Bosch Sensortec

COINES SDK

COmmunication for INertial and Environmental Sensors SDK

# User Manual

| | |
|---|---|
| Document revision | 2.1 |
| Document release date | June 2025 |
| Document number | BST-DHW-AN013 |
| Technical reference code(s) | n.a. |
| Notes | Data and descriptions in this document are subject to change without notice. Product photos and pictures are for illustration purposes only and may differ from the real product appearance. |

## Contents

## Acronyms and abbreviations

| | |
|---|---|
| APP3.X | Application Board 3.0 and Application Board 3.1 |
| BLE | Bluetooth Low Energy |
| COINES | COmmunication with Inertial and Environmental Sensors |
| SDK | Software Development Kit |
| DFU | Device Firmware Upgrade |
| DTR | Data Terminal Ready |
| MCU | Microcontroller Unit |
| MEMS | Micro-Electro-Mechanical Systems |
| USB | Universal Serial Bus |

## 1 Introduction

Bosch Sensortec offers a toolkit for evaluation of it's sensor products.The toolkit consists of 3 elements:

1. **Engineering Board**: Application board named APP3.X in this document, serves as interface translator from the sensor interface ($I^2C$ or SPI) to a USB interface, allowing PC software to communicate with the sensor on the shuttle board. Nicla Sense ME board combines four state-of-the-art sensors from Bosch Sensortec (BHI260AP, BMP390, BMM150 and BME688) in the Arduino ecosystem.

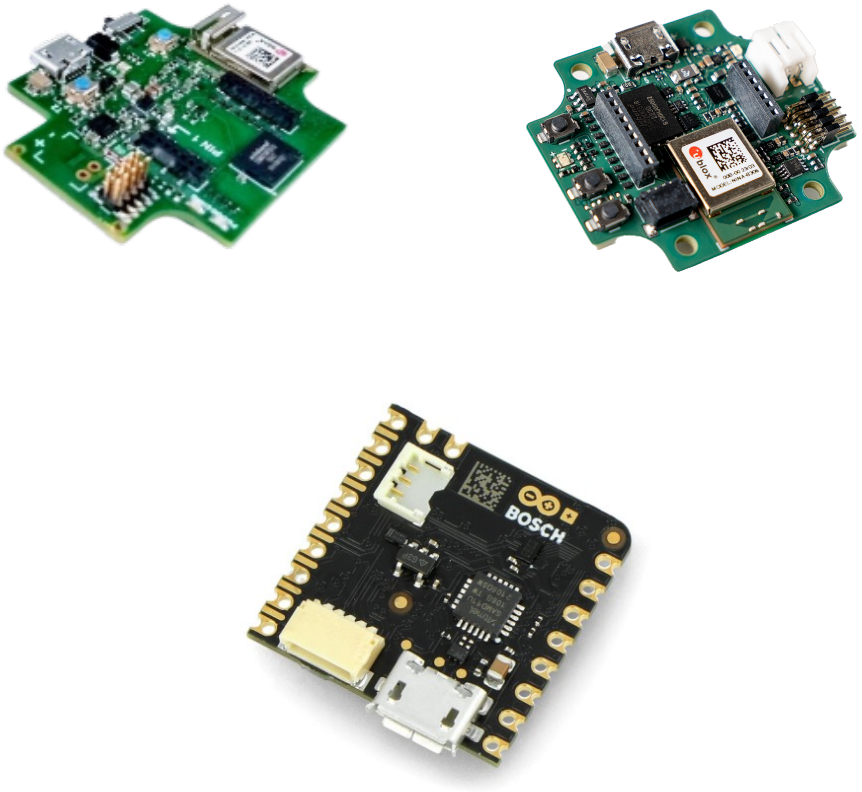



Fig. 1: Application Board 3.0/3.1/Nicla Sense ME

2. **Sensor Shuttle board**: A sensor specific shuttle board also known as breakout board is a PCB with the sensor mounted on it. The shuttle board allows easy access to the sensor pins via a simple socket and can be directly plugged into the Bosch Sensortec's Application Boards.


Fig. 2: APP3.X sensor shuttle board

3. **COINES SDK**: COINES SDK provides a low-level interface for communication with Bosch Sensortec's Engineering Boards enabling access to their MEMS sensors through sample applications and SensorAPI. For a detailed description, refer to the following sections.

## 2 Introduction to COINES SDK

**CO**mmunication with **IN**ertial and **E**nvironmental **S**ensors (COINES) is an SDK (Software Development Kit), implemented in C as a programming language that provides a low-level interface to Bosch Sensortec's Engineering Boards. The user can access Bosch Sensortec's MEMS sensors through this C interface.

**Key Features of the SDK**:

- **Sensor Communication:** I2C and SPI read/write (8-bit and 16-bit support)
- **EEPROM Access:** Read and write operations on the sensor shuttle
- **GPIO Control:** Configuration and management
- **Power Management:** VDD and VDDIO configuration
- **Peripheral Configuration:** LED, button, and timer settings
- **Real-Time OS Support:** FreeRTOS compatibility

COINES SDK can be used with SensorAPI of the sensor which is available at https://github.com/BoschSensortec. The user can modify, compile and run the sample applications in COINES SDK and SensorAPI.

The working environment consists of the following:

- A Bosch Sensortec MEMS sensor on a shuttle board mounted on the socket of Bosch Sensortec's Application Board (APP3.X).
- Windows, Linux or Mac PC to which the Engineering Board is connected via USB or BLE.
- The release of the COINES SDK is available at https://www.bosch-sensortec.com/software-tools/tools/coines/
- C compiler is also required (for details, see sections below)

## 3 COINES SDK usage

The following diagram represents COINES SDK usage.

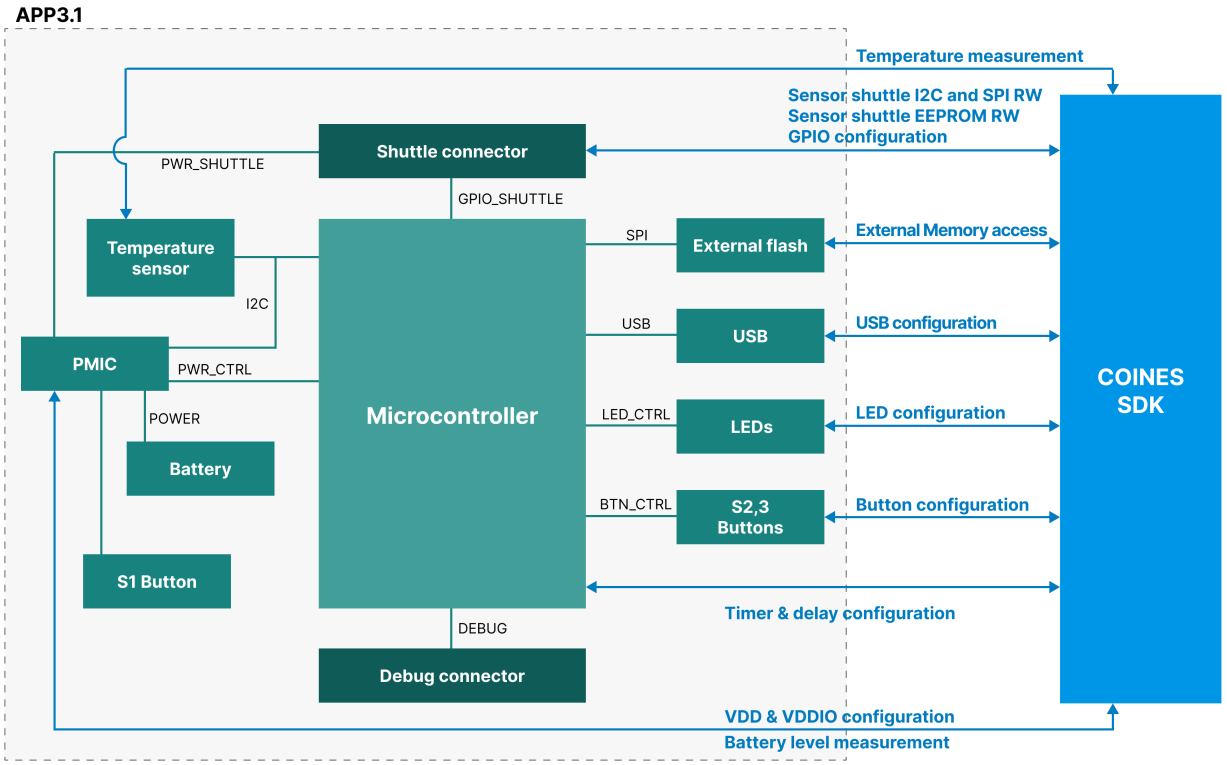



Fig. 3: COINES SDK usage with APP3.1

## 4 Installation

COINES SDK can be used on any recent PC or laptop system. The hardware must provide a USB interface.

COINES SDK runs on recent versions of Windows, Linux, and Mac Operating systems.

### 4.1 Installation (Windows)

#### 4.1.1 System requirements

The supported OS versions are Windows 10 and 11.

