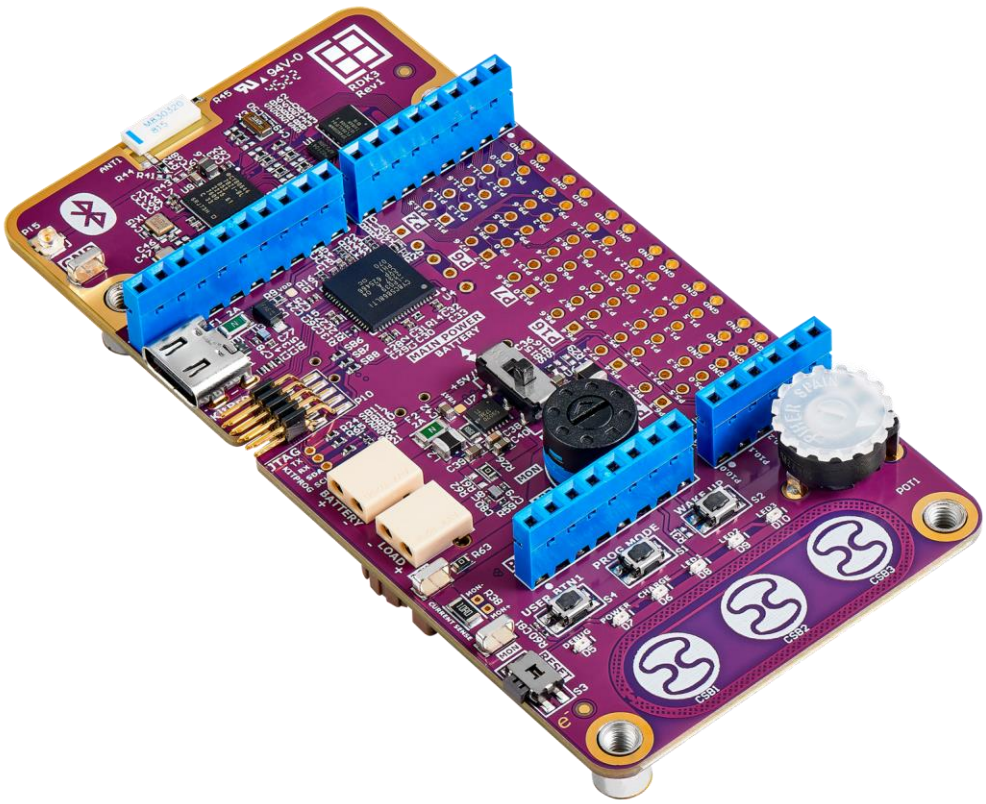
Application Note

Adding a New Sensor to RSS Android App



Versions

Version	Date	Rationale
1.0	November 14, 2023	First release. Autor: ROJ
1.1	May 19, 2025	PSOC 64 disclaimer. Author: KOA.

