Micro-ROS Sensor Node

on Renesas EK-RA6M5

Complete Build, Debug, and Deployment Guide

Marcela Campassi
GitHub: mccampassi

Spring 2026
Supervisor: Dr. Betts

Table of Contents

Part I: What Is Micro-ROS?

1. The Problem Micro-ROS Solves

1.1 What Is ROS 2?

ROS 2 (Robot Operating System 2) is an open-source framework used throughout robotics, autonomous vehicles, industrial automation, and research. It is not an operating system in the traditional sense. It is a communication framework: a standardized way for software components (called "nodes") to discover each other, exchange data, and coordinate behavior across a network.

A ROS 2 system is composed of nodes that publish and subscribe to named data channels called "topics." A temperature sensor node might publish to a topic called /sensors/room_temp. A climate control node might subscribe to that same topic and use the data to decide whether to turn on a heater. Neither node needs to know where the other is running — they could be on the same computer, on different computers, or on different continents. ROS 2 handles the discovery and transport automatically through a middleware layer called DDS (Data Distribution Service).

ROS 2 runs on full-featured operating systems like Ubuntu Linux. It requires a POSIX-compatible environment, dynamic memory allocation, multithreading, a network stack, and typically several hundred megabytes of RAM. A standard ROS 2 installation consumes gigabytes of disk space.

1.2 The Microcontroller Problem

Microcontrollers are the opposite of that environment. A typical microcontroller like the Renesas RA6M5 used in this project has 512 KB of SRAM and 2 MB of flash storage. It runs bare-metal code or a lightweight RTOS like FreeRTOS. It has no operating system in the traditional sense, no filesystem, no network stack beyond what you build yourself, and no concept of processes or dynamic linking.

This creates a fundamental gap. The sensors, actuators, and embedded controllers that robots and industrial systems depend on run on microcontrollers. The high-level decision-making, data logging, visualization, and coordination run on full computers with ROS 2. Getting data from one world to the other traditionally required writing custom serial protocols, custom parsers, custom message formats, and custom bridging code. Every project reinvented this wheel.

1.3 What Micro-ROS Does

Micro-ROS bridges this gap. It is a stripped-down implementation of the ROS 2 client libraries, specifically designed to run on microcontrollers with as little as 32 KB of RAM. It allows a microcontroller to act as a first-class ROS 2 node: it can publish topics, subscribe to topics, offer services, and participate in the ROS 2 ecosystem exactly as if it were a full Linux machine running standard ROS 2.

The key insight is that micro-ROS does not try to run the full DDS middleware on the microcontroller. Instead, it uses a lightweight serialization protocol called XRCE-DDS (eXtremely Resource Constrained Environments DDS). The microcontroller communicates through a thin serial link to a "micro-ROS agent" running on a host computer. The agent

translates between XRCE-DDS and full DDS, making the microcontroller's topics appear on the ROS 2 network as if they originated from a native node.

Key Concept
From the perspective of any other ROS 2 node on the network, there is no difference between a topic published by a desktop computer running Ubuntu and a topic published by a microcontroller running micro-ROS. The data types are the same, the discovery mechanism is the same, the quality-of-service settings are the same. The microcontroller is a full participant in the ROS 2 graph.

2. How Micro-ROS Works

2.1 The Architecture

A micro-ROS system has three layers:

Layer	Runs On	What It Does
Micro-ROS Client	Microcontroller (RA6M5)	Creates nodes, publishers, subscribers, timers. Serializes messages into XRCE-DDS frames. Sends/receives frames over a transport (USB, UART, Wi-Fi, etc.)
Micro-ROS Agent	Host computer (Linux)	Receives XRCE-DDS frames from the serial link. Translates them into full DDS messages. Acts as a proxy for the microcontroller on the ROS 2 network.
ROS 2 Network	Any machine on the network	Standard ROS 2 nodes that subscribe to, publish to, or interact with the microcontroller's topics. Completely unaware that the data originates from a microcontroller.

The communication between the microcontroller and the agent is bidirectional. The agent does not simply relay data — it manages the lifecycle of the microcontroller's ROS 2 entities. When the micro-ROS client creates a publisher, the agent creates a corresponding real DDS publisher on the host side. When the client publishes a message, the agent receives the serialized bytes, deserializes them, and republishes through the real DDS publisher. The reverse happens for subscriptions.

2.2 The Transport Layer

Micro-ROS is transport-agnostic. The client library defines four callback functions: open, close, write, and read. Any physical link that can move bytes can serve as a transport. Common options include UART, USB-CDC, Wi-Fi, Ethernet, and CAN bus.

In this project, the transport is USB-CDC (Communications Device Class). The microcontroller presents itself to the host as a virtual serial port. The agent opens this serial port and exchanges XRCE-DDS frames over it. USB-CDC was chosen because the EK-RA6M5 has a USB Full Speed peripheral, providing higher throughput and lower latency than UART.

2.3 The Client Library Stack

The micro-ROS client running on the microcontroller is composed of several layers, all compiled into a single static library called libmicroros.a:

- rclc (ROS Client Library for C) — the high-level API the application calls. Functions like rclc_publisher_init_default(), rclc_executor_spin_some().
- rcl (ROS Client Library) — the core library managing nodes, publishers, subscribers, timers, and execution.
- rcutils — utility functions (logging, allocators, error handling, string operations).
- rmw_microxrcedds — the middleware layer translating between rcl's internal representation and the XRCE-DDS wire format.
- Micro-XRCE-DDS Client — the serialization/deserialization engine producing binary frames for the transport.
- rosidl type support — generated code defining how each ROS 2 message type (sensor_msgs/Temperature, std_msgs/String, etc.) is serialized.

All are compiled into libmicroros.a, approximately 2–4 MB. The application links against it and calls its API.

2.4 The Agent

The micro-ROS agent is a Linux process on the host computer. It bridges XRCE-DDS and full DDS:

- Opens the serial port (e.g., /dev/ttyACM0) and listens for XRCE-DDS frames.
- When the client creates a node, the agent creates a corresponding DDS participant on the ROS 2 network.
- When the client creates a publisher and topic, the agent creates a matching DDS topic and datawriter.
- When the client publishes a message, the agent deserializes it from XRCE-DDS, converts it to a full DDS message, and publishes it.
- For subscriptions, the reverse: the agent receives full DDS messages, serializes them to XRCE-DDS, and sends them to the microcontroller.

Started with a single command:

```
ros2 run micro_ros_agent micro_ros_agent serial --dev /dev/ttyACM0
```

3. What Micro-ROS Does in This Project

3.1 The Without-Micro-ROS Baseline

Phase A (bare-metal USB version) demonstrates sensor monitoring without micro-ROS. The microcontroller reads three I2C sensors and prints formatted text over USB:

```
Room: 23.4C|74.1F | Hot: 24.1C|75.4F | Cold: 23.8C|74.8F | Hum: 45% |
Door: Opened | Lux: 312
```

This works for basic monitoring but has fundamental limitations:

- Unstructured text. Every consumer must parse the string and handle formatting changes.
- Point-to-point. Only one program (the terminal) can receive data at a time.
- No timestamps, no units metadata, no quality-of-service guarantees, no standard schema.
- No discovery. You must manually configure the COM port and data format.
- Integration requires custom glue code for every consumer (logger, dashboard, controller, ML pipeline).

3.2 What Micro-ROS Changes

Phase B replaces text printing with micro-ROS. The same sensors are read, but structured ROS 2 messages are published to named topics:

Topic	Message Type	Data
/sensors/room_temp	sensor_msgs/msg/Temperature	Temperature in °C, with header timestamp and frame_id
/sensors/humidity	sensor_msgs/msg/RelativeHumidity	Humidity as 0.0–1.0 fraction, with header timestamp
/sensors/thermocouple_hot	sensor_msgs/msg/Temperature	MCP9600 hot junction temperature in °C
/sensors/thermocouple_cold	sensor_msgs/msg/Temperature	MCP9600 cold junction temperature in °C
/sensors/proximity	std_msgs/msg/UInt16	VCNL4010 raw proximity count (higher = closer)
/sensors/ambient_light	std_msgs/msg/UInt16	VCNL4010 raw ambient light count
/sensors/diagnostic	std_msgs/msg/String	Node status messages and error reports

Every Phase A limitation is now solved:

- Structured data. sensor_msgs/Temperature has defined fields. Any ROS 2 program in any language can deserialize it without parsing text.
- Multicast. Any number of nodes can subscribe simultaneously — logger, dashboard, and controller all receive the same stream.
- Discoverable. "ros2 topic list" shows all seven topics. "ros2 topic info" shows message types and publisher/subscriber counts.
- Timestamped. Each message carries publication time for time-series analysis and synchronization.
- Zero-effort integration. Log: "ros2 bag record /sensors/room_temp". Plot: open rqt_plot. Python: three lines of rclpy. Cloud: rosbridge node. None require knowledge of I2C or USB.