#### 4.1.2 Installation of COINES SDK

Follow the steps below to install COINES SDK:

1. Download the latest version of COINES SDK from Bosch Sensortec website.
2. Run the Installer
3. Accept the End User License Agreement and click **Next**

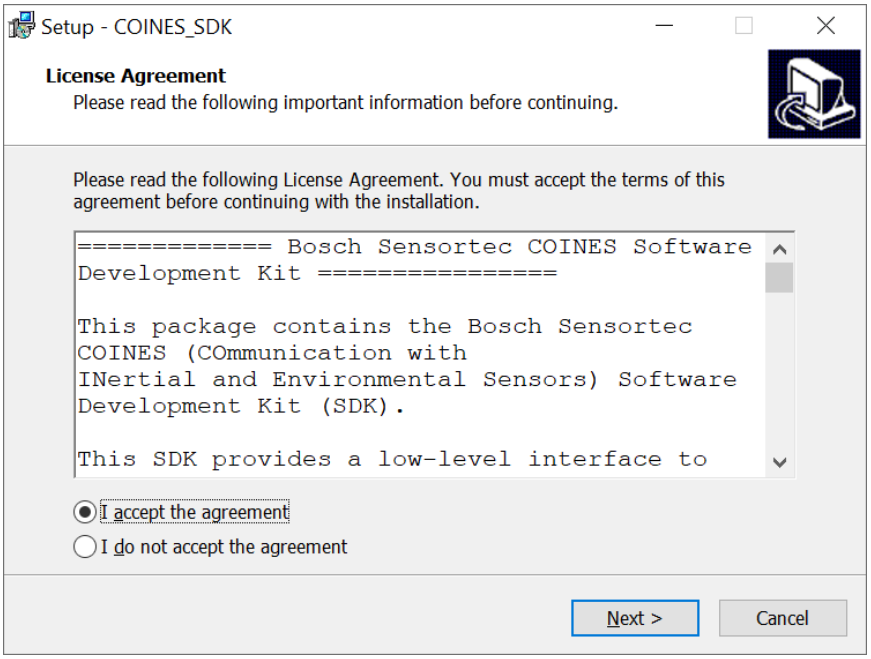



Fig. 4: Windows installer end user agreement dialog

**4.** Click **Install** to start the Installation.

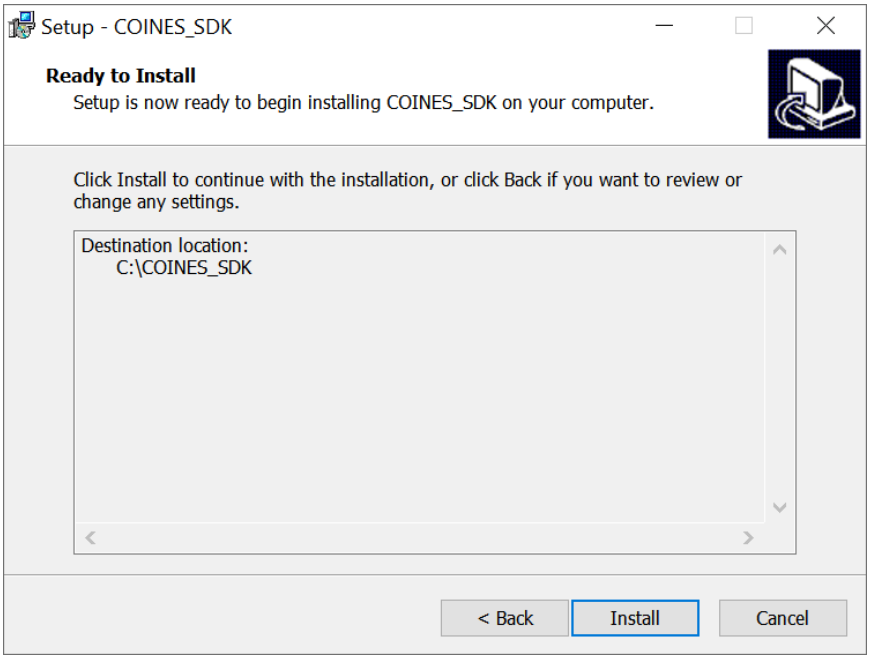



Fig. 5: Windows install dialog

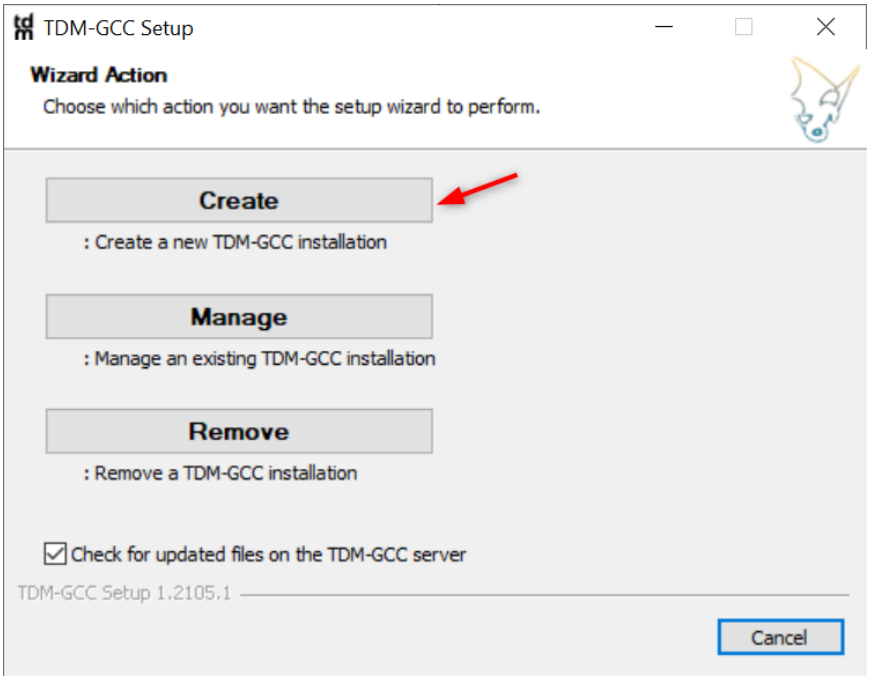
### 4.1.3 Installation of compiler environment

COINES SDK examples can be built using GNU C compiler (GCC). There are various distributions of GCC. TDM-GCC is easy to install and therefore preferred for COINES SDK. TDM GCC is based on MinGW GCC.

If you have already installed GCC (MinGW/Cygwin/MSYS2 GCC) and added it to the 'PATH' environmental variable, you can skip the compiler installation.

The steps to the install the compiler environment are as follows:

1. Download the TDM32/TDM64 bundle (link). **Use TDM32 bundle if your Windows OS is 32-bit and TDM64 bundle if 64-bit.**
2. Start the Installer. Ensure that the option **Check for updated files on the TDM GCC server** is deselected. Click **Create** and proceed with the installation.
3. If you intend to run the COINES SDK example on the Application Board's microcontroller, install the latest version of the GNU Embedded Toolchain for ARM for Windows. Make sure you have selected **Add to PATH**.
4. Click **Install**.
5. Click **Finish**.



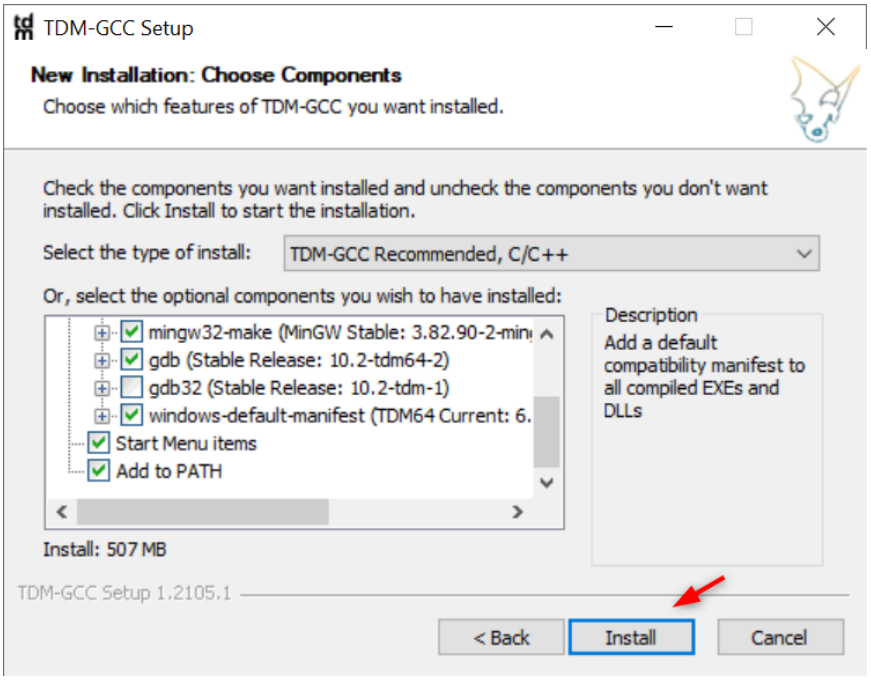



Fig. 6: TDM-GCC installation dialog

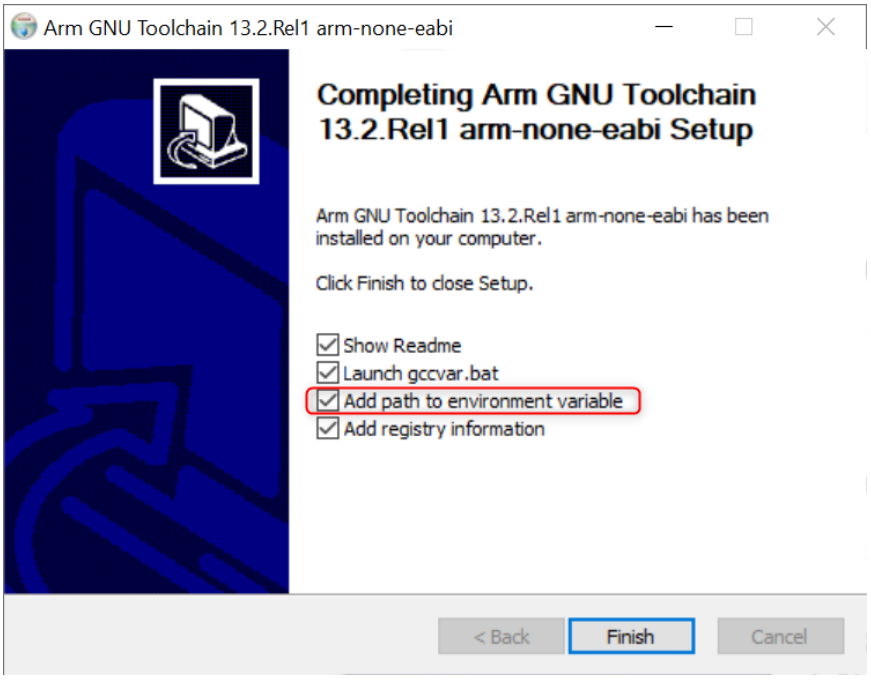



Fig. 7: GNU ARM Toolchain installation

## 4.2 Installation (Linux/MacOS)

### 4.2.1 System requirements

- The supported Linux OS versions are Debian based - Ubuntu 18.04 and 22.04.
- The supported macOS versions are MacOS 13 Ventura ( 13.4.1 and 13.5.2 ) and MacOS 14 Sonoma ( 14.3.1).