Legal Disclaimer

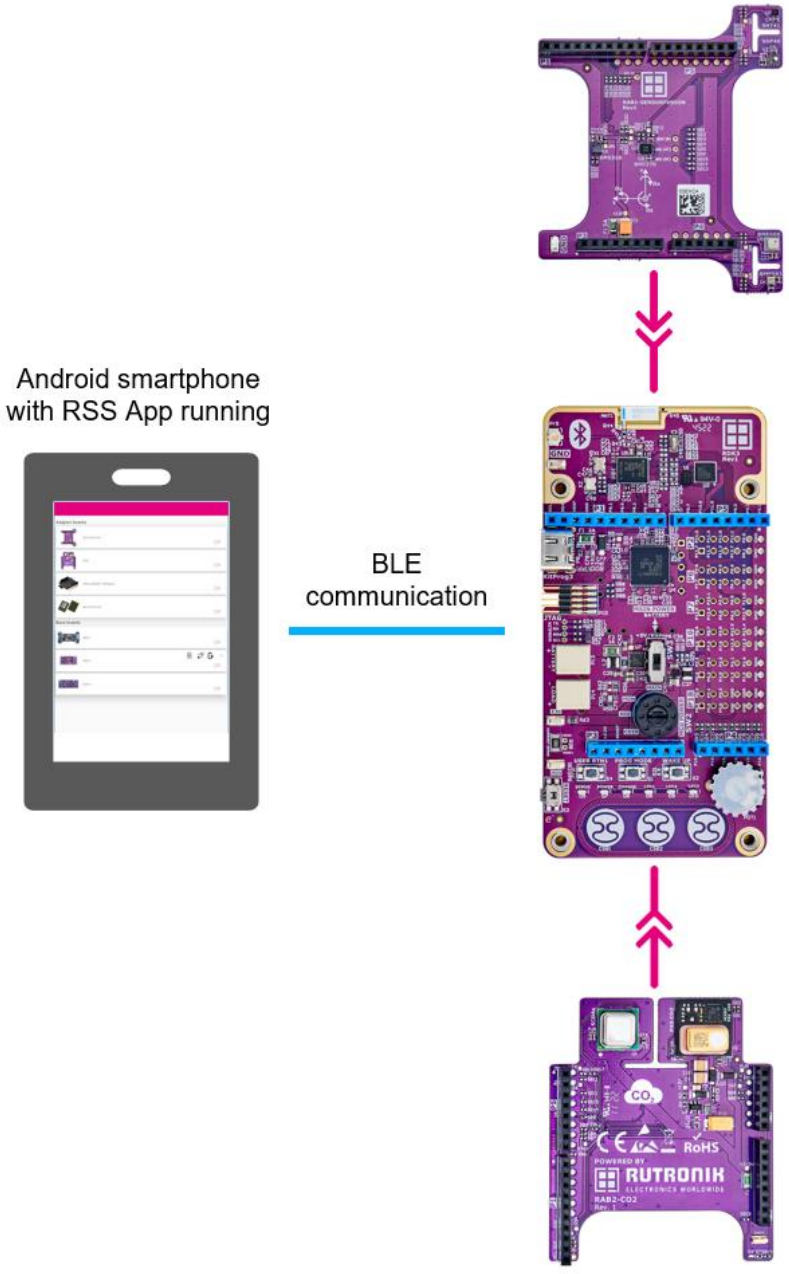
The evaluation board is for testing purposes only and, because it has limited functions and limited resilience, is not suitable for permanent use under real conditions. If the evaluation board is nevertheless used under real conditions, this is done at one's responsibility;
any liability of Rutronik is insofar excluded.

Table of Contents

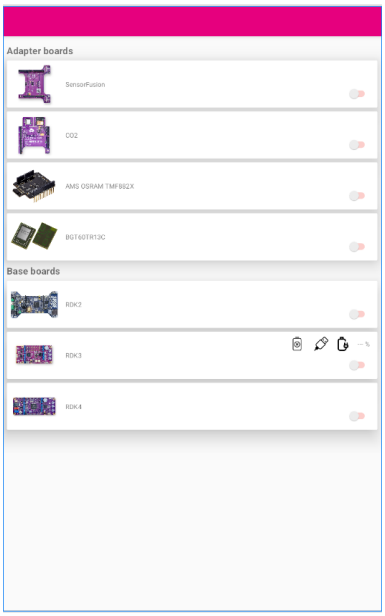
Infineon has discontinued the PSOC™ 64 Secured MCU product line. As a result, the CYB06447BZI-BLD53 MCU used in the RDK3 is not recommended for new designs. The Infineon CY8C6347BZI-BLD53 MCU may be considered a suitable alternative.

Overview of Solution

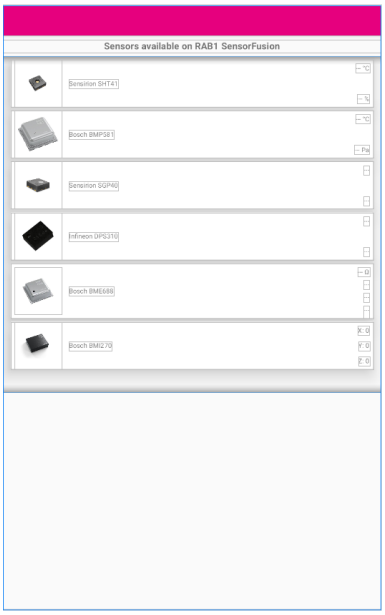
The Android app enables to display the values measured by different sensors. The RDK3 enables to send those measurements using BLE (Bluetooth Low Energy), so no physical connection between the smartphone and the sensor is needed.