3.3 The Data Flow

Here is exactly what happens when the RA6M5 publishes a temperature reading:

- The 1 Hz timer callback fires inside the FreeRTOS task on the RA6M5.
- sensors_read_sht3x() performs an I2C transaction to the SHT3x at address 0x44 and returns temperature in °C.
- The firmware populates a sensor_msgs/Temperature struct: temperature field, header timestamp (from FreeRTOS tick count), frame_id = "sht3x".
- rcl_publish() serializes the struct into a CDR byte buffer via rosidl type support.
- The buffer is wrapped in an XRCE-DDS frame by the Micro-XRCE-DDS client.
- The frame is passed to cdc_transport_write(), which calls R_USB_Write() to send bytes over USB.
- Bytes travel over USB to /dev/ttyACM0 on the host.
- The micro-ROS agent deserializes the XRCE-DDS frame back into a sensor_msgs/Temperature and publishes through DDS.
- Any subscribed ROS 2 node receives the message. It is indistinguishable from a native Linux publisher.

This entire chain — from I2C sensor read to DDS publication — completes in under 10 milliseconds.

3.4 Micro-ROS vs. Alternatives

Approach	Pros	Cons
Raw serial (Phase A)	Simple, no dependencies, fast to implement	Unstructured, point-to-point, no discovery, no timestamps, custom parsers needed
MQTT	Lightweight, IoT-friendly, cloud-native	Not native to ROS 2, no standardized message types, requires a bridge

rosserial (ROS 1)	Proven, well-documented	Deprecated, ROS 1 only, no DDS integration
Custom bridge	Full control	Must write and maintain serialization, transport, and bridging code
micro-ROS	Native ROS 2, standard types, discovery, multi-subscriber, timestamped	Complex setup, ~200 KB heap, requires host-side agent

3.5 What Micro-ROS Requires from the Firmware

Micro-ROS imposes requirements that bare-metal does not:

- An RTOS — FreeRTOS in this project. The micro-ROS node runs as a task with a 40,000-byte stack.
- A large heap — 200,000 bytes (configTOTAL_HEAP_SIZE). Micro-ROS allocates 80–120 KB during initialization.
- A working malloc — overridden globally to route through pvPortMalloc (freertos_malloc_override.c).
- POSIX stubs — clock_gettime, gettimeofday, usleep, isatty, fprintf (microros_stubs.c).
- A custom transport — USB-CDC open/close/write/read callbacks (microros_transports.c).
- A cross-compiled library — libmicroros.a built for Cortex-M33 with hardware FPU.

Part II: Complete Build Guide

4. Project Overview

4.1 Hardware Bill of Materials

Component	Description	Interface	I2C Address
Renesas EK-RA6M5	Dev board, ARM Cortex-M33, 512 KB SRAM	USB Micro-B (J11 debug, J12 FS)	N/A
SHT3x	Temperature and humidity sensor (Sensirion)	I2C via QWIIC	0x44
MCP9600	Thermocouple interface (hot + cold junction)	I2C via QWIIC	0x67
VCNL4010	Proximity and ambient light sensor (Vishay)	I2C via QWIIC	0x13
USB Micro-B cable	Data cable (NOT power-only)	USB	N/A
QWIIC cables	4-pin JST connectors for I2C daisy-chain	I2C	N/A

4.2 Software Stack

Software	Version	Purpose
e² studio	Latest (Renesas IDE)	Project configuration, build, flash, debug
Renesas FSP	Bundled with e² studio	Hardware abstraction layer and drivers
clang / ATfE	21.1.1	C compiler toolchain
FreeRTOS	Bundled with FSP	RTOS (micro-ROS version only)
micro-ROS (libmicroros.a)	Humble	ROS 2 client library for microcontrollers
ROS 2 Humble	Humble Hawksbill	Host-side ROS 2 framework
micro_ros_agent	Built from source	Serial-to-DDS bridge on host
WSL2 (Ubuntu 22.04)	Jammy Jellyfish	Linux environment on Windows
usbipd-win	Latest	USB passthrough from Windows to WSL2
Podman	3.4.4+	Container runtime for deployment
TeraTerm	Latest	Serial terminal (bare-metal testing)

4.3 I2C Wiring

All three sensors connect via QWIIC daisy-chain on I2C channel 2. Pin assignments in FSP:

- SDA: P414
- SCL: P415
- I2C mode: Standard rate, 7-bit addressing

QWIIC connectors handle pull-up resistors internally. No external pull-ups needed.

5. Phase A: Bare-Metal USB Version

Standalone firmware that reads all three sensors and prints formatted text over USB-CDC to a serial terminal. No RTOS, no ROS. Hardware validation.

5.1 Create the e² Studio Project

1. File → New → Renesas C/C++ Project → Renesas RA.
2. Board: EK-RA6M5.
3. Template: "Bare Metal – Minimal". Name: "Sensor_Project_USB_Version". Finish.

5.2 Configure FSP Peripherals

5.2.1 I2C Master

1. Stacks tab → New Stack → Connectivity → I2C Master (r_iic_master).
2. Properties: Channel 2, Standard rate, Slave Address 0x44, 7-Bit, Callback: i2c_master_callback, SDA: P414, SCL: P415.

5.2.2 USB PCDC

1. Stacks tab → New Stack → Connectivity → USB PCDC (r_usb_pcdc).
2. Properties: USB Mode = Peri, Speed = Full Speed, Module = USB_IP0.
3. Generate Project Content (Ctrl+Shift+G).

5.2.3 Pins

Pins tab: verify P414 = IIC2_SDA, P415 = IIC2_SCL, USB pins auto-configured.