### 4.2.2 Installation of COINES SDK

Follow the steps below to install COINES SDK:

1. Download the installer.
2. Use the command `cd` to go to the directory where the installer is located and make the installer executable:

   `$ chmod +x COINES_SDK_vX.Y.sh`
3. Ensure that you are connected to the Internet before running the installer:

   `$ ./COINES_SDK_vX.Y.sh`
4. Accept the End User License agreement.

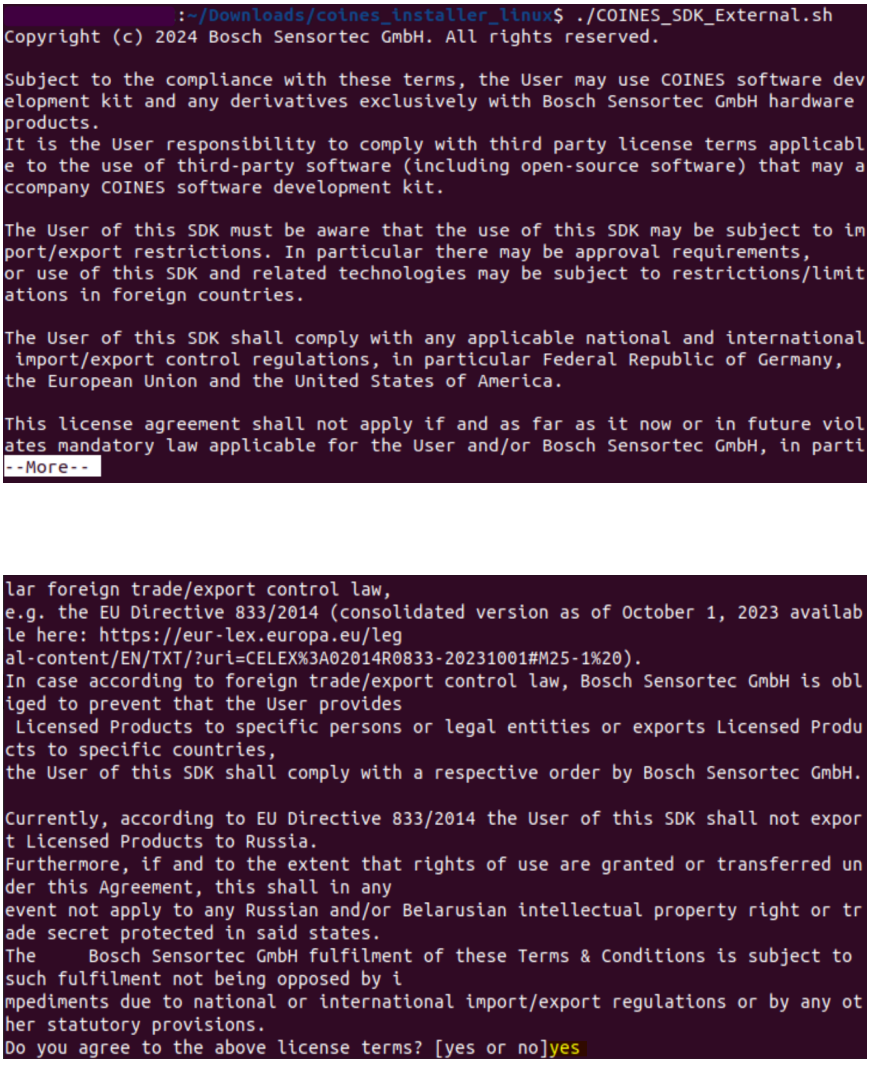



Fig. 8: Linux installer end user agreement

5. The installer will prompt you if the required dependencies/packages are not installed. (This step requires root privileges.)

### 4.2.3 Installation of compiler environment

On a Debian or Redhat based Linux distro, the installer prompts for installation of missing dependencies, `gcc`, `make`, and `libusb-dev` packages. If the installation fails, the using can manually install the dependencies.

- Debian based distros - `gcc`, `make`, `libusb-1.0-0-dev`, `dfu-util` , `libdbus-1-dev`
- Redhat based distros - `gcc`, `make`, `libusbx-devel`, `dfu-util`, `dbus-devel`
- MacOS - `libusb`, `dfu-util`

If you intend to run the COINES SDK example on the Application Board's microcontroller, download the latest version of GNU Embedded Toolchain for ARM for Linux and extract the package. Add the compiler to the PATH variable by editing `$HOME/.bashrc` or a similar file like `/etc/profile or /etc/environment`.

# 5 Using COINES SDK to access the sensor on Engineering Board

## 5.1 Running examples on the MCU of the Application Board

### 5.1.1 Working principle

The COINES SDK can be cross-compiled on the PC and then downloaded into the memory of the Application Board and executed there. The user has the option to download the generated binary either into the flash memory or the RAM memory.

Downloading the COINES SDK example to APP3.X flash memory will overwrite the default firmware. To update the firmware again, refer to section 14.

In this configuration, the COINES SDK layer provides a simple abstraction on top of the MCU BSP (i.e. board level support layer of the microcontroller). Any `printf` command will now not output to the console, but rather to the USB connection, which appears as a virtual COM port on the PC side.

This mode facilitates the execution of many time-critical operations on the sensor, such as fast reading of FIFO content at high data rates.

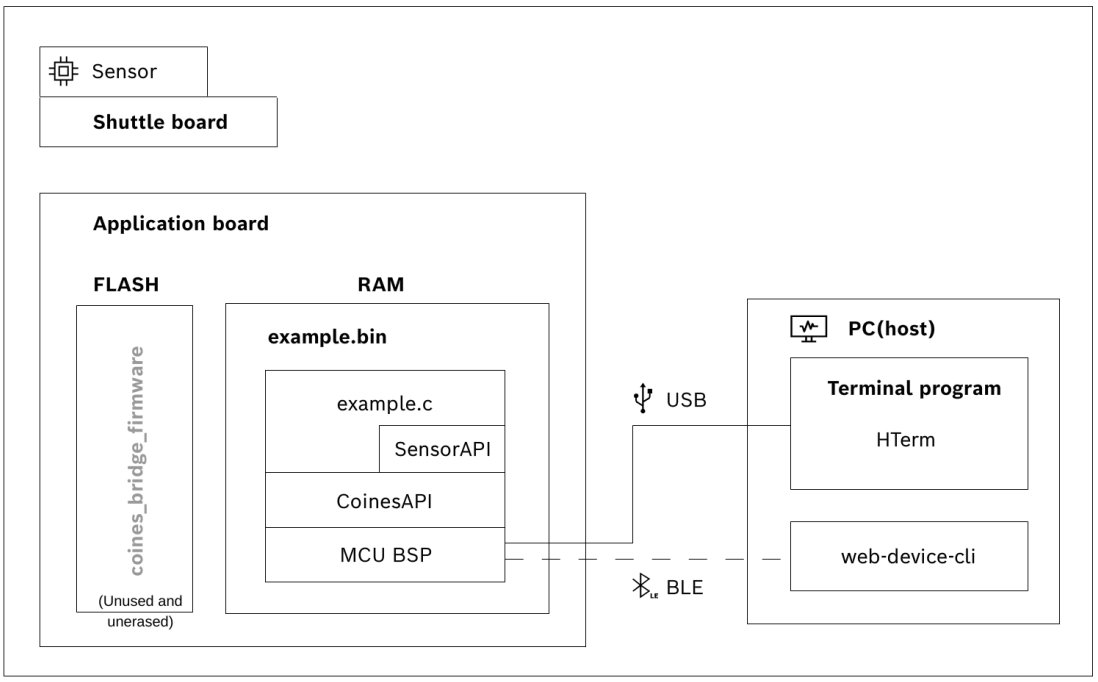



Fig. 9: Working principle: Running example on the MCU of the Application Board at LOCATION=RAM

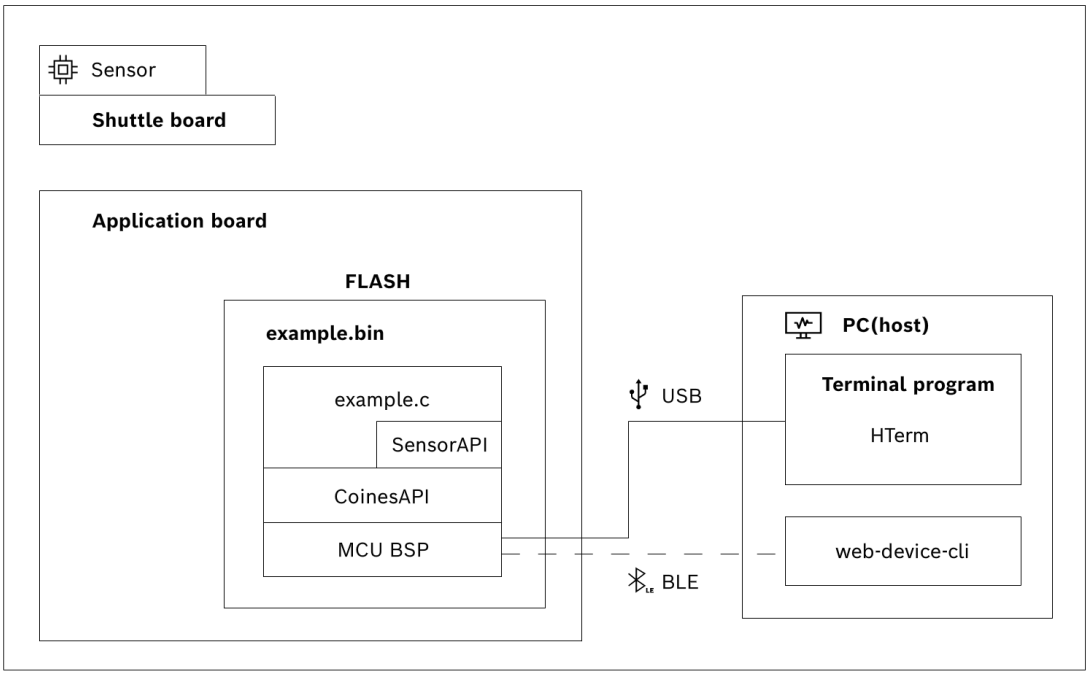



Fig. 10: Working principle: Running example on the MCU of the Application Board at LOCATION=FLASH

### 5.1.2 Getting started

To get started with the example execution, follow these steps:

1. Make sure the GNU Embedded Toolchain for ARM is installed on your PC and added to the environmental variable `PATH`.
2. Connect the Application Board via USB, with the sensor shuttle board mounted.
3. Open the command prompt or the terminal.