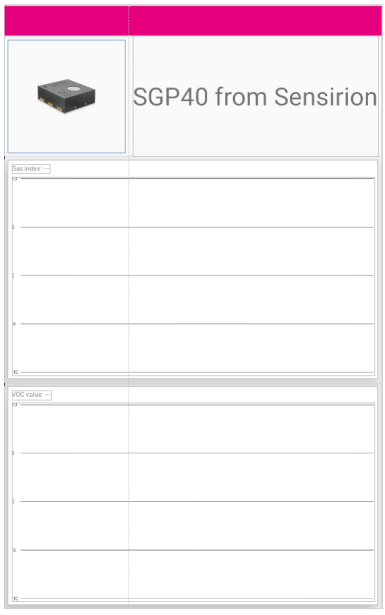
Some screenshots of the app are shown below.



Available boards



Sensors on the SensorFusion board



Live values of the SGP40

Adding a new sensor

The solution is composed of two softwares. One software is running on the microcontroller of the RDK3 (we will call it **BSP**) and another software running on the smartphone (we will call it **APP**).

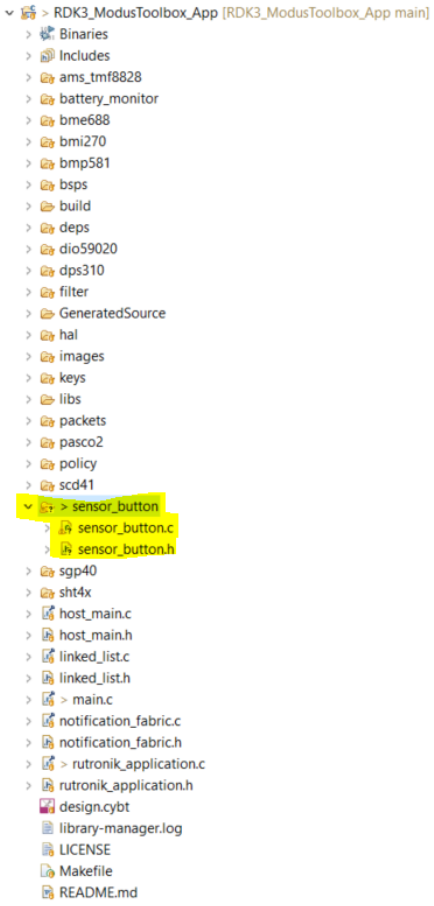
The BSP is written in C and can be edited using Modus Toolbox. For more information about Modus Toolbox, follow the link.

The APP is written in Java and can be edited using Android Studio. For more information about Android Studio, follow the link.

For this example, we will add a new sensor: a button (that is already present on the RDK3 ☺).

New sensor inside BSP

Create a directory for your source code



First, try to follow the structure that has been chosen: one directory <-> one sensor. As you can see at the screenshot, every sensor has its own directory.

So, for our example, we will create a directory called **sensor_button** and put our source code inside.

We will implement two functions enabling to initialize the module and to get the value. See appendix for the implementation of the functions.

```
/**
 * @brief Initialize the module
 *
 * Configure the GPIO linked with the user button to be a GPIO input
 *
 * @retval 0 Success
 * @retval !=0 An error occurred
 */
int sensor_button_init();

/**
 * @brief Get the state of the button
 *
 * @retval true Button is pressed
 * @retval false Button is not pressed
 */
bool sensor_button_get_state();
```

Modify notification_fabric.c

Create your own function to serialize the data you want to send over BLE. The structure of a binary packet sent from the RDK3 to the App looks like:

Sensor ID	Data size	Data	CRC
2 bytes	1 byte	“Data size” bytes	1 byte (computed over sensor ID, data size, and data)

Our packet to send the status of the button (one byte) will then look like:

Sensor ID	Data size	Data	CRC
0xF001 Custom sensor ID should start with 0xFxxx	1	0 or 1	Depends on the data ☺

The implementation of our function will then look like below:

```
notification_t* notification_fabric_create_for_sensor_button(bool button_state)
{
        const uint8_t data_size = 1;
        const uint8_t notification_size = data_size + notification_overhead;
        const uint16_t sensor_id = 0xF001;

        uint8_t* data = (uint8_t*) malloc(notification_size);

        data[0] = (uint8_t) (sensor_id & 0xFF);
        data[1] = (uint8_t) (sensor_id >> 8);

        data[2] = data_size;

        data[3] = button_state;

        data[notification_size - 1] = compute_crc(data, notification_size - 1);

        notification_t* retval = (notification_t*) malloc(sizeof(notification_t));
        retval->length = notification_size;
        retval->data = data;

        return retval;
}
```