5.3 Application Code: hal_entry.c

The entire application lives in src/hal_entry.c:

```
#include "hal_data.h"
#include <stdio.h>
#include <string.h>

#define SHT3X_ADDR    0x44
#define MCP9600_ADDR  0x67
#define VCNL4010_ADDR 0x13

volatile bool i2c_done = false;

void i2c_master_callback(i2c_master_callback_args_t *p_args) {
```

```
    if (p_args->event == I2C_MASTER_EVENT_TX_COMPLETE ||
        p_args->event == I2C_MASTER_EVENT_RX_COMPLETE) {
        i2c_done = true;
    }
}

void i2c_write_blocking(uint8_t dev_addr, uint8_t *data, uint32_t bytes) {
    R_IIC_MASTER_SlaveAddressSet(&g_i2c_master0_ctrl, dev_addr,
I2C_MASTER_ADDR_MODE_7BIT);
    i2c_done = false;
    R_IIC_MASTER_Write(&g_i2c_master0_ctrl, data, bytes, false);
    while (!i2c_done);
}

void i2c_read_blocking(uint8_t dev_addr, uint8_t *data, uint32_t bytes) {
    R_IIC_MASTER_SlaveAddressSet(&g_i2c_master0_ctrl, dev_addr,
I2C_MASTER_ADDR_MODE_7BIT);
    i2c_done = false;
    R_IIC_MASTER_Read(&g_i2c_master0_ctrl, data, bytes, false);
    while (!i2c_done);
}

void hal_entry(void) {
    R_USB_Open(&g_basic0_ctrl, &g_basic0_cfg);
    R_IIC_MASTER_Open(&g_i2c_master0_ctrl, &g_i2c_master0_cfg);

    char print_buffer[250];
    uint8_t setup_data[2];

    uint8_t sht_cmd[2] = {0x2C, 0x06};
    uint8_t sht_data[6];
    uint8_t mcp_reg;
    uint8_t mcp_data[2];
    uint8_t vcnl_cmd[2];
    uint8_t vcnl_data[2];

    // Wake up the VCNL4010 Lux sensor
    setup_data[0] = 0x82; setup_data[1] = 20;
    i2c_write_blocking(VCNL4010_ADDR, setup_data, 2);
    setup_data[0] = 0x84; setup_data[1] = 0x9D;
    i2c_write_blocking(VCNL4010_ADDR, setup_data, 2);

    uint32_t loop_counter = 0;
    bool usb_connected = false;

    while (1) {
        usb_event_info_t event_info;
        usb_status_t event;

        // --- THE MAGIC FIX ---
```

```
        // This MUST be called constantly in bare-metal to process the USB
state machine
        fsp_err_t err = R_USB_EventGet(&event_info, &event);

        if (err == FSP_SUCCESS) {
            if (event == USB_STATUS_CONFIGURED) {
                usb_connected = true;
            } else if (event == USB_STATUS_DETACH) {
                usb_connected = false;
            }
        }

        if (usb_connected) {
            loop_counter++;

            // Read sensors roughly every 1000 loops (~1 second)
            if (loop_counter >= 1000) {
                loop_counter = 0;

                // 1. SHT3x
                i2c_write_blocking(SHT3X_ADDR, sht_cmd, 2);
                R_BSP_SoftwareDelay(20, BSP_DELAY_UNITS_MILLISECONDS);
                i2c_read_blocking(SHT3X_ADDR, sht_data, 6);
                uint16_t rawTemp = (uint16_t)((sht_data[0] << 8) |
sht_data[1]);
                uint16_t rawHum  = (uint16_t)((sht_data[3] << 8) |
sht_data[4]);
                float roomC = -45.0f + (175.0f * ((float)rawTemp /
65535.0f));
                float roomF = (roomC * 1.8f) + 32.0f;
                float humid = 100.0f * ((float)rawHum / 65535.0f);

                // 2. MCP9600
                mcp_reg = 0x00;
                i2c_write_blocking(MCP9600_ADDR, &mcp_reg, 1);
                i2c_read_blocking(MCP9600_ADDR, mcp_data, 2);
                int16_t rawHot = (int16_t)((mcp_data[0] << 8) |
mcp_data[1]);
                float hotC = (float)rawHot * 0.0625f;
                float hotF = (hotC * 1.8f) + 32.0f;

                mcp_reg = 0x02;
                i2c_write_blocking(MCP9600_ADDR, &mcp_reg, 1);
                i2c_read_blocking(MCP9600_ADDR, mcp_data, 2);
                int16_t rawCold = (int16_t)((mcp_data[0] << 8) |
mcp_data[1]);
                float coldC = (float)rawCold * 0.0625f;
                float coldF = (coldC * 1.8f) + 32.0f;

                // 3. VCNL4010
                vcnl_cmd[0] = 0x80; vcnl_cmd[1] = 0x18;
                i2c_write_blocking(VCNL4010_ADDR, vcnl_cmd, 2);
```

```
                R_BSP_SoftwareDelay(100, BSP_DELAY_UNITS_MILLISECONDS);

                vcnl_cmd[0] = 0x87;
                i2c_write_blocking(VCNL4010_ADDR, vcnl_cmd, 1);
                i2c_read_blocking(VCNL4010_ADDR, vcnl_data, 2);
                uint16_t prox = (uint16_t)((vcnl_data[0] << 8) |
vcnl_data[1]);

                vcnl_cmd[0] = 0x85;
                i2c_write_blocking(VCNL4010_ADDR, vcnl_cmd, 1);
                i2c_read_blocking(VCNL4010_ADDR, vcnl_data, 2);
                uint16_t lux = (uint16_t)((vcnl_data[0] << 8) |
vcnl_data[1]);

                char* doorStatus = (prox >= 10000) ? "Closed" : "Opened";

                // 4. Print
                sprintf(print_buffer,
                    "Room: %.1fC|%.1fF | Hot: %.1fC|%.1fF | Cold:
%.1fC|%.1fF | Hum: %.0f%% | Door: %s | Lux: %u\r\n",
                    roomC, roomF, hotC, hotF, coldC, coldF, humid,
doorStatus, lux);

                R_USB_Write(&g_basic0_ctrl, (uint8_t *)print_buffer,
(uint32_t)strlen(print_buffer), USB_CLASS_PCDC);
            }
        }

        // A tiny 1ms delay allows the loop to spin fast enough to keep
USB alive
        R_BSP_SoftwareDelay(1, BSP_DELAY_UNITS_MILLISECONDS);
    }
}
```

> ⚠ **CRITICAL: USB Polling**
> R_USB_EventGet() in the while(1) loop is essential. In bare-metal mode, the USB state machine only advances when you call this function. Without it, the board never enumerates.

5.4 Build, Flash, and Verify

1. Build (Ctrl+B). Verify zero errors.
2. Connect J11 (debug port) to PC. Debug → Renesas GDB Hardware Debugging. Flash is automatic.
3. Play (F8). Connect second USB cable to J12 (data port).
4. TeraTerm: Serial, select COM port. Output appears once per second.

> ⚠️ **Known Issue: Slow USB Enumeration**
> On some Windows laptops, the board can take up to 15 minutes to appear as a COM device. This is a Windows USB driver issue. Try a different cable (many are power-only) or USB port.

6. Phase B: Micro-ROS Version

6.1 Create the Project

1. File → New → Renesas C/C++ Project → Renesas RA.
2. Board: EK-RA6M5. Toolchain: LLVM for ARM Embedded (clang) 21.1.1.
3. Template: FreeRTOS – Minimal. Name: "Sensor_Project_ROS". Finish.

6.2 FSP Configuration

6.2.1 I2C Master

Same as Section 5.2.1 except callback = sensors_i2c_callback.

6.2.2 USB PCDC

Same as Section 5.2.2 except callback = cdc_usb_callback.

6.2.3 FreeRTOS Thread

1. Stacks tab: click existing thread. Name: sensor_node. Entry: new_thread0_entry.
2. Stack Size: 40000 bytes. Priority: 1. Generate Project Content.

6.2.4 Critical: FreeRTOS Heap Size

After generating, open ra_cfg/aws/FreeRTOSConfig.h. Change:

```
#define configTOTAL_HEAP_SIZE (200000)
```

> ⚠ **WARNING**
> The -D compiler flag CANNOT override this because of the #ifndef guard. Edit the file directly. FSP may reset it to 1024 on every Generate Project Content. Always verify.

6.3 Building libmicroros.a

6.3.1 What Did NOT Work

- library_generation.sh pulls bleeding-edge source with outdated rules → cascading failures
- Windows PATH parentheses cause Makefile syntax errors
- Each attempt: 40+ minutes before failing
- Even successful builds targeted wrong ARM (ARMv7-M instead of ARMv8-M): "conflicting CPU architectures 17/2"

6.3.2 What Worked

- Clean PATH stripping Windows directories:

```
env -i HOME="$HOME" USER="$USER"
PATH="/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin" bash -
c '...'
```

- Correct flags: -mcpu=cortex-m33 -mthumb -mfpu=fpv5-sp-d16 -mfloat-abi=hard
- Extra package exclusions for C++-only packages

Successful build: ~1.5 hours.

6.3.3 Installing the Library

1. Copy libmicroros.a to: micro_ros_renesas2estudio_component/libmicroros/libmicroros.a
2. Copy include/ to: micro_ros_renesas2estudio_component/libmicroros/include/
3. Project Properties → Compiler → Includes: add include/ path.
4. Linker: add libmicroros.a path and "-lmicroros". Build to verify.

6.4 Source Files

File	Purpose
new_thread0_entry.c	Main FreeRTOS task: micro-ROS init, 7 publishers, 1 Hz timer callback
sensors.c / sensors.h	Sensor abstraction: init, read with timeouts
microros_transports.c / .h	USB-CDC transport: stream buffers, semaphores
microros_stubs.c	POSIX shims: _exit, clock_gettime, usleep, fprintf, isatty
freertos_malloc_override.c	Global malloc/free/calloc/realloc → pvPortMalloc/vPortFree
usb_descriptor.c	CDC-ACM descriptors. VID 0x045B, PID 0x2013

7. Complete Source Code Listings (Micro-ROS Version)

Every source file in src/ reproduced verbatim. Copy-paste into your e² studio project.