4. Use the command `cd` to go to the directory where the example to be built is located.

### 5.1.3 Interfacing via BLE

The procedure to interface via BLE involves these steps:

1. Open the script to be executed (in case of SensorAPI - common.c file in the selected example folder) in your IDE.
2. Change COINES_COMM_INTF_USB to COINES_COMM_INTF_BLE
3. Change all print statments:

   `printf(...) to fprintf(bt_w,...)`
4. Now follow the steps from 1 - 4 in the section 5.1.2.

### 5.1.4 Cross compiling

To compile and download an example to the Engineering Board's microcontroller, type any of the build commands below based on available Engineering Board type and target memory location. Use '`mingw32-make`' (TDM-GCC/MinGW) or '`make`' (Linux/Cygwin/MSYS2/MacOS) for compilation.

| Function | Command |
|---|---|
| Download example to APP3.0 MCU RAM | `mingw32-make LOCATION=RAM TARGET=MCU_APP30 download` |
| Download example to APP3.0 MCU FLASH | `mingw32-make LOCATION=FLASH TARGET=MCU_APP30 download` |
| Download example to APP3.1 MCU RAM | `mingw32-make LOCATION=RAM TARGET=MCU_APP31 download` |
| Download example to APP3.1 MCU FLASH | `mingw32-make LOCATION=FLASH TARGET=MCU_APP31 download` |

*Note*: Nicla board programs can only be executed as PC target at this moment.

### 5.1.5 Viewing the results

The ways to view the execution results are outlined as follows:

1. Use a Serial Terminal application to view output.
   - Windows - PuTTY, HTerm, etc.,
   - Linux - `cat` command. Eg: `cat /dev/ttyACM0`
   - macOS - `screen` command. Eg: `screen /dev/tty.usbmodem9F31`

   *Note*: The binary on the MCU will be executed once the serial port is opened. The port must be opened including the DTR signal set, otherwise the binary will not be executed. Some terminal programs such as HTerm allow explicit setting of the DTR signal.
2. For bluetooth communication, connect the Application Board to another power source and keep it within the BLE range. Use any of the below tools to view the output:
   - Android app - Serial Bluetooth terminal
   - Website - Web Device CLI
   - Python script - `\tools\ble-nus-term\ble-nus-term.py`

### 5.1.6 Data logging

The user can use any serial terminal program to access and store the provided data via a virtual COM port, e.g., HTerm, which has a "Save output" option to store logs.

## 5.2 Running examples on the PC side

### 5.2.1 Working principle

When compiling the COINES SDK for PC side, the COINES SDK layer provides an abstraction of the embedded environment on the host side. COINES SDK library provides read and write functions for $I^2C$ and SPI on PC side. These functions receive the arguments of the user input (i.e. what register address to read from) and tunnels them through the USB connection to the Application Board, where they are fed into the embedded $I^2C$ and SPI functions and are executed to access the sensor. Any result or response from those functions is tunneled back to the PC side and provided to the example application.

This approach allows easy and flexible programming and allows integrating the example code into other applications or adding advanced logging options. The drawback is that the code is not executed in real-time in this mode, as it runs on a multitasking operating system. The examples can also be run on the MCU side (see section 5.1) to overcome this drawback.

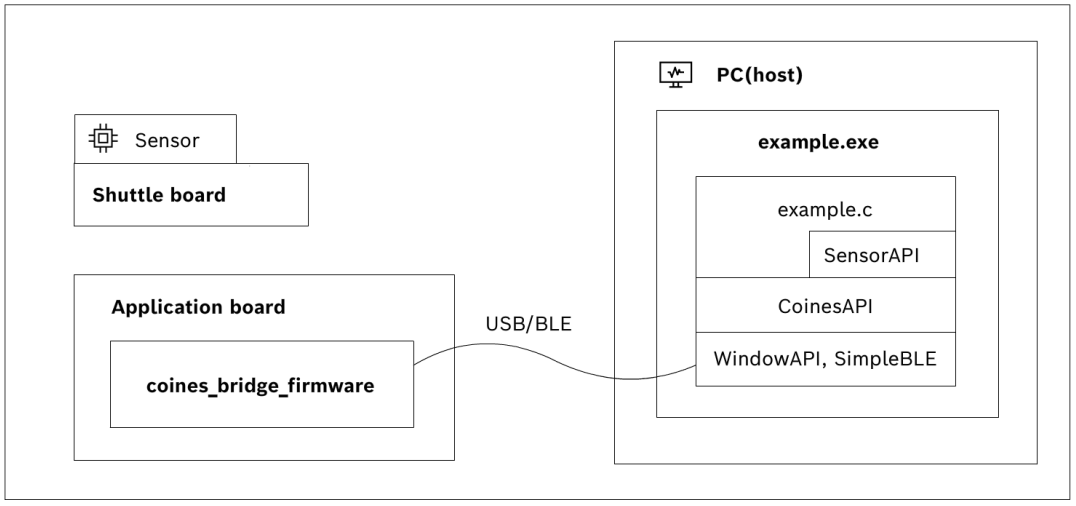



Fig. 11: Working principle: Running example on PC side

### 5.2.2 PC side implementation

This setup has the challenge of lacking the real-time capabilities known from a pure microcontroller environment. To overcome this, the coinesAPI offers streaming functions, which allow the user to schedule data readout directly on the microcontroller, either based on a data interrupt coming from the sensors or the microcontroller's timer. The scheduler waits for the configured interrupt (sensor interrupt or timer interrupt) and reads out areas of the register map, which the user can configure.

As an example, the user could choose to read out the 6 bytes from the register map of a certain inertial sensor, containing the sensor data of three axis (2 bytes per axis). If the user would configure e.g., a readout once per milliseconds, the result would be a data stream of three-axis sensor data at a rate of 1 kHz.

### 5.2.3 Getting started

To get started with example execution, follow these steps:

1. Connect the Application Board via USB, with the sensor shuttle board mounted.
2. Refer to section 14 and update the COINES Bridge firmware to the board.
3. Open the command prompt or the terminal.
4. Use the command `cd` to go to the directory where the example that is to be built is located.

*Note*: Some examples may not compile for both PC and MCU targets. Refer to the example documentation or simply the example name (e.g., examples that can only be compiled for the PC are named with a following '_pc').

### 5.2.4 Interfacing via BLE

The procedure to interface via BLE involves these steps:

1. Open the script to be executed (in case of SensorAPI - common.c file in the selected example folder) in your IDE.
2. Change COINES_COMM_INTF_USB to COINES_COMM_INTF_BLE.
3. Follow steps 1 to 4 in section 5.2.3.

### 5.2.5 Compiling

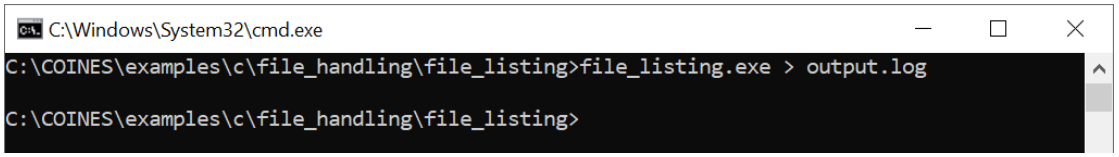
To compile an example on the PC side, execute the command "mingw32-make TARGET=PC". Use '`mingw32-make`' (TDM-GCC/MinGW) or '`make`' (Linux/Cygwin/MSYS2/MacOS) for compilation.

### 5.2.6 Viewing the results

The user can view the results after running the output executable from the PC command prompt. To view the ouput via BLE, connect the Application Board to another power source and keep it within the BLE range and run the executable on the PC.

### 5.2.7 Data logging

The user can use the terminal's output redirection command to store the result of a command/executable in a file, as demonstrated below.




## 5.3 Project Cleanup

The commands to clean build files are listed below:

- "`mingw32-make clean`" or "`make clean`" - Removes the compilation artifacts specific to the current project.
- "`mingw32-make clean-all`" or "`make clean`" - Removes both the project-specific compilation artifacts and artifacts associated with the COINES SDK itself, including backend, BSP, etc.

# 6 Using COINESPY to access the sensor on the Engineering Board

## 6.1 Introduction to the COINESPY library

The COINESPY library provides a Python interface for interacting with Bosch Sensortec's Engineering Boards.

The library offers the following range of functionalities:

- Control VDD and VDDIO of sensor
- Configure SPI and $I^2C$ bus parameters
- Read and write into registers of sensors from Bosch Sensortec via SPI and $I^2C$
- Read and write digital pins of the Application Board

## 6.2 Installation

The COINESPY module can be installed using pip:

```
pip install coinespy
```

The module can be found at https://pypi.org/project/coinespy/. It is highly recommended to test the following script examples\python\coinespy_test.py in the COINES SDK installation or Refer to 8.2.1 to check if the installation was successful.