Modify rutronik_application.c

First you need to initialize the module. Place the call to your initialization function inside the **`rutronik_application_init`** function which will be called at the start of the program.

```
void rutronik_application_init(rutronik_application_t* app)
{
       // ……


        if (sensor_button_init() != 0)
        {
               // Button not available :-(
        }
}
```

Then you need to modify the function **rutronik_application_do** to read the state of the button and send it per BLE:

```
void rutronik_application_do(rutronik_application_t* app)
{
        if (host_main_is_ready_for_notification() == 0) return;

        host_main_add_notification(
                        notification_fabric_create_for_sensor_button(sensor_button_get_state()));

       // ……
}
```

That's it for the BSP. No more modifications are necessary. Build the project and program the RDK3 with your software.

Problem because of limited flash space

Remark: because of limited flash space, you might run out of memory when adding your sensor. You will get such an error when building the project:

```
/gcc/arm-none-eabi/10.3.1/../../../../arm-none-eabi/bin/ld.exe: C:/rutronik/github/RDK3_ModusToolbox_App/build/RDK3/Custom/rdk3-modustoolbox-app.elf section `.data' will not fit in region `flash'
/gcc/arm-none-eabi/10.3.1/../../../../arm-none-eabi/bin/ld.exe: region `flash' overflowed by 2720 bytes
```

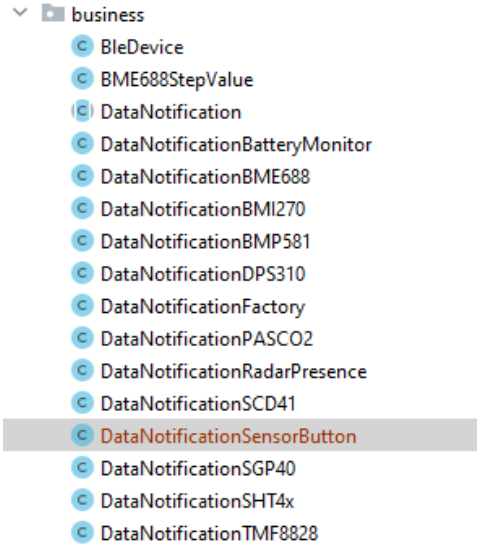
In that case, remove sensor that you do not use by changing the properties inside the **Makefile**:

```
# Add additional defines to the build process (without a leading -D).
# Possible defines:
# AMS_TMF_SUPPORT => To enable the support of the time of flight board
# BME688_SUPPORT => To enable the support of the BME688 sensor
DEFINES=AMS_TMF_SUPPORT BME688_SUPPORT
```

Just comment out DEFINES to gain some space.

New sensor inside Android App

Create your data type to receive the values



You will need to create the file **DataNotificationSensorButton** (which will enable to extract relevant information out of the bytes received by BLE) and you will have to modify the **DataNotification** file.

Let's start with modifying the **DataNotification** file by adding the SENSORBUTTON value:

```
public enum Sensor
{
    …
    SENSORBUTTON,
    UNKNOWN;

    public static Sensor fromInteger(int x) {
        switch(x) {
            …
            case 0xF001:
                return SENSORBUTTON;
        }
        return UNKNOWN;
    }
}
```

Our `DataNotificationSensorButton` class looks like:

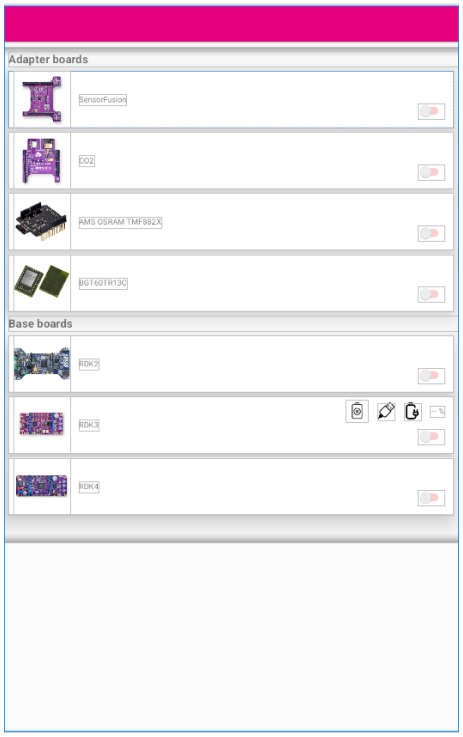
```
package com.rutronik.rdk3.business;

public class DataNotificationSensorButton extends DataNotification {
    public DataNotificationSensorButton(Sensor sensorType) {
        super(sensorType);
    }

    public boolean getSensorButtonState() {
        if (this.data[0] == 0) return  false;
        return true;
    }
}
```