7.1 new_thread0_entry.c

Main FreeRTOS task. Initializes sensors, registers USB-CDC transport, waits for agent, creates 7 publishers, spins 1 Hz timer.

```
#include "sensors.h"
#include "microros_transports.h"
#include <rcl/rcl.h>
#include <rcl/error_handling.h>
#include <rclc/rclc.h>
#include <rclc/executor.h>
#include <rmw_microros/rmw_microros.h>

#include <sensor_msgs/msg/temperature.h>
#include <sensor_msgs/msg/relative_humidity.h>
#include <std_msgs/msg/u_int16.h>
#include <std_msgs/msg/string.h>

#include <stdio.h>
#include <string.h>

#include "../ra_gen/new_thread0.h"
#include "../ra/aws/FreeRTOS/FreeRTOS/Source/include/FreeRTOS.h"
#include "../ra/aws/FreeRTOS/FreeRTOS/Source/include/task.h"

/* ---------------------------------------------------------------------
 * Debug variables — check these in the debugger if something fails.
 *   g_breadcrumb:  which init step we reached
 *   g_last_rc:     return code from the last RCCHECK call
 * --------------------------------------------------------------------- */
volatile int g_breadcrumb = 0;
volatile int g_last_rc = 0;
volatile int g_rc_check_num = 0;
char g_last_error_buf[256] = {0};

/* ---------------------------------------------------------------------
 * micro-ROS state
 * --------------------------------------------------------------------- */

static rcl_allocator_t      g_allocator;
static rclc_support_t       g_support;
static rcl_node_t           g_node;
static rclc_executor_t      g_executor;
static rcl_timer_t          g_timer;
```

```
static rcl_publisher_t pub_room_temp;
static rcl_publisher_t pub_humidity;
static rcl_publisher_t pub_hot;
static rcl_publisher_t pub_cold;
static rcl_publisher_t pub_prox;
static rcl_publisher_t pub_ambient;
static rcl_publisher_t pub_diag;

static sensor_msgs__msg__Temperature        msg_room_temp;
static sensor_msgs__msg__RelativeHumidity   msg_humidity;
static sensor_msgs__msg__Temperature        msg_hot;
static sensor_msgs__msg__Temperature        msg_cold;
static std_msgs__msg__UInt16                msg_prox;
static std_msgs__msg__UInt16                msg_ambient;
static std_msgs__msg__String                msg_diag;

static char diag_text[128];

/* ---------------------------------------------------------------------
 * Helpers
 * --------------------------------------------------------------------- */

#define RCCHECK(fn) do {
\
    g_rc_check_num++;
\
    rcl_ret_t _rc = (fn);
\
    g_last_rc = (int)_rc;
\
    if (_rc != RCL_RET_OK) {
\
        rcutils_error_string_t _err = rcl_get_error_string();
\
        strncpy(g_last_error_buf, _err.str, sizeof(g_last_error_buf) - 1);
\
        g_last_error_buf[sizeof(g_last_error_buf) - 1] = '\0';
\
        g_breadcrumb = 999;
\
        while (1) { vTaskDelay(pdMS_TO_TICKS(1000)); }
\
    }
\
} while (0)

static void diag_log(const char *text)
{
    size_t len = strlen(text);
    if (len >= sizeof(diag_text)) len = sizeof(diag_text) - 1;
    memcpy(diag_text, text, len);
```

```
    diag_text[len] = '\0';

    msg_diag.data.data = (char *)diag_text;
    msg_diag.data.size = len;
    msg_diag.data.capacity = sizeof(diag_text);

    (void)rcl_publish(&pub_diag, &msg_diag, NULL);
}

static void stamp_header(builtin_interfaces__msg__Time *stamp)
{
    int64_t now_ns = rmw_uros_epoch_nanos();
    stamp->sec     = (int32_t)(now_ns / 1000000000LL);
    stamp->nanosec = (uint32_t)(now_ns % 1000000000LL);
}

/* ---------------------------------------------------------------------
 * Timer callback — runs every 1 second, reads all sensors, publishes
 * --------------------------------------------------------------------- */

static void timer_callback(rcl_timer_t *timer, int64_t last_call_time)
{
    (void)timer;
    (void)last_call_time;

    float  tempC, humid, hotC, coldC;
    uint16_t prox, ambient;

    if (sensors_read_sht3x(&tempC, &humid)) {
        stamp_header(&msg_room_temp.header.stamp);
        msg_room_temp.temperature = tempC;
        msg_room_temp.variance    = 0.0;
        (void)rcl_publish(&pub_room_temp, &msg_room_temp, NULL);

        stamp_header(&msg_humidity.header.stamp);
        msg_humidity.relative_humidity = humid;
        msg_humidity.variance          = 0.0;
        (void)rcl_publish(&pub_humidity, &msg_humidity, NULL);
    } else {
        diag_log("SHT3x read failed");
    }

    if (sensors_read_mcp9600(&hotC, &coldC)) {
        stamp_header(&msg_hot.header.stamp);
        msg_hot.temperature = hotC;
        msg_hot.variance    = 0.0;
        (void)rcl_publish(&pub_hot, &msg_hot, NULL);

        stamp_header(&msg_cold.header.stamp);
        msg_cold.temperature = coldC;
        msg_cold.variance    = 0.0;
```

```
        (void)rcl_publish(&pub_cold, &msg_cold, NULL);
    } else {
        diag_log("MCP9600 read failed");
    }

    if (sensors_read_vcnl4010(&prox, &ambient)) {
        msg_prox.data = prox;
        (void)rcl_publish(&pub_prox, &msg_prox, NULL);

        msg_ambient.data = ambient;
        (void)rcl_publish(&pub_ambient, &msg_ambient, NULL);
    } else {
        diag_log("VCNL4010 read failed");
    }
}

/* ---------------------------------------------------------------------
 * Thread entry
 * --------------------------------------------------------------------- */

void new_thread0_entry(void *pvParameters)
{
    FSP_PARAMETER_NOT_USED(pvParameters);

    /* --- 1. Init sensors (I2C) --- */
    g_breadcrumb = 1;

    if (!sensors_init()) {
        g_breadcrumb = 100;
        while (1) { vTaskDelay(pdMS_TO_TICKS(1000)); }
    }

    /* --- 2. Register USB-CDC transport --- */
    g_breadcrumb = 2;

    rmw_uros_set_custom_transport(
        true,
        NULL,
        cdc_transport_open,
        cdc_transport_close,
        cdc_transport_write,
        cdc_transport_read
    );

    /* --- 3. Set up message frame IDs --- */
    g_breadcrumb = 3;

    static const char frame_sht[] = "sht3x";
    static const char frame_mcp[] = "mcp9600";

    msg_room_temp.header.frame_id.data     = (char *)frame_sht;
```

```
    msg_room_temp.header.frame_id.size     = sizeof(frame_sht) - 1;
    msg_room_temp.header.frame_id.capacity = sizeof(frame_sht);

    msg_humidity.header.frame_id.data      = (char *)frame_sht;
    msg_humidity.header.frame_id.size      = sizeof(frame_sht) - 1;
    msg_humidity.header.frame_id.capacity  = sizeof(frame_sht);

    msg_hot.header.frame_id.data           = (char *)frame_mcp;
    msg_hot.header.frame_id.size           = sizeof(frame_mcp) - 1;
    msg_hot.header.frame_id.capacity       = sizeof(frame_mcp);

    msg_cold.header.frame_id.data          = (char *)frame_mcp;
    msg_cold.header.frame_id.size          = sizeof(frame_mcp) - 1;
    msg_cold.header.frame_id.capacity      = sizeof(frame_mcp);

    /* --- 4. Wait for micro-ROS agent --- */
    g_breadcrumb = 4;
    g_allocator = rcl_get_default_allocator();

    while (rmw_uros_ping_agent(1000, 1) != RMW_RET_OK) {
        vTaskDelay(pdMS_TO_TICKS(500));
    }

    /* --- 5. Initialize micro-ROS support (logging, context, rmw) --- */
    g_breadcrumb = 5;

    RCCHECK(rclc_support_init(&g_support, 0, NULL, &g_allocator));

    /* --- 6. Create node --- */
    g_breadcrumb = 6;

    RCCHECK(rclc_node_init_default(&g_node, "ra6m5_sensor_node", "",
&g_support));

    /* --- 7. Create publishers --- */
    g_breadcrumb = 7;

    RCCHECK(rclc_publisher_init_default(
        &pub_room_temp, &g_node,
        ROSIDL_GET_MSG_TYPE_SUPPORT(sensor_msgs, msg, Temperature),
        "/sensors/room_temp"));

    RCCHECK(rclc_publisher_init_default(
        &pub_humidity, &g_node,
        ROSIDL_GET_MSG_TYPE_SUPPORT(sensor_msgs, msg, RelativeHumidity),
        "/sensors/humidity"));

    RCCHECK(rclc_publisher_init_default(
        &pub_hot, &g_node,
        ROSIDL_GET_MSG_TYPE_SUPPORT(sensor_msgs, msg, Temperature),
        "/sensors/thermocouple_hot"));
```

```
    RCCHECK(rclc_publisher_init_default(
        &pub_cold, &g_node,
        ROSIDL_GET_MSG_TYPE_SUPPORT(sensor_msgs, msg, Temperature),
        "/sensors/thermocouple_cold"));

    RCCHECK(rclc_publisher_init_default(
        &pub_prox, &g_node,
        ROSIDL_GET_MSG_TYPE_SUPPORT(std_msgs, msg, UInt16),
        "/sensors/proximity"));

    RCCHECK(rclc_publisher_init_default(
        &pub_ambient, &g_node,
        ROSIDL_GET_MSG_TYPE_SUPPORT(std_msgs, msg, UInt16),
        "/sensors/ambient_light"));

    RCCHECK(rclc_publisher_init_default(
        &pub_diag, &g_node,
        ROSIDL_GET_MSG_TYPE_SUPPORT(std_msgs, msg, String),
        "/sensors/diagnostic"));

    /* --- 8. Create timer + executor --- */
    g_breadcrumb = 8;

    RCCHECK(rclc_timer_init_default2(
        &g_timer, &g_support,
        RCL_MS_TO_NS(1000),
        timer_callback,
        true));

    RCCHECK(rclc_executor_init(&g_executor, &g_support.context, 1,
&g_allocator));
    RCCHECK(rclc_executor_add_timer(&g_executor, &g_timer));

    /* --- 9. Running --- */
    g_breadcrumb = 9;

    diag_log("ra6m5_sensor_node initialized");

    while (1) {
        rclc_executor_spin_some(&g_executor, RCL_MS_TO_NS(100));
        vTaskDelay(pdMS_TO_TICKS(10));
    }
}
```