# 7 Using SensorAPI with COINES SDK

## 7.1 SensorAPI

Bosch Sensortec recommends using SensorAPI to communicate with the sensors. A SensorAPI, an abstraction layer written in C, makes it much more convenient for the user to access the register map of the sensor, configure certain functionality, and obtain certain information from it.

To make use of SensorAPI, some function pointers must be set to the appropriate read/write functions of the selected bus on the system (either $I^2C$ or SPI), as well as one function pointer to a system's function causing delays in milliseconds.

To execute C code using SensorAPI, the COINES SDK API provides the read, write and delay functions. These functions are wrapper functions, embedding the actual SensorAPI payloads into a transport package and sending this via USB or BLE to the Engineering Board, where the payload is translated into corresponding SPI or $I^2C$ messages and sent to the sensor on the shuttle board. The mapping would look similar to the one below.

```
#include "coines.h"
#include "bst_sensor.h"

struct bst_sensor_dev sensordev;
....
....
sensordev.intf = BST_SENSOR_I2C_INTF; // SPI - BST_SENSOR_SPI_INTF
sensordev.read = coines_read_i2c; // coines_read_spi
sensordev.write = coines_write_i2c; // coines_write_spi
sensordev.delay_ms = coines_delay_usec;
```

For the description of COINES SDK functions used, refer to 16.3.

## 7.2 Downloading SensorAPI

In order to download SensorAPI, follow the steps below:

- Download SensorAPI repo using the Download zip option for selected sensors from the BoschSensortec GitHub https://github.com/BoschSensortec.
- Unzip the downloaded SensorAPI repo to `\examples`.
- Rename the unzipped folder to the sensor name, e.g., `\examples\bmi270`, and change the directory to an example folder to execute it.

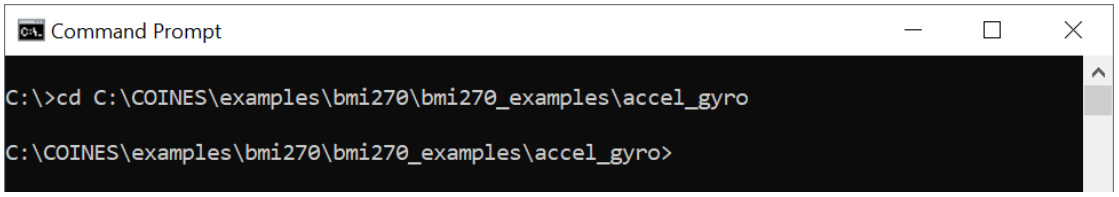
## 7.3 Running example on MCU side

Here are the requirements and instructions to run examples on the MCU side:

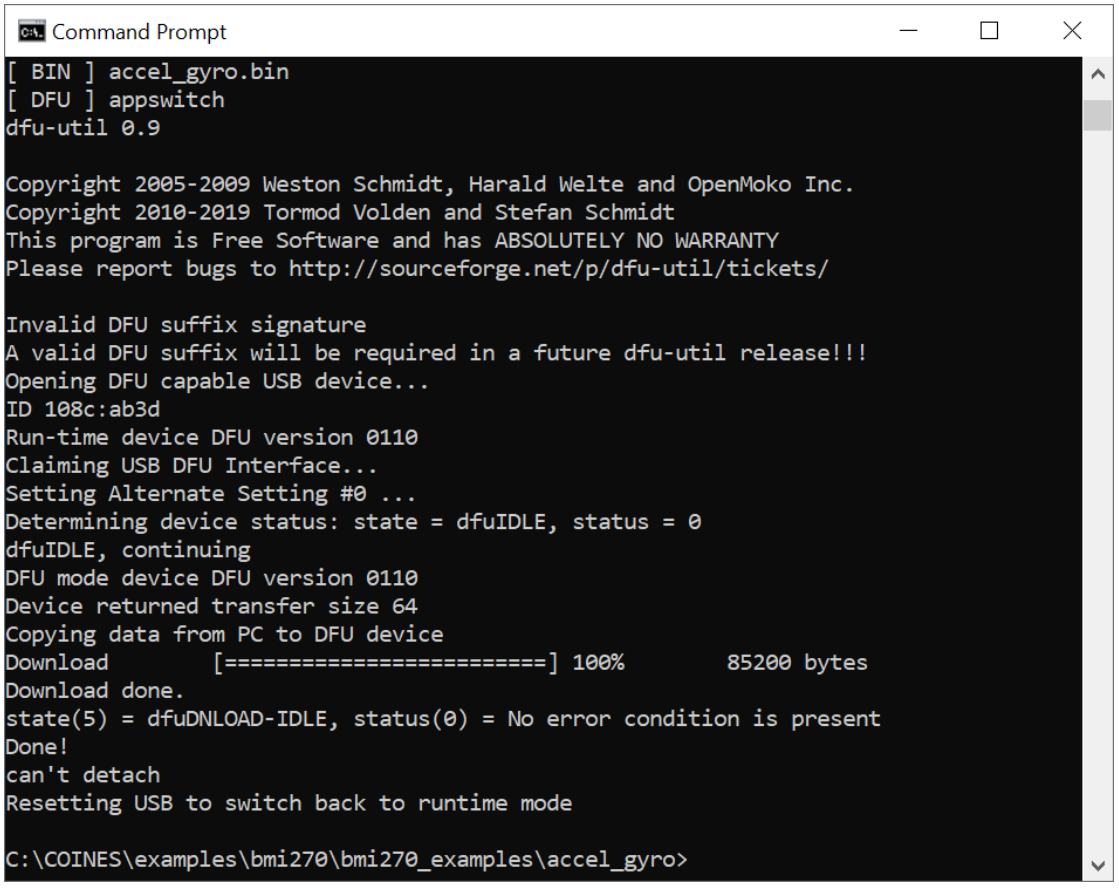
- Selected Platform: Windows
- Board: APP3.1
- Sensor shuttle: BMI270
- Example: `\examples\bmi270\bmi270_examples\accel_gyro`

1. Connect the Application Board via USB, with the sensor shuttle board mounted.
2. Open the command prompt or the terminal.

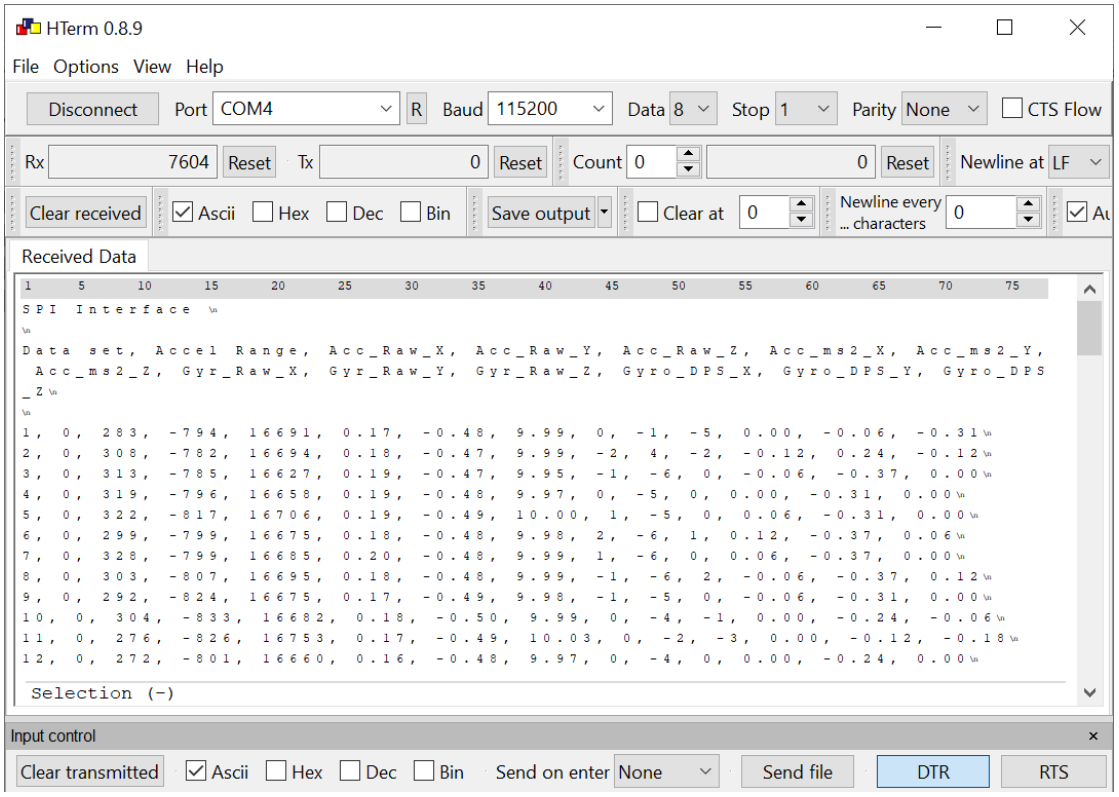
**3.** Use the command `cd` to go to the directory where the example that is to be built is located.