Display the value

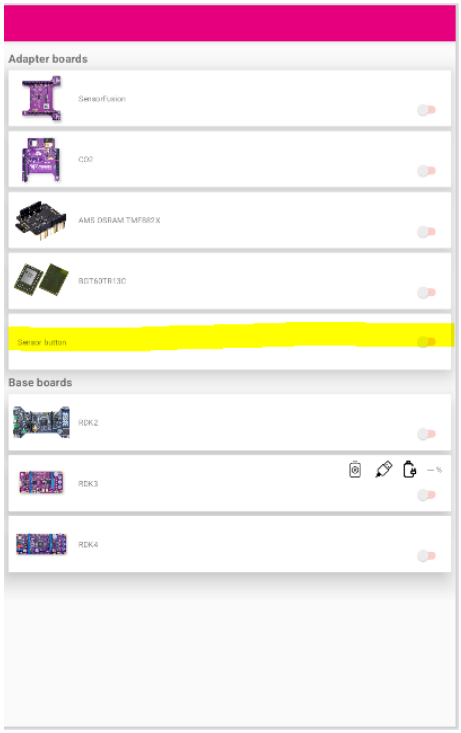
In order to keep that application note simple, we will not create a new activity to display the state of the button but an existing one.

We will modify the **BoardsOverviewActivity** to display the state.

First, we will modify the layout to add a new box containing our button:



Before



After

Then we have to edit the **BoardsOverviewActivity.java** file and modify the

```
private final BroadcastReceiver gattUpdateReceiver =new BroadcastReceiver() {
    @Override
    public void onReceive(Context context, Intent intent) {
        final String action = intent.getAction();
        if (action == null) return;

        switch (action){
            […]
            case BleCommunicationServiceIntents.DATA_NOTIFICATION:
                byte [] content =
intent.getByteArrayExtra(BleCommunicationServiceIntents.EXTRA_VALUE);
                if (content == null) return;

                DataNotification notification = DataNotificationFactory.GetNotification(content);
                if (notification != null) {
                    switch (notification.getSensorType()) {
                        case SENSORBUTTON:
                            Switch sensorButtonSwitch =
((Switch)findViewById(R.id.sensorButtonSwitch));
                            DataNotificationSensorButton sensorButtonData =
(DataNotificationSensorButton) notification;
                           sensorButtonSwitch.setChecked(sensorButtonData.getSensorButtonState());
                            break;
```

That's it! When connected, the status of the switch in the App should now reflect the status of the physical button.

Appendix

sensor_button.h

```
#ifndef SENSOR_BUTTON_SENSOR_BUTTON_H_
#define SENSOR_BUTTON_SENSOR_BUTTON_H_

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

/**
 * @brief Initialize the module
 *
 * Configure the GPIO linked with the user button to be a GPIO input
 *
 * @retval 0 Success
 * @retval !=0 An error occurred
 */
int sensor_button_init();

/**
 * @brief Get the state of the button
 *
 * @retval true Button is pressed
 * @retval false Button is not pressed
 */
bool sensor_button_get_state();

#endif /* SENSOR_BUTTON_SENSOR_BUTTON_H_ */
```

sensor_button.c

```
#include "sensor_button.h"

#include "cyhal_gpio.h"          // Needed to get access to the GPIOs
#include "cycfg_pins.h"          // Needed to know the address of the USER_BTN
#include "cybsp_types.h"// Needed for CYBSP_BTN_OFF and CYBSP_BTN_PRESSED

int sensor_button_init()
{
        if (cyhal_gpio_init(USER_BTN, CYHAL_GPIO_DIR_INPUT, CYHAL_GPIO_DRIVE_PULLUP, CYBSP_BTN_OFF)
!= CY_RSLT_SUCCESS)
        {
                return -1;
        }
        return 0;
}

bool sensor_button_get_state()
{
        if (cyhal_gpio_read(USER_BTN) == CYBSP_BTN_OFF)
        {
                return false;
        }
        return true;
}
```