7.2 sensors.h

```
#ifndef SENSORS_H
#define SENSORS_H

#include <stdint.h>
#include <stdbool.h>

/* Call once before any read. Opens I2C, wakes VCNL4010. */
bool sensors_init(void);

/* Read SHT3x temperature (°C) and humidity (0.0–1.0 fraction). */
bool sensors_read_sht3x(float *temp_c, float *humidity);

/* Read MCP9600 hot-junction and cold-junction temperatures (°C). */
bool sensors_read_mcp9600(float *hot_c, float *cold_c);

/* Read VCNL4010 raw proximity and ambient light counts. */
bool sensors_read_vcnl4010(uint16_t *prox, uint16_t *ambient);

#endif
```

7.3 sensors.c

Sensor abstraction. Uses vTaskDelay and 100ms I2C timeouts to avoid hanging on bus errors.

```
#include "sensors.h"

#include "../ra_gen/hal_data.h"
#include "../ra/aws/FreeRTOS/FreeRTOS/Source/include/FreeRTOS.h"
#include "../ra/aws/FreeRTOS/FreeRTOS/Source/include/task.h"

#define SHT3X_ADDR    0x44
#define MCP9600_ADDR  0x67
#define VCNL4010_ADDR 0x13

static volatile bool g_i2c_done = false;
static volatile bool g_i2c_error = false;

/* Called by FSP I2C driver on transfer complete. Wire this up in FSP
 * configurator as the callback for g_i2c_master0 (see FSP setup step). */
void sensors_i2c_callback(i2c_master_callback_args_t *p_args)
{
    if (p_args->event == I2C_MASTER_EVENT_TX_COMPLETE ||
        p_args->event == I2C_MASTER_EVENT_RX_COMPLETE) {
        g_i2c_done = true;
```

```
    } else {
        g_i2c_error = true;
        g_i2c_done = true;
    }
}

static bool i2c_write(uint8_t dev_addr, const uint8_t *data, uint32_t len)
{
    if (R_IIC_MASTER_SlaveAddressSet(&g_i2c_master0_ctrl, dev_addr,
                                     I2C_MASTER_ADDR_MODE_7BIT) !=
FSP_SUCCESS)
        return false;
    g_i2c_done = false;
    g_i2c_error = false;
    if (R_IIC_MASTER_Write(&g_i2c_master0_ctrl, (uint8_t *)data, len,
false)
        != FSP_SUCCESS)
        return false;
    /* Wait with a timeout so we don't hang forever on bus errors. */
    uint32_t timeout = 100;  /* 100 ms */
    while (!g_i2c_done && timeout--) {
        vTaskDelay(pdMS_TO_TICKS(1));
    }
    return g_i2c_done && !g_i2c_error;
}

static bool i2c_read(uint8_t dev_addr, uint8_t *data, uint32_t len)
{
    if (R_IIC_MASTER_SlaveAddressSet(&g_i2c_master0_ctrl, dev_addr,
                                     I2C_MASTER_ADDR_MODE_7BIT) !=
FSP_SUCCESS)
        return false;
    g_i2c_done = false;
    g_i2c_error = false;
    if (R_IIC_MASTER_Read(&g_i2c_master0_ctrl, data, len, false) !=
FSP_SUCCESS)
        return false;
    uint32_t timeout = 100;
    while (!g_i2c_done && timeout--) {
        vTaskDelay(pdMS_TO_TICKS(1));
    }
    return g_i2c_done && !g_i2c_error;
}

bool sensors_init(void)
{
    if (R_IIC_MASTER_Open(&g_i2c_master0_ctrl, &g_i2c_master0_cfg) !=
FSP_SUCCESS)
        return false;

    /* Wake up the VCNL4010 ambient light / proximity sensor. */
    uint8_t setup[2];
```

```
    setup[0] = 0x82; setup[1] = 20;       /* IR LED current */
    if (!i2c_write(VCNL4010_ADDR, setup, 2)) return false;
    setup[0] = 0x84; setup[1] = 0x9D;     /* ambient light config */
    if (!i2c_write(VCNL4010_ADDR, setup, 2)) return false;

    return true;
}

bool sensors_read_sht3x(float *temp_c, float *humidity)
{
    uint8_t cmd[2] = {0x2C, 0x06};
    uint8_t raw[6];
    if (!i2c_write(SHT3X_ADDR, cmd, 2)) return false;
    vTaskDelay(pdMS_TO_TICKS(20));
    if (!i2c_read(SHT3X_ADDR, raw, 6)) return false;

    uint16_t rawTemp = (uint16_t)((raw[0] << 8) | raw[1]);
    uint16_t rawHum  = (uint16_t)((raw[3] << 8) | raw[4]);
    *temp_c   = -45.0f + (175.0f * ((float)rawTemp / 65535.0f));
    *humidity = (float)rawHum / 65535.0f;  /* 0.0–1.0 fraction */
    return true;
}

bool sensors_read_mcp9600(float *hot_c, float *cold_c)
{
    uint8_t reg;
    uint8_t raw[2];

    reg = 0x00;
    if (!i2c_write(MCP9600_ADDR, &reg, 1)) return false;
    if (!i2c_read(MCP9600_ADDR, raw, 2)) return false;
    int16_t rawHot = (int16_t)((raw[0] << 8) | raw[1]);
    *hot_c = (float)rawHot * 0.0625f;

    reg = 0x02;
    if (!i2c_write(MCP9600_ADDR, &reg, 1)) return false;
    if (!i2c_read(MCP9600_ADDR, raw, 2)) return false;
    int16_t rawCold = (int16_t)((raw[0] << 8) | raw[1]);
    *cold_c = (float)rawCold * 0.0625f;

    return true;
}

bool sensors_read_vcnl4010(uint16_t *prox, uint16_t *ambient)
{
    uint8_t cmd[2];
    uint8_t raw[2];

    /* Trigger a proximity + ambient on-demand measurement. */
    cmd[0] = 0x80; cmd[1] = 0x18;
    if (!i2c_write(VCNL4010_ADDR, cmd, 2)) return false;
```

```
    vTaskDelay(pdMS_TO_TICKS(100));

    cmd[0] = 0x87;
    if (!i2c_write(VCNL4010_ADDR, cmd, 1)) return false;
    if (!i2c_read(VCNL4010_ADDR, raw, 2)) return false;
    *prox = (uint16_t)((raw[0] << 8) | raw[1]);

    cmd[0] = 0x85;
    if (!i2c_write(VCNL4010_ADDR, cmd, 1)) return false;
    if (!i2c_read(VCNL4010_ADDR, raw, 2)) return false;
    *ambient = (uint16_t)((raw[0] << 8) | raw[1]);

    return true;
}
```

7.4 microros_transports.h

```
#ifndef MICROROS_TRANSPORTS_H
#define MICROROS_TRANSPORTS_H

#include <uxr/client/transport.h>
#include <stdbool.h>
#include <stdint.h>
#include <stddef.h>

#include "../ra_gen/hal_data.h"
#include "../ra/fsp/inc/api/r_usb_basic_api.h"
#include "../ra/fsp/inc/instances/r_usb_basic.h"

bool cdc_transport_open(struct uxrCustomTransport *transport);
bool cdc_transport_close(struct uxrCustomTransport *transport);
size_t cdc_transport_write(struct uxrCustomTransport *transport,
                           const uint8_t *buf, size_t len, uint8_t *err);
size_t cdc_transport_read(struct uxrCustomTransport *transport,
                          uint8_t *buf, size_t len, int timeout, uint8_t
*err);

/* Called from FSP USB PCDC callback when data arrives or a write
completes.
 * Wire this up as the callback for g_basic0 in FSP (USB stack's
p_callback). */
void cdc_usb_callback(usb_event_info_t *p_event_info, usb_hdl_t handle,
                      usb_onoff_t on_off);

#endif
```

