Handler();

 }

/* USER CODE BEGIN CAN_Init 2 */

/* USER CODE END CAN_Init 2 */



}
```

```
/**

 * @brief USART1 Initialization Function

 * @param None

 * @retval None

 */

static void MX_USART1_UART_Init(void)

{



/* USER CODE BEGIN USART1_Init 0 */

/* USER CODE END USART1_Init 0 */



/* USER CODE BEGIN USART1_Init 1 */

/* USER CODE END USART1_Init 1 */

 huart1.Instance = USART1;
```

```
 huart1.Init.BaudRate = 115200;

 huart1.Init.WordLength = UART_WORDLENGTH_8B;

 huart1.Init.StopBits = UART_STOPBITS_1;

 huart1.Init.Parity = UART_PARITY_NONE;

 huart1.Init.Mode = UART_MODE_TX_RX;

 huart1.Init.HwFlowCtl = UART_HWCONTROL_NONE;

 huart1.Init.OverSampling = UART_OVERSAMPLING_16;

if (HAL_UART_Init(&huart1) != HAL_OK)

 {

 Error_Handler();

 }

/* USER CODE BEGIN USART1_Init 2 */

/* USER CODE END USART1_Init 2 */



}
```

```
/**

 * Enable DMA controller clock

 */

static void MX_DMA_Init(void)

{



/* DMA controller clock enable */

__HAL_RCC_DMA1_CLK_ENABLE();



/* DMA interrupt init */

/* DMA1_Channel5_IRQn interrupt configuration */

 HAL_NVIC_SetPriority(DMA1_Channel5_IRQn, 0, 0);

 HAL_NVIC_EnableIRQ(DMA1_Channel5_IRQn);



}
```

```
/**

 * @brief GPIO Initialization Function

 * @param None

 * @retval None

 */

static void MX_GPIO_Init(void)

{

/* USER CODE BEGIN MX_GPIO_Init_1 */

/* USER CODE END MX_GPIO_Init_1 */



/* GPIO Ports Clock Enable */

__HAL_RCC_GPIOD_CLK_ENABLE();

__HAL_RCC_GPIOA_CLK_ENABLE();



/* USER CODE BEGIN MX_GPIO_Init_2 */
```

```
/* USER CODE END MX_GPIO_Init_2 */

}



/* USER CODE BEGIN 4 */



// UART DMA HALF: indices [0..8] (9 bytes: 8 data + '\n')

void HAL_UART_RxHalfCpltCallback(UART_HandleTypeDef *huart)

{

if (huart == &huart1)

 {

if (rx_buf[8] == '\n')

 {

uint8_t d1[8];

for (int i = 0; i < 8; ++i) d1[i] = rx_buf[i];

 (void)HAL_CAN_AddTxMessage(&hcan, &tx, d1, &canMailbox);
```

```
 }

 }

}



// UART DMA FULL: indices [9..17] (9 bytes: 8 data + '\n')

void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart)

{

if (huart == &huart1)

 {

if (rx_buf[17] == '\n')

 {

uint8_t d2[8];

for (int i = 0; i < 8; ++i) d2[i] = rx_buf[9 + i];

 (void)HAL_CAN_AddTxMessage(&hcan, &tx, d2, &canMailbox);

 }

 }
```

```
}
```

```
/* USER CODE END 4 */
```

```
/**
 * @brief This function is executed in case of error occurrence.
 * @retval None
 */
void Error_Handler(void)
{
/* USER CODE BEGIN Error_Handler_Debug */
 __disable_irq();
while (1) { }
```

```
/* USER CODE END Error_Handler_Debug */

}

#ifdef USE_FULL_ASSERT

/**

 * @brief Reports the name of the source file and the source line number

 * where the assert_param error has occurred.

 * @param file: pointer to the source file name

 * @param line: assert_param error line source number

 * @retval None

 */

void assert_failed(uint8_t *file, uint32_t line)

{

/* USER CODE BEGIN 6 */

/* USER CODE END 6 */

}

#endif /* USE_FULL_ASSERT */
```

CAN_RX :

```
/* USER CODE BEGIN Header */
/**
 ******************************************************************************
 * @file : main.c
 * @brief : Main program body (CAN RX → UART + '\n')
 ******************************************************************************
 * @attention
 * Copyright (c) 2025 ST.
 * Licensed as-is.
 ******************************************************************************
 */
/* USER CODE END Header */
```

```
#include "main.h"



/* Handles -----------------------------------------------------------------*/

CAN_HandleTypeDef hcan;

UART_HandleTypeDef huart1;



/* USER CODE BEGIN PV */

static CAN_RxHeaderTypeDef canRx;

static uint8_t rxData[8];

/* USER CODE END PV */



/* Prototypes --------------------------------------------------------------*/

void SystemClock_Config(void);

static void MX_GPIO_Init(void);

static void MX_CAN_Init(void);

static void MX_USART1_UART_Init(void);
```

```
/* USER CODE BEGIN 0 */

/* USER CODE END 0 */



int main(void)

{

 HAL_Init();

 SystemClock_Config();

 MX_GPIO_Init();

 MX_CAN_Init();

 MX_USART1_UART_Init();