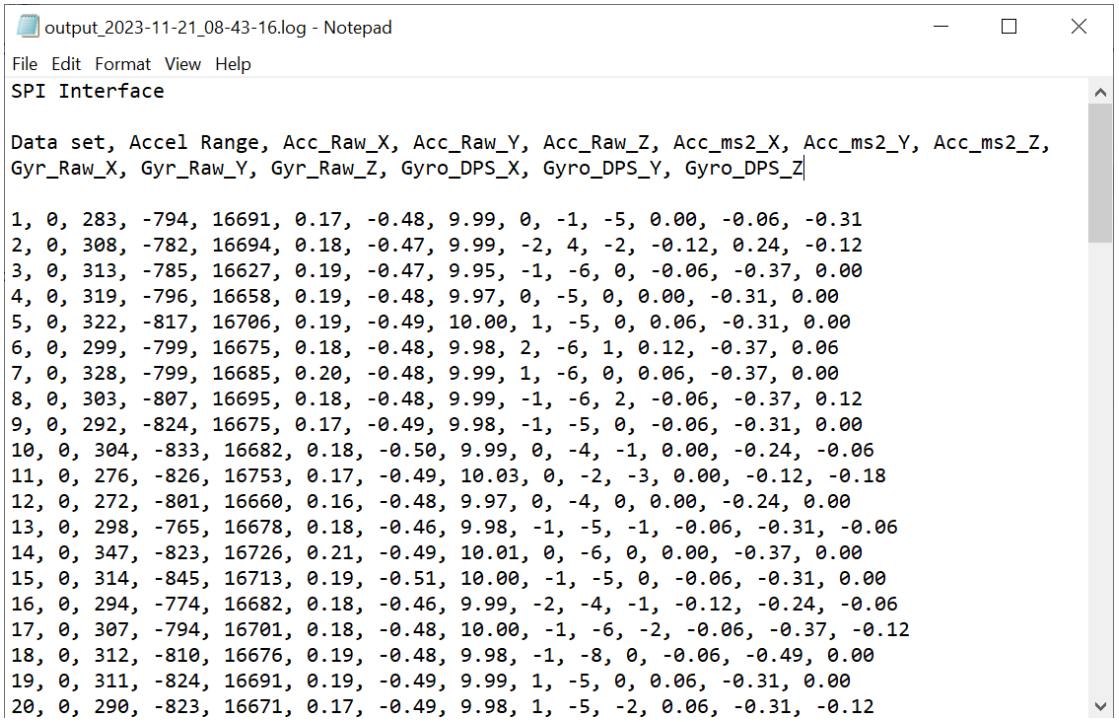



**4.** Ensure the LEDs on the Application Board are turned on for proper connection and then execute the command "mingw32-make TARGET=MCU_APP31 download".

**5.** View the output in a serial terminal application like HTerm.




## 7.4 Running example on MCU side via BLE

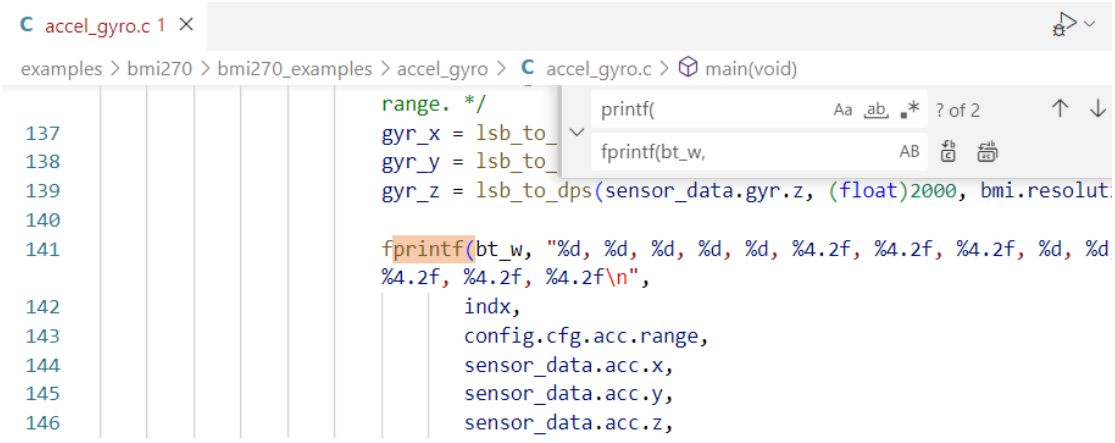
To interface via BLE, follow the steps below:

**1.** Go to the `examples` folder in file explorer.

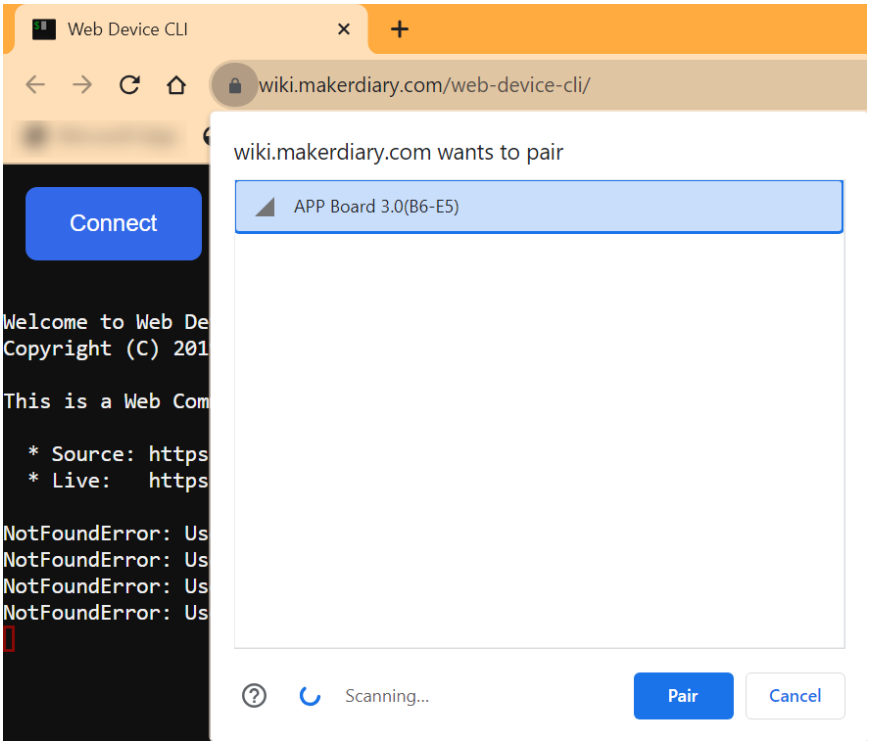
2. Open the common.c file in the selected example folder in your IDE.
3. Change COINES_COMM_INTF_USB to COINES_COMM_INTF_BLE

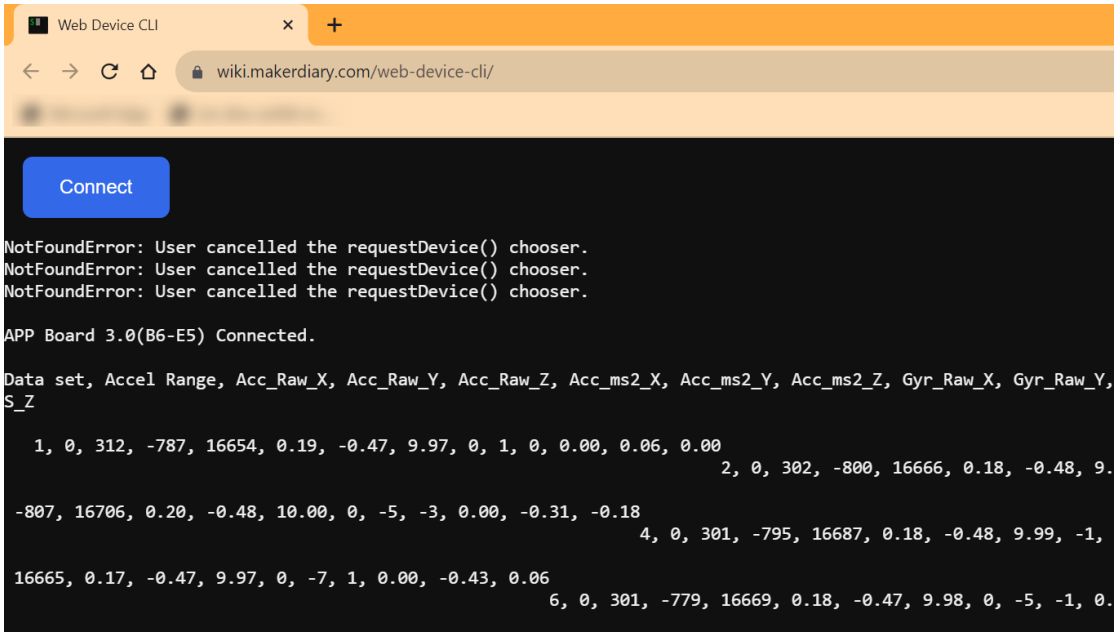

4. Open the example script and change the

   `printf(...) to fprintf(bt_w,...)`




5. Now follow the same steps from 1 - 4 in the 7.3.
6. Connect the Application Board to another power source and keep it within the BLE range.
7. View the output in the Web Device CLI site in your browser by connecting to board via BLE and by clicking **Pair**.



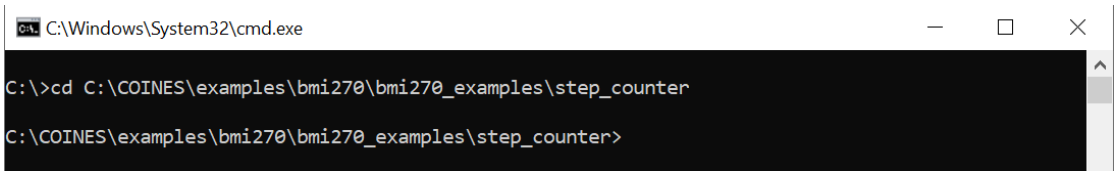

## 7.5 Running example on the PC side

Here are the requirements and instructions to run examples on PC side:

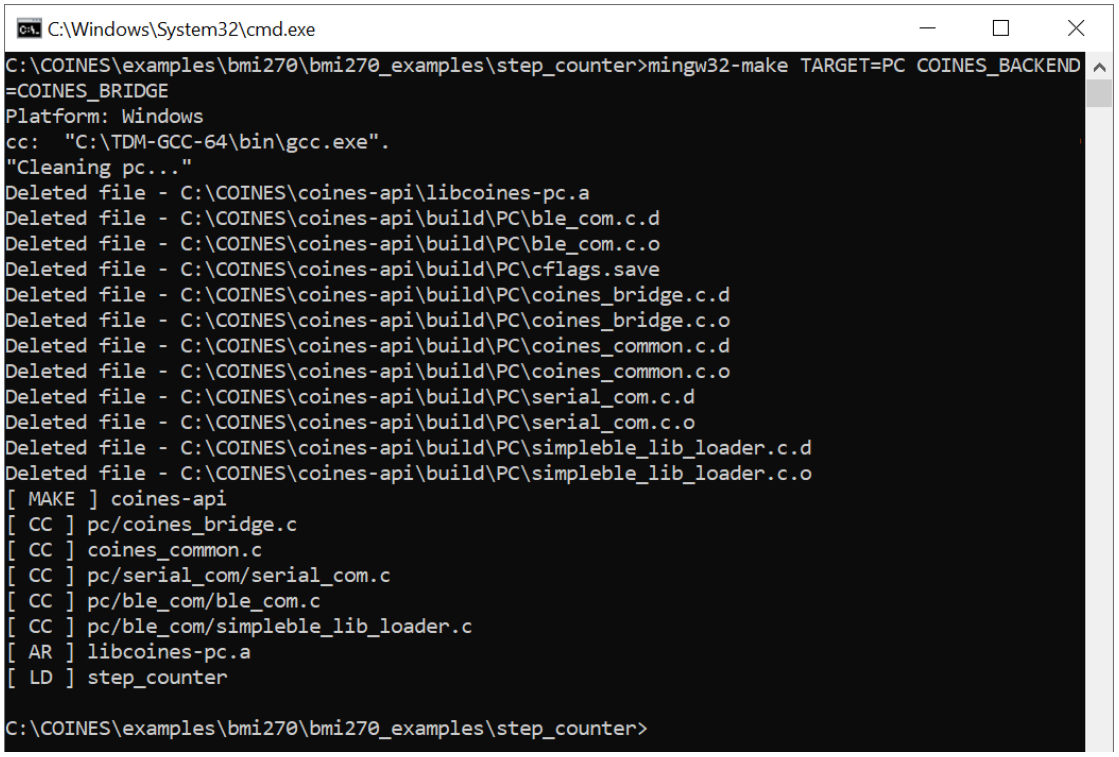
- Selected Platform: Windows
- Board: APP3.1
- Sensor shuttle: BMI270
- Example: `\examples\bmi270\bmi270_examples\step_counter`

1. Connect the Application Board via USB, with the sensor shuttle board mounted.

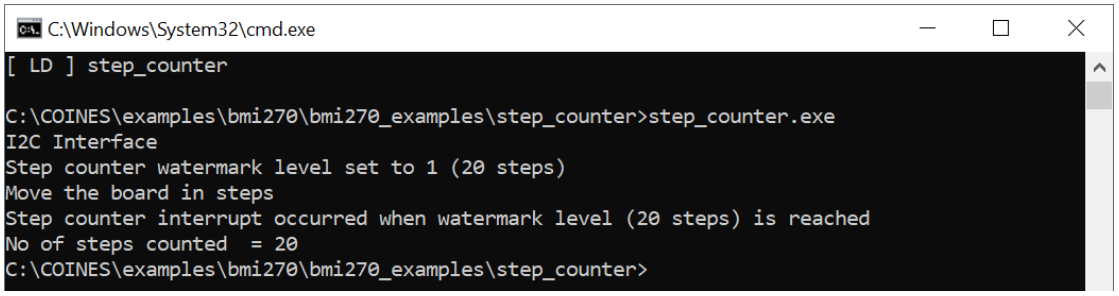
2. Refer to section 14 and update the COINES Bridge firmware to the board.
3. Open the command prompt or the terminal.
4. Use the command `cd` to go to the directory where the example to be built is located.




5. Execute the command "mingw32-make TARGET=PC COINES_BACKEND=COINES_BRIDGE"




6. View the output in the command prompt by running the example executable.

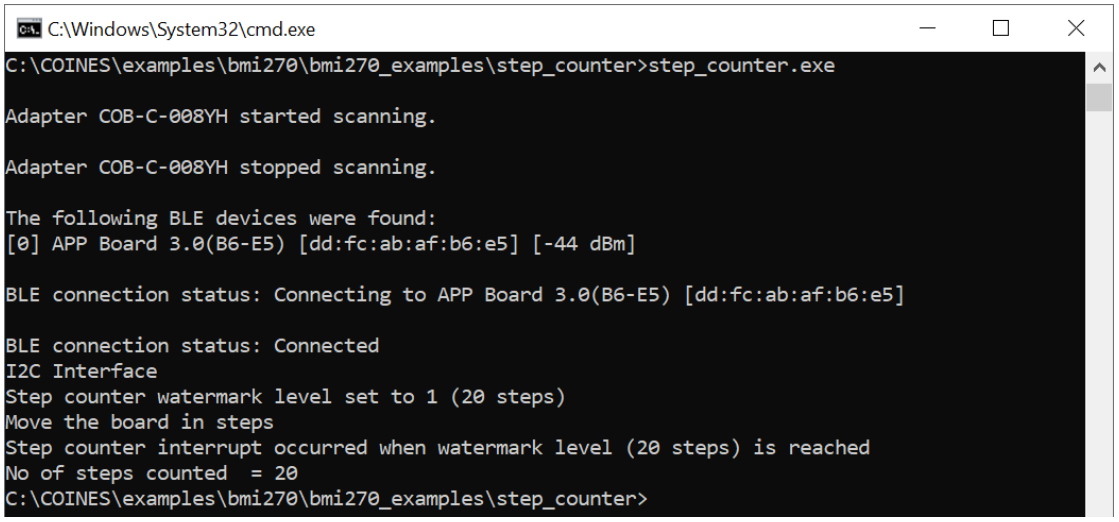



### 7.6 Running example on the PC side via BLE

To interface via BLE, follow the steps below:

1. Go to the examples folder in file explorer.

2. Open the common.c file in the selected example folder in your IDE.
3. Change COINES_COMM_INTF_USB to COINES_COMM_INTF_BLE.
4. Connect the Application Board to another power source and keep it within the BLE range.
5. Follow the steps 3 - 6 in the section 7.5.



# 8 Examples on how to use COINES SDK

## 8.1 C examples

### 8.1.1 Establishing communication

The following code snippet shows how to set up a connection with the board.

```
#include <stdio.h>
#include <stdlib.h>

#include "coines.h"

int main(void)
{
   int8_t error_code;
   enum coines_comm_intf comm_intf = COINES_COMM_INTF_USB;

   error_code = coines_open_comm_intf(comm_intf, NULL);
   if (error_code == COINES_SUCCESS)
   {
      printf("\nSuccessfully connected to board!\n");
   }
   else
   {
      printf("\nUnable to connect with board!\n");
      exit(error_code);
   }

   coines_close_comm_intf(comm_intf, NULL);

   return 0;
}
```

The APIs below can be used for the board interface.

- coines_open_comm_intf
- coines_close_comm_intf

### 8.1.2 Getting board info

The following code snippet shows how to get board information.

```
#include <stdio.h>
#include <stdlib.h>

#include "coines.h"

int main(void)
{
    int8_t error_code;
    struct coines_board_info board_info;
    enum coines_comm_intf comm_intf = COINES_COMM_INTF_USB;

    error_code = coines_open_comm_intf(comm_intf, NULL);
    if (error_code < COINES_SUCCESS)
    {
        printf("\nUnable to connect with board!\n");
```

```
        exit(error_code);
    }

    error_code = coines_get_board_info(&board_info);
    if (error_code == COINES_SUCCESS)
    {
        printf("Board Info:\n");
        printf("Software - v%d.%d.%d\n", (board_info.software_id >> 12) & 0xF,
            (board_info.software_id >> 6) & 0x3F, board_info.software_id & 0x3F);
        printf("Type of board - 0x%x\n", board_info.board );
        printf("Shuttle ID - 0x%x\n", board_info.shuttle_id);
    }

    coines_close_comm_intf(comm_intf, NULL);

    return 0;
}
```

### 8.1.3 I2C config and read

This basic program shows how to configure and perform I2C read.
Sensor: BMI270

```
#include <stdio.h>
#include <stdlib.h>

#include "coines.h"

#define BMI2_I2C_PRIM_ADDR 0x68

int main(void)
{
   int8_t error_code;
   uint8_t chip_id;
   uint8_t reg_addr = 0x0;
   enum coines_comm_intf comm_intf = COINES_COMM_INTF_USB;

   error_code = coines_open_comm_intf(comm_intf, NULL);
   if (error_code == COINES_SUCCESS)
   {
      printf("\nSuccessfully connected to board!\n");
   }
   else
   {
      printf("\nUnable to connect with board!\n");
      exit(error_code);
   }

   /* Power up the board */
   (void)coines_set_shuttleboard_vdd_vddio_config(3300, 3300);
   coines_delay_usec(200);

   /* Make CSB pin HIGH */
   coines_set_pin_config(COINES_SHUTTLE_PIN_21, COINES_PIN_DIRECTION_OUT,
       COINES_PIN_VALUE_HIGH);
   coines_delay_msec(100);

   /* SDO pin is made low */
```

```
    coines_set_pin_config(COINES_SHUTTLE_PIN_SDO, COINES_PIN_DIRECTION_OUT,
        COINES_PIN_VALUE_LOW);

    /* I2C config */
    coines_config_i2c_bus(COINES_I2C_BUS_0, COINES_I2C_STANDARD_MODE);

    /* I2C read */
    (void)coines_read_i2c(COINES_I2C_BUS_0, BMI2_I2C_PRIM_ADDR, reg_addr, &chip_id, 1);

    printf("I2C read: Sensor chip ID - 0x%x\n", chip_id);

    (void)coines_set_shuttleboard_vdd_vddio_config(0, 0);
    coines_delay_msec(100);

    /* Coines interface reset */
    coines_soft_reset();
    coines_delay_msec(100);

    coines_close_comm_intf(comm_intf, NULL);

    return 0;
}
```

The user must pass GPIO pin numbers, the read register address, and the I2C device address for sensors based on the selected sensor shuttle board. I2C communication requires the proper setting of VDD and VDDIO using `coines_set_shuttleboard_vdd_vddio_config`. The APIs below can be used for I2C configure/read/write.