7.5 microros_transports.c

USB-CDC transport. FreeRTOS stream buffer for RX, binary semaphore for TX. Note USB_STATUS_SUSPEND is a no-op — Windows suspends during idle.

```
#include "microros_transports.h"
#include <string.h>
#include "../ra/aws/FreeRTOS/FreeRTOS/Source/include/FreeRTOS.h"
#include "../ra/aws/FreeRTOS/FreeRTOS/Source/include/semphr.h"
#include "../ra/aws/FreeRTOS/FreeRTOS/Source/include/stream_buffer.h"

extern usb_instance_ctrl_t g_basic0_ctrl;
extern const usb_cfg_t     g_basic0_cfg;

#define RX_BUFFER_SIZE      2048
#define RX_TRIGGER_LEVEL    1
```

```
#define WRITE_TIMEOUT_MS    1000

static StreamBufferHandle_t g_rx_stream = NULL;
static SemaphoreHandle_t    g_tx_done   = NULL;
static volatile bool        g_usb_ready = false;
static bool                 g_usb_opened = false;
static uint8_t              g_rx_scratch[64];

void cdc_usb_callback(usb_event_info_t *p_event_info, usb_hdl_t handle,
                      usb_onoff_t on_off)
{
    (void)handle;
    (void)on_off;

    BaseType_t xHigherPriorityTaskWoken = pdFALSE;

    switch (p_event_info->event) {
    case USB_STATUS_CONFIGURED:
        g_usb_ready = true;
        R_USB_Read(&g_basic0_ctrl, g_rx_scratch, sizeof(g_rx_scratch),
                   USB_CLASS_PCDC);
        break;

    case USB_STATUS_READ_COMPLETE:
        if (g_rx_stream != NULL && p_event_info->data_size > 0) {
            xStreamBufferSendFromISR(g_rx_stream, g_rx_scratch,
                                     p_event_info->data_size,
                                     &xHigherPriorityTaskWoken);
        }
        R_USB_Read(&g_basic0_ctrl, g_rx_scratch, sizeof(g_rx_scratch),
                   USB_CLASS_PCDC);
        break;

    case USB_STATUS_WRITE_COMPLETE:
        if (g_tx_done != NULL) {
            xSemaphoreGiveFromISR(g_tx_done, &xHigherPriorityTaskWoken);
        }
        break;

    case USB_STATUS_DETACH:
            g_usb_ready = false;
            break;

    case USB_STATUS_SUSPEND:
            /* Don't kill the connection on suspend — Windows does this
during idle */
            break;

    default:
        break;
    }
```

```
    portYIELD_FROM_ISR(xHigherPriorityTaskWoken);
}

bool cdc_transport_open(struct uxrCustomTransport *transport)
{
    (void)transport;

    if (g_rx_stream == NULL) {
        g_rx_stream = xStreamBufferCreate(RX_BUFFER_SIZE,
RX_TRIGGER_LEVEL);
        if (g_rx_stream == NULL) return false;
    }
    if (g_tx_done == NULL) {
        g_tx_done = xSemaphoreCreateBinary();
        if (g_tx_done == NULL) return false;
    }

    /* Only open USB once — keep it alive across retries */
    if (!g_usb_opened) {
        if (R_USB_Open(&g_basic0_ctrl, &g_basic0_cfg) != FSP_SUCCESS) {
            return false;
        }
        g_usb_opened = true;
    }

    /* Wait up to 5 seconds for host to enumerate us. */
    TickType_t start = xTaskGetTickCount();
    while (!g_usb_ready) {
        if ((xTaskGetTickCount() - start) > pdMS_TO_TICKS(5000)) {
            return false;
        }
        vTaskDelay(pdMS_TO_TICKS(10));
    }
    return true;
}

bool cdc_transport_close(struct uxrCustomTransport *transport)
{
    (void)transport;
    /* Don't close USB — keep it alive so the COM port stays visible */
    return true;
}

size_t cdc_transport_write(struct uxrCustomTransport *transport,
                           const uint8_t *buf, size_t len, uint8_t *err)
{
    (void)transport;

    if (!g_usb_ready) {
        *err = 1;
```

```
        return 0;
    }

    xSemaphoreTake(g_tx_done, 0);

    if (R_USB_Write(&g_basic0_ctrl, (uint8_t *)buf, (uint32_t)len,
                    USB_CLASS_PCDC) != FSP_SUCCESS) {
        *err = 1;
        return 0;
    }

    if (xSemaphoreTake(g_tx_done, pdMS_TO_TICKS(WRITE_TIMEOUT_MS)) !=
pdTRUE) {
        *err = 1;
        return 0;
    }

    return len;
}

size_t cdc_transport_read(struct uxrCustomTransport *transport,
                          uint8_t *buf, size_t len, int timeout, uint8_t
*err)
{
    (void)transport;

    if (!g_usb_ready) {
        *err = 1;
        return 0;
    }

    TickType_t ticks = (timeout < 0) ? portMAX_DELAY :
pdMS_TO_TICKS(timeout);
    size_t got = xStreamBufferReceive(g_rx_stream, buf, len, ticks);
    if (got == 0) *err = 1;
    return got;
}
```

7.6 microros_stubs.c

POSIX shims for clang libc. clock_gettime/gettimeofday backed by xTaskGetTickCount().

```
#include <time.h>
#include <sys/time.h>
#include <stdint.h>
#include <stdio.h>
#include <unistd.h>

#include "../ra/aws/FreeRTOS/FreeRTOS/Source/include/FreeRTOS.h"
#include "../ra/aws/FreeRTOS/FreeRTOS/Source/include/task.h"
/* tick rate — FreeRTOS default. If you change configTICK_RATE_HZ in FSP,
 * change this to match, or include FreeRTOSConfig.h instead. */
#ifndef configTICK_RATE_HZ
#define configTICK_RATE_HZ 1000
#endif
/* --- Stub for clang libc abort() --- */
void _exit(int status)
{
    (void)status;
    while (1) { }
}
/* --- clock_gettime used by micro-ROS for timestamps --- */
int clock_gettime(clockid_t clk_id, struct timespec *tp)
{
    (void)clk_id;
    uint32_t ticks = xTaskGetTickCount();
    tp->tv_sec  = ticks / configTICK_RATE_HZ;
    tp->tv_nsec = (ticks % configTICK_RATE_HZ) * (1000000000L /
configTICK_RATE_HZ);
    return 0;
}
/* --- gettimeofday used by clang libc time() --- */
int gettimeofday(struct timeval *tv, void *tz)
{
    (void)tz;
    if (tv) {
        uint32_t ticks = xTaskGetTickCount();
        tv->tv_sec  = ticks / configTICK_RATE_HZ;
        tv->tv_usec = (ticks % configTICK_RATE_HZ) * (1000000L /
configTICK_RATE_HZ);
    }
    return 0;
}
/* --- usleep used by rclc_sleep_ms --- */
int usleep(useconds_t usec)
{
    /* Round up to at least 1 tick so short sleeps still yield. */
    uint32_t ms = (usec + 999) / 1000;
    if (ms == 0) ms = 1;
```

```
    vTaskDelay(pdMS_TO_TICKS(ms));
    return 0;
}
/* --- stderr stub ---
 * rosidl_runtime_c references fprintf(stderr, ...) for error reporting,
 * but clang's libc declares stderr as extern without defining it. Provide
 * a dummy FILE and a no-op fprintf that drops writes. */
static FILE stderr_stub;
//FILE *const stderr = &stderr_stub;
int fprintf(FILE *stream, const char *format, ...)
{
    (void)stream;
    (void)format;
    return 0;
}

/* --- stdout stub ---
 * rcutils_logging_initialize references stdout for console output.
 * Provide a dummy FILE just like stderr above. */
static FILE stdout_stub;
//FILE *const stdout = &stdout_stub;

/* --- isatty stub ---
 * rcutils logging calls isatty() to decide whether to use color codes.
 * Always return 0 (not a terminal). */
int isatty(int fd)
{
    (void)fd;
    return 0;
}
```

7.7 freertos_malloc_override.c

Critical file. Overrides global malloc/free/calloc/realloc so ALL allocations route through FreeRTOS.

```
/**
 * freertos_malloc_override.c
 *
 * Drop this file into your src/ folder. It replaces the standard libc
 * malloc/free/calloc/realloc with versions that route through FreeRTOS
 * pvPortMalloc / vPortFree.
 *
 * WHY: On RA6M5 with clang/ATfE, the default malloc() calls _sbrk(),
 *      which returns -1 because the BSP system heap isn't configured.
 *      You already fixed this for the rcl allocator, but rcutils,
 *      rmw_microxrcedds, and other micro-ROS internals still call
 *      malloc() directly. This file catches ALL of those.
```

```
 *
 * REQUIRES: FreeRTOS heap (heap_4.c or heap_5.c) with sufficient
 *           configTOTAL_HEAP_SIZE (you have 200000 — should be fine).
 */

#include <stddef.h>
#include <string.h>

#include "../ra/aws/FreeRTOS/FreeRTOS/Source/include/FreeRTOS.h"

/* ------------------------------------------------------------------ */
/*  Global overrides — the linker will prefer these over libc's weak   */
/*  definitions of malloc/free/calloc/realloc.                         */
/* ------------------------------------------------------------------ */

void *malloc(size_t size)
{
    if (size == 0) {
        return NULL;
    }
    return pvPortMalloc(size);
}

void free(void *ptr)
{
    vPortFree(ptr);  /* vPortFree already handles NULL */
}

void *calloc(size_t nmemb, size_t size)
{
    size_t total = nmemb * size;

    /* Overflow check */
    if (nmemb != 0 && total / nmemb != size) {
        return NULL;
    }

    void *ptr = pvPortMalloc(total);
    if (ptr != NULL) {
        memset(ptr, 0, total);
    }
    return ptr;
}

void *realloc(void *ptr, size_t new_size)
{
    /*
     * FreeRTOS doesn't have a native realloc. This is a simple
     * allocate-copy-free approach. It works but always copies
     * the full new_size (safe because we can't query the old size
     * from heap_4 without hacking the header). Since micro-ROS
```

```
     * rarely reallocs, this is fine.
     *
     * Edge cases handled:
     *   realloc(NULL, size) = malloc(size)
     *   realloc(ptr, 0)     = free(ptr), return NULL
     */
    if (ptr == NULL) {
        return malloc(new_size);
    }

    if (new_size == 0) {
        vPortFree(ptr);
        return NULL;
    }

    void *new_ptr = pvPortMalloc(new_size);
    if (new_ptr != NULL) {
        /*
         * We don't know the old allocation size, so we copy new_size
         * bytes. If the old block was smaller, we're reading past it —
         * but into FreeRTOS heap memory we own, so it's safe (just
         * garbage in the tail). The caller only uses the valid portion.
         *
         * This is the standard workaround for FreeRTOS realloc.
         */
        memcpy(new_ptr, ptr, new_size);
        vPortFree(ptr);
    }
    return new_ptr;
}
```