/* USER CODE BEGIN 2 */

// Filter: accept only StdID 0x100 into FIFO0

 {
```

```
CAN_FilterTypeDef f = {0};

uint32_t id = (0x100U << 21); // StdID at bits 21..31

uint32_t mask = (0x7FFU << 21); // match all 11 bits



 f.FilterBank = 0;

 f.FilterMode = CAN_FILTERMODE_IDMASK;

 f.FilterScale = CAN_FILTERSCALE_32BIT;

 f.FilterIdHigh = (uint16_t)(id >> 16);

 f.FilterIdLow = (uint16_t)(id & 0xFFFF);

 f.FilterMaskIdHigh = (uint16_t)(mask >> 16);

 f.FilterMaskIdLow = (uint16_t)(mask & 0xFFFF);

 f.FilterFIFOAssignment = CAN_RX_FIFO0;

 f.FilterActivation = ENABLE;

 f.SlaveStartFilterBank = 14; // default split point on F1
```

```
 HAL_CAN_ConfigFilter(&hcan, &f);

 }



// Start CAN and enable RX FIFO0 interrupt

 HAL_CAN_Start(&hcan);

 HAL_CAN_ActivateNotification(&hcan, CAN_IT_RX_FIFO0_MSG_PENDING);

/* USER CODE END 2 */



while (1)

 {

// nothing here; RX handled in callback

 }

}



/* Clock ------------------------------------------------------------------*/
```

```
void SystemClock_Config(void)

{

RCC_OscInitTypeDef RCC_OscInitStruct = {0};

RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};



 RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSE;

 RCC_OscInitStruct.HSEState = RCC_HSE_ON;

 RCC_OscInitStruct.HSEPredivValue = RCC_HSE_PREDIV_DIV1;



 RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;

 RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;

 RCC_OscInitStruct.PLL.PLLMUL = RCC_PLL_MUL9;

if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK) Error_Handler();



 RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK|

 RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
```

```
 RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;

 RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;

 RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV2; // 36 MHz

 RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;

if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_2) != HAL_OK) Error_Handler();

}



/* CAN ----------------------------------------------------------------------*/

static void MX_CAN_Init(void)

{

 hcan.Instance = CAN1;



// 1 Mbps @ APB1=36 MHz: Presc=2, BS1=15TQ, BS2=2TQ (36/(2*(1+15+2))=1e6)

 hcan.Init.Prescaler = 2;

 hcan.Init.Mode = CAN_MODE_NORMAL; // NORMAL on real bus
```

```
 hcan.Init.SyncJumpWidth = CAN_SJW_1TQ;

 hcan.Init.TimeSeg1 = CAN_BS1_15TQ;

 hcan.Init.TimeSeg2 = CAN_BS2_2TQ;

 hcan.Init.TimeTriggeredMode = DISABLE;

 hcan.Init.AutoBusOff = ENABLE;

 hcan.Init.AutoWakeUp = ENABLE;

 hcan.Init.AutoRetransmission = ENABLE;

 hcan.Init.ReceiveFifoLocked = DISABLE;

 hcan.Init.TransmitFifoPriority= DISABLE;

if (HAL_CAN_Init(&hcan) != HAL_OK) Error_Handler();



// NVIC for CAN1 RX0 IRQ (FIFO0 message pending)

 HAL_NVIC_SetPriority(USB_LP_CAN1_RX0_IRQn, 0, 0);

 HAL_NVIC_EnableIRQ(USB_LP_CAN1_RX0_IRQn);

}
```

```
/* UART ----------------------------------------------------------------*/

static void MX_USART1_UART_Init(void)

{

 huart1.Instance = USART1;

 huart1.Init.BaudRate = 115200;

 huart1.Init.WordLength = UART_WORDLENGTH_8B;

 huart1.Init.StopBits = UART_STOPBITS_1;

 huart1.Init.Parity = UART_PARITY_NONE;

 huart1.Init.Mode = UART_MODE_TX_RX;

 huart1.Init.HwFlowCtl = UART_HWCONTROL_NONE;

 huart1.Init.OverSampling = UART_OVERSAMPLING_16;

if (HAL_UART_Init(&huart1) != HAL_OK) Error_Handler();

}



/* GPIO ----------------------------------------------------------------*/
```

```
static void MX_GPIO_Init(void)

{

 __HAL_RCC_GPIOD_CLK_ENABLE();

 __HAL_RCC_GPIOA_CLK_ENABLE();

}



/* === CAN RX → UART+'\n' callback ==========================================*/

void HAL_CAN_RxFifo0MsgPendingCallback(CAN_HandleTypeDef *h)

{

uint8_t newline = '\n';



if (HAL_CAN_GetRxMessage(h, CAN_RX_FIFO0, &canRx, rxData) != HAL_OK)

return;



 HAL_UART_Transmit(&huart1, rxData, canRx.DLC, HAL_MAX_DELAY);

 HAL_UART_Transmit(&huart1, &newline, 1, HAL_MAX_DELAY);
```

```
}



/* Error ----------------------------------------------------------------*/

void Error_Handler(void)

{

 __disable_irq();

while (1) { }

}



#ifdef USE_FULL_ASSERT

void assert_failed(uint8_t *file, uint32_t line)

{

 (void)file; (void)line;

}

#endif
```