7.8 usb_descriptor.c

CDC-ACM descriptors. Device appears as "RA6M5 Sensor Node".

```
/* Standard USB CDC-ACM descriptor for Renesas RA FSP.
 * This is boilerplate from the Renesas USB PCDC example project. */

#include "../ra/fsp/inc/api/r_usb_basic_api.h"
#include "../ra/fsp/inc/instances/r_usb_basic.h"

#define USB_BCDNUM              (0x0200u)
#define USB_RELEASE             (0x0200u)
#define USB_VENDORID            (0x045Bu)   /* Renesas */
#define USB_PRODUCTID           (0x2013u)   /* PCDC example */
#define USB_DCPMAXP             (64u)
#define USB_CONFIGURATION       (0u)
#define USB_DEVICECLASS         (0x02u)     /* Communications Device Class
*/

/* --- Device Descriptor --- */
uint8_t g_apl_device[] = {
    18u,                                    /* bLength */
    USB_DT_DEVICE,                          /* bDescriptorType */
    (uint8_t)USB_BCDNUM, (uint8_t)(USB_BCDNUM >> 8), /* bcdUSB */
    USB_DEVICECLASS,                        /* bDeviceClass */
    0x00u,                                  /* bDeviceSubClass */
    0x00u,                                  /* bDeviceProtocol */
    (uint8_t)USB_DCPMAXP,                   /* bMaxPacketSize0 */
    (uint8_t)USB_VENDORID, (uint8_t)(USB_VENDORID >> 8),   /* idVendor */
    (uint8_t)USB_PRODUCTID, (uint8_t)(USB_PRODUCTID >> 8), /* idProduct */
    (uint8_t)USB_RELEASE, (uint8_t)(USB_RELEASE >> 8),     /* bcdDevice */
    1,                                      /* iManufacturer */
    2,                                      /* iProduct */
    3,                                      /* iSerialNumber */
    1                                       /* bNumConfigurations */
};

/* --- Configuration Descriptor (CDC-ACM, 2 interfaces) --- */
uint8_t g_apl_configuration[] = {
    /* Configuration Descriptor */
    9,                                  /* bLength */
    USB_DT_CONFIGURATION,               /* bDescriptorType */
    67, 0,                              /* wTotalLength */
    2,                                  /* bNumInterfaces */
    1,                                  /* bConfigurationValue */
    0,                                  /* iConfiguration */
    0xC0,                               /* bmAttributes: self-powered */
    50,                                 /* bMaxPower: 100 mA */

    /* Interface 0: CDC Control */
    9,                                  /* bLength */
```

```
    USB_DT_INTERFACE,                   /* bDescriptorType */
    0,                                  /* bInterfaceNumber */
    0,                                  /* bAlternateSetting */
    1,                                  /* bNumEndpoints */
    0x02,                               /* bInterfaceClass: CDC */
    0x02,                               /* bInterfaceSubClass: Abstract
Control */
    0x01,                               /* bInterfaceProtocol: AT commands
*/
    0,                                  /* iInterface */

    /* CDC Header Functional */
    5, 0x24, 0x00, 0x10, 0x01,

    /* CDC Call Management Functional */
    5, 0x24, 0x01, 0x00, 0x01,

    /* CDC ACM Functional */
    4, 0x24, 0x02, 0x02,

    /* CDC Union Functional */
    5, 0x24, 0x06, 0x00, 0x01,

    /* Endpoint: Interrupt IN (notifications) */
    7, USB_DT_ENDPOINT, 0x83, 0x03, 0x10, 0x00, 0xFF,

    /* Interface 1: CDC Data */
    9,                                  /* bLength */
    USB_DT_INTERFACE,                   /* bDescriptorType */
    1,                                  /* bInterfaceNumber */
    0,                                  /* bAlternateSetting */
    2,                                  /* bNumEndpoints */
    0x0A,                               /* bInterfaceClass: CDC Data */
    0x00,                               /* bInterfaceSubClass */
    0x00,                               /* bInterfaceProtocol */
    0,                                  /* iInterface */

    /* Endpoint: Bulk OUT */
    7, USB_DT_ENDPOINT, 0x01, 0x02, 0x40, 0x00, 0x00,

    /* Endpoint: Bulk IN */
    7, USB_DT_ENDPOINT, 0x82, 0x02, 0x40, 0x00, 0x00
};

/* --- Qualifier & Other-speed Configuration (full-speed only, dummies) --
- */
uint8_t g_apl_hs_configuration[] = { 0 };
uint8_t g_apl_qualifier_descriptor[] = { 0 };

/* --- String Descriptors --- */
/* String 0: LANGID (English US) */
```

```
static uint8_t lang_id[] = { 4, USB_DT_STRING, 0x09, 0x04 };

/* String 1: Manufacturer = "Renesas" */
static uint8_t manufacturer[] = {
    16, USB_DT_STRING,
    'R', 0, 'e', 0, 'n', 0, 'e', 0, 's', 0, 'a', 0, 's', 0
};

/* String 2: Product = "RA6M5 Sensor Node" */
static uint8_t product[] = {
    36, USB_DT_STRING,
    'R', 0, 'A', 0, '6', 0, 'M', 0, '5', 0, ' ', 0,
    'S', 0, 'e', 0, 'n', 0, 's', 0, 'o', 0, 'r', 0, ' ', 0,
    'N', 0, 'o', 0, 'd', 0, 'e', 0
};

/* String 3: Serial Number */
static uint8_t serial[] = {
    14, USB_DT_STRING,
    '0', 0, '0', 0, '0', 0, '0', 0, '0', 0, '1', 0
};

uint8_t *g_apl_string_table[] = {
    lang_id,
    manufacturer,
    product,
    serial
};

/* --- The descriptor struct micro-ROS + FSP reference --- */
usb_descriptor_t g_usb_descriptor = {
    .p_device                   = g_apl_device,
    .p_config_f                 = g_apl_configuration,
    .p_config_h                 = g_apl_hs_configuration,
    .p_qualifier                = g_apl_qualifier_descriptor,
    .p_string                   = g_apl_string_table,
    .num_string                 = 4
};
```

8. Debugging: What Went Wrong and How It Was Fixed

8.1 Bug #1: Broken malloc (rc=10)

Symptom: rclc_support_init() returned RCL_RET_BAD_ALLOC.

Root Cause: clang libc's _sbrk() not wired by FSP. Every malloc() returned NULL.

Fix: freertos_malloc_override.c (Section 7.7) overrides global malloc/free/calloc/realloc.

8.2 Bug #2: FreeRTOS Heap = 1024 Bytes

Symptom: Large allocations failed even after fixing malloc.

Root Cause: -DconfigTOTAL_HEAP_SIZE silently ignored by #ifndef guard in generated header.

Fix: Edit ra_cfg/aws/FreeRTOSConfig.h directly. Set to 200000. Verify .bss grew to ~268 KB.

8.3 Bug #3: rclc_support_init Returns rc=1

Symptom: Error code 1 (RCL_RET_ERROR). USB never enumerated.

Debugging — Breadcrumb Tracing: e² studio debugger has no FreeRTOS task awareness. Always shows idle task when paused. Workaround: volatile int g_breadcrumb set at sequential code points. Pause, read value in Expressions panel.

Error String: rcl_get_error_string() returned garbage ("eF" + nulls). Failure path doesn't set error.

Resolution: Combination of global malloc override + correct heap size. rc=1 was downstream of silent allocation failures.

8.4 Bug #4: USB Device Disappearing

Symptom: Agent connected, 7 topics created, then /dev/ttyACM0 vanished.

Fix: Handle USB_STATUS_SUSPEND as no-op (microros_transports.c line 55–57).

8.5 Bug #5: Permission Denied (errno 13)

Fix: sudo chmod 666 /dev/ttyACM0. Permanent: sudo usermod -a -G dialout $USER.

9. Host-Side Setup: WSL2, ROS 2, and the Agent

9.1 Install WSL2 with Ubuntu 22.04

Admin PowerShell:

```
wsl --install -d Ubuntu-22.04
```

Reboot. Enter later: wsl -d Ubuntu-22.04

9.2 Install ROS 2 Humble

```
sudo apt update && sudo apt install locales -y
sudo locale-gen en_US en_US.UTF-8
sudo update-locale LC_ALL=en_US.UTF-8 LANG=en_US.UTF-8
export LANG=en_US.UTF-8


sudo apt install software-properties-common -y
sudo add-apt-repository universe
sudo apt update && sudo apt install curl -y
sudo curl -sSL
https://raw.githubusercontent.com/ros/rosdistro/master/ros.key \
  -o /usr/share/keyrings/ros-archive-keyring.gpg
echo "deb [arch=$(dpkg --print-architecture) \
  signed-by=/usr/share/keyrings/ros-archive-keyring.gpg] \
  http://packages.ros.org/ros2/ubuntu jammy main" \
  | sudo tee /etc/apt/sources.list.d/ros2.list > /dev/null


sudo apt update
sudo apt install ros-humble-ros-base python3-rosdep build-essential -y
echo 'source /opt/ros/humble/setup.bash' >> ~/.bashrc
source ~/.bashrc
```

9.3 Build the Micro-ROS Agent

```
mkdir -p ~/microros_ws/src && cd ~/microros_ws/src
git clone -b humble https://github.com/micro-ROS/micro_ros_setup.git
cd ~/microros_ws
source /opt/ros/humble/setup.bash
colcon build
source install/local_setup.bash
ros2 run micro_ros_setup create_agent_ws.sh
ros2 run micro_ros_setup build_agent.sh
echo 'source ~/microros_ws/install/local_setup.bash' >> ~/.bashrc
```

9.4 USB Passthrough (usbipd-win)

Install from https://github.com/dorssel/usbipd-win/releases. Admin PowerShell:

```
usbipd bind --hardware-id 045b:2013
usbipd attach --wsl --hardware-id 045b:2013 --auto-attach --unplugged
```

Leave running. --auto-attach --unplugged intercepts before Windows grabs the device.

9.5 Run the Agent and Verify

Terminal 1:

```
sudo chmod 666 /dev/ttyACM0
ros2 run micro_ros_agent micro_ros_agent serial --dev /dev/ttyACM0
```

Press board reset. Agent shows 1 participant, 7 topics, 7 publishers, 7 datawriters.

Terminal 2:

```
ros2 topic list
ros2 topic echo /sensors/room_temp
```

WSL2 DDS Limitation

ros2 topic list may only show /parameter_events and /rosout. This is a WSL2 networking issue, NOT a firmware bug. Use agent -v6 to verify. Works on native Linux.

10. Deployment: Podman Container

10.1 Containerfile

```
mkdir -p ~/microros_container
cat > ~/microros_container/Containerfile << 'EOF'
FROM ros:humble-ros-base
RUN apt-get update && apt-get install -y python3-pip \
    && pip3 install vcstool && rm -rf /var/lib/apt/lists/*
SHELL ["/bin/bash", "-c"]
RUN mkdir -p /microros_ws/src \
    && cd /microros_ws/src \
    && source /opt/ros/humble/setup.bash \
    && git clone -b humble https://github.com/micro-
ROS/micro_ros_setup.git \
    && cd /microros_ws && colcon build \
    && source install/local_setup.bash \
    && ros2 run micro_ros_setup create_agent_ws.sh \
    && ros2 run micro_ros_setup build_agent.sh
ENTRYPOINT ["/bin/bash", "-c", \
  "source /opt/ros/humble/setup.bash && \
   source /microros_ws/install/local_setup.bash && \
   ros2 run micro_ros_agent micro_ros_agent serial --dev /dev/ttyACM0"]
EOF
```

10.2 Build and Run

```
cd ~/microros_container && podman build -t microros-agent .
podman run --rm --device=/dev/ttyACM0 microros-agent
```

11. Troubleshooting Quick Reference

Symptom	Likely Cause	Fix
Board not in Device Manager	Power-only cable or firmware hung	Different cable; debugger check R_USB_Open
"rcl/rcl.h not found"	Include path missing	Add include/ in Compiler → Includes
"conflicting CPU architectures"	libmicroros.a wrong ARM target	Rebuild with -mcpu=cortex-m33
RCL_RET_BAD_ALLOC (rc=10)	malloc returning NULL	Add freertos_malloc_override.c
Heap resets to 1024	FSP regenerated FreeRTOSConfig.h	Re-edit after every Generate Project Content
Agent: errno 13	Permission denied	sudo chmod 666 /dev/ttyACM0
ros2 topic list empty	WSL2 DDS discovery broken	Not a bug; agent -v6; works on native Linux
USB disappears	SUSPEND treated as DETACH	Handle USB_STATUS_SUSPEND as no-op

12. Lessons Learned

12.1 USB Polling Is Non-Negotiable

In bare-metal mode, R_USB_EventGet() must be called in a tight loop or the USB state machine never advances. Not prominently documented by Renesas.

12.2 Fix the Allocator at Every Level

Fixing only the rcl-level allocator is insufficient. rcutils and rmw_microxrcedds call malloc() directly. The global override in freertos_malloc_override.c is the complete fix.

12.3 Compiler -D Flags Can Be Silently Ignored

#ifndef guards in generated headers mean -D flags have no effect if the header defines the value first. Read the actual files.

12.4 WSL2 Is Not Linux

USB requires usbipd-win. DDS multicast discovery is broken. Assume any USB or topic discovery issue is WSL2. Test on native Linux.

12.5 Budget a Full Day for Library Generation

Keep a clean PATH, exclude C++ packages, verify the ARM architecture tag.

13. Project File Reference

13.1 Bare-Metal Project

```
Sensor_Project_USB_Version/
├─ src/
│   ├─ hal_entry.c              ← ALL application logic
│   ├─ hal_warmstart.c          ← Auto-generated startup
│   └─ r_usb_pcdc_descriptor.c  ← USB descriptors
├─ ra_gen/                       ← FSP-generated drivers
├─ ra_cfg/                       ← FSP config headers
└─ configuration.xml             ← FSP Configurator data
```

13.2 Micro-ROS Project

```
Sensor_Project_ROS/
├─ src/
│   ├─ new_thread0_entry.c        ← Micro-ROS node (7 publishers)
│   ├─ sensors.c / sensors.h      ← Sensor abstraction
│   ├─ microros_transports.c / .h ← USB-CDC transport
│   ├─ microros_stubs.c           ← POSIX shims
│   ├─ freertos_malloc_override.c ← Global allocator override
│   └─ usb_descriptor.c           ← CDC-ACM descriptors
├─ micro_ros_renesas2estudio_component/
│   └─ libmicroros/
│      ├─ libmicroros.a            ← Pre-compiled library
│      └─ include/                 ← Micro-ROS headers
├─ ra_cfg/aws/FreeRTOSConfig.h      ← MUST have heap = 200000
└─ configuration.xml                ← FSP Configurator data
```

14. Glossary

Term	Definition
e² studio	Renesas' Eclipse-based IDE for RA microcontrollers
FSP	Flexible Software Package — Renesas' HAL and driver package
FreeRTOS	Real-time OS providing threads, semaphores, memory management
micro-ROS	ROS 2 for microcontrollers with limited resources
libmicroros.a	Pre-compiled static library: entire micro-ROS client stack
micro-ROS agent	Host-side serial-to-DDS bridge
XRCE-DDS	eXtremely Resource Constrained Environments DDS — lightweight protocol between client and agent
USB-CDC (PCDC)	USB Communications Device Class — virtual serial port
DDS	Data Distribution Service — ROS 2's middleware protocol for topic discovery and transport
WSL2	Windows Subsystem for Linux 2 — VM running Linux inside Windows
usbipd-win	Forwards USB devices from Windows into WSL2
Podman	Container runtime (Docker alternative)
QWIIC	SparkFun's 4-pin JST I2C connector system
Breadcrumb Tracing	Debug technique: volatile int set at code points, read in debugger to track execution
pvPortMalloc/vPortFree	FreeRTOS heap allocation functions
_sbrk()	POSIX function for heap extension; not implemented by FSP for clang
CDR	Common Data Representation — serialization format used by DDS for message encoding