- `coines_config_i2c_bus`
- `coines_read_i2c`
- `coines_write_i2c`

### 8.1.4 SPI config and read

This basic program shows how to configure and perform SPI read.
Sensor: BMI270

```
#include <stdio.h>
#include <stdlib.h>

#include "coines.h"

#define BMI2_SPI_RD_MASK 0x80

int main(void)
{
    int8_t error_code;
    uint8_t chip_id[2], dummy_byte;

    /* An extra dummy byte is read during SPI read */
    uint8_t dummy_byte_len = 1;
    uint8_t reg_addr = 0x0;
    enum coines_comm_intf comm_intf = COINES_COMM_INTF_USB;

    error_code = coines_open_comm_intf(comm_intf, NULL);
    if (error_code == COINES_SUCCESS)
    {
```

```
      printf("\nSuccessfully connected to board!\n");
   }
   else
   {
      printf("\nUnable to connect with board!\n");
      exit(error_code);
   }

   /* Power up the board */
   coines_set_shuttleboard_vdd_vddio_config(3300, 3300);
   coines_delay_msec(200);

   /* SPI config */
   (void)coines_config_spi_bus(COINES_SPI_BUS_0, COINES_SPI_SPEED_5_MHZ,
       COINES_SPI_MODE3);

   /* Pin config */
   coines_set_pin_config(COINES_SHUTTLE_PIN_21, COINES_PIN_DIRECTION_OUT,
       COINES_PIN_VALUE_HIGH);

   /* Mask read register address for SPI */
   reg_addr = (reg_addr | BMI2_SPI_RD_MASK);

   /* Dummy read for SPI init*/
   (void)coines_read_spi(COINES_SPI_BUS_0, COINES_MINI_SHUTTLE_PIN_2_1, reg_addr,
       &dummy_byte, 1);
   coines_delay_usec(450);

   /* SPI read */
   (void)coines_read_spi(COINES_SPI_BUS_0, COINES_MINI_SHUTTLE_PIN_2_1, reg_addr,
       chip_id, 1 + dummy_byte_len);
   coines_delay_usec(450);

   printf("SPI read: Sensor chip ID - 0x%x\n", chip_id[dummy_byte_len]);

   (void)coines_set_shuttleboard_vdd_vddio_config(0, 0);
   coines_delay_msec(100);

   /* Coines interface reset */
   coines_soft_reset();
   coines_delay_msec(100);

   coines_close_comm_intf(comm_intf, NULL);

   return 0;
}
```

The user must pass GPIO pin numbers, the read register, and the SPI CS pins for sensors based on the selected sensor shuttle board. SPI communication requires the proper setting of VDD and VDDIO using `coines_set_shuttleboard_vdd_vddio_config`. The APIs below can be used for SPI configure/read/write.

- `coines_config_spi_bus`
- `coines_read_spi`
- `coines_write_spi`

### 8.1.5 Configure VDD and VDDIO

This program shows how to set custom VDD and VDDIO voltages.
Sensor: BMI270
Target board: APP31

```
#include <stdio.h>
#include <stdbool.h>

#include "coines.h"

#define BMI2_I2C_PRIM_ADDR 0x68

int main(void)
{
   uint8_t chip_id;
   uint16_t no_of_bytes = 1;
   uint8_t reg_addr = 0x0;
   uint16_t vdd_millivolt = 1800;
   uint16_t vddio_millivolt = 1800;

   coines_open_comm_intf(COINES_COMM_INTF_USB, NULL);

   /* Customized VDD and VDDIO - 1.8V */
   printf("Configuring VDD: %.1fV\n", vdd_millivolt / 1000.00);
   printf("Configuring VDDIO: %.1fV\n", vddio_millivolt / 1000.00);
   coines_set_shuttleboard_vdd_vddio_config(vdd_millivolt, vddio_millivolt);
   coines_delay_msec(200);

   /* I2C config & I2C read */
   coines_config_i2c_bus(COINES_I2C_BUS_0, COINES_I2C_STANDARD_MODE);
   (void)coines_read_i2c(COINES_I2C_BUS_0, BMI2_I2C_PRIM_ADDR, reg_addr, &chip_id,
       no_of_bytes);
   printf("I2C read: Sensor chip ID - 0x%x\n", chip_id);

   /* Device reset - 0V */
   (void)coines_set_shuttleboard_vdd_vddio_config(0, 0);
   coines_delay_msec(100);

   coines_close_comm_intf(COINES_COMM_INTF_USB, NULL);

   return 0;
}
```

In this example, the voltage levels for VDD and VDDIO are configured according to the typical values specified in the BMI270 datasheet. Within the specified operating range, users can customize the voltage levels applied to the VDD and VDDIO pins according to their requirements. APP3.0 supports only two voltage levels for VDD, which are 1.8V and 2.8V, and a single voltage level of 1.8V for VDDIO. However, in APP3.1, it is possible to configure custom voltage levels for both VDD and VDDIO.

### 8.1.6 Led and button control

The example program below controls LEDs and buttons on the board.
Target: MCU
Target board: APP31

```
#include <stdio.h>
#include <stdbool.h>
#include "coines.h"

/* Callback for button 3 interrupt */
static void button3CB(uint32_t param1, uint32_t param2);

/*Callback for button 3 event */
void button3CB(uint32_t param1, uint32_t param2)
{
   (void)param1;
   (void)param2;

   coines_set_led(COINES_LED_RED, COINES_LED_STATE_ON);
   coines_set_led(COINES_LED_GREEN, COINES_LED_STATE_OFF);
   coines_set_led(COINES_LED_BLUE, COINES_LED_STATE_ON);
}

int main(void)
{
   coines_open_comm_intf(COINES_COMM_INTF_USB, NULL);

   coines_set_pin_config(COINES_APP31_BUTTON_3, COINES_PIN_DIRECTION_IN,
       COINES_PIN_VALUE_HIGH);
   coines_attach_interrupt(COINES_APP31_BUTTON_3, button3CB,
       COINES_PIN_INTERRUPT_FALLING_EDGE);

   coines_close_comm_intf(COINES_COMM_INTF_USB, NULL);

   return 0;
}
```

### 8.1.7 File listing in External memory

The snippet below can be used to list the files in the external memory.
Target: MCU

```
#include <stdio.h>
#include "coines.h"

int main(void)
{
   coines_open_comm_intf(COINES_COMM_INTF_USB, NULL);
   DIR *directory;
   struct dirent *dir;
   directory = opendir(".");
   if (directory)
   {
      while ((dir = readdir(directory)) != NULL)
      {
         printf("%s\n", dir->d_name);
      }

      closedir(directory);
   }

   coines_close_comm_intf(COINES_COMM_INTF_USB, NULL);
```

```
    return 0;
}
```

### 8.1.8 Temperature measurement

This program demonstrates how to measure the temperature of the board.
Target: MCU

```
#include <stdio.h>
#include <stdlib.h>

#include "coines.h"

int main(void)
{
    int8_t error_code;
    float temp_data = 0;
    enum coines_comm_intf comm_intf = COINES_COMM_INTF_USB;

    error_code = coines_open_comm_intf(comm_intf, NULL);
    if (error_code == COINES_SUCCESS)
    {
        printf("\nSuccessfully connected to board!\n");
    }
    else
    {
        printf("\nUnable to connect with board!\n");
        exit(error_code);
    }

    /* Power up the board */
    coines_set_shuttleboard_vdd_vddio_config(1800, 1800);
    coines_delay_msec(200);

    /* Read temperature data */
    coines_read_temp_data(&temp_data);
    printf("\nTemperature data = %f in degC", temp_data);

    coines_set_shuttleboard_vdd_vddio_config(0, 0);
    coines_delay_msec(100);

    coines_close_comm_intf(comm_intf, NULL);

    return 0;
}
```

### 8.1.9 Battery level measurement

This program demonstrates how to measure battery level when a battery is connected to the board.
Target: MCU

```
#include <stdio.h>
#include <stdlib.h>

#include "coines.h"
```

```
int main(void)
{
   int8_t error_code;
   uint8_t batt_status_percentage = 0;
   uint16_t batt_status_in_milli_volts = 0;
   enum coines_comm_intf comm_intf = COINES_COMM_INTF_BLE;

   error_code = coines_open_comm_intf(comm_intf, NULL);
   if (error_code == COINES_SUCCESS)
   {
      printf("\nSuccessfully connected to board!\n");
   }
   else
   {
      printf("\nUnable to connect with board!\n");
      exit(error_code);
   }

   /* Read battery level */
   coines_read_bat_status(&batt_status_in_milli_volts, &batt_status_percentage);
   fprintf(bt_w, "Battery level in percentage = %d %% \r\n", batt_status_percentage);
   fprintf(bt_w, "Battery level in millivolts = %d mV \r\n",
       batt_status_in_milli_volts);

   coines_close_comm_intf(comm_intf, NULL);

   return 0;
}
```

### 8.1.10 Configure BLE communication

This example shows how to configure the BLE connection.
Target: PC

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

#include "coines.h"

/*! Macros to hold the BLE peripheral name and address to be connected */
/*! Please change the name and address with BLE name of the Application Board under
    test */
#define BLE_NAME "APP Board 3.1(B6-E5)"
#define BLE_ADDR "dd:fc:ab:af:b6:e5"

/*! Variable to hold the communication interface type */
const enum coines_comm_intf comm_intf = COINES_COMM_INTF_BLE;

int main(void)
{
   struct ble_peripheral_info ble_config = { BLE_ADDR, "" }; /* Connection using BLE
       address */
   /*struct ble_peripheral_info ble_config = { "", BLE_NAME }; // Connection using BLE
       name */
   struct ble_peripheral_info ble_info[40];
   uint8_t peripheral_count, i;
```

```
    int8_t result;

    /* Get the BLE peripheral list */
    result = coines_scan_ble_devices(ble_info, &peripheral_count, 7000);
    if (result != COINES_SUCCESS)
    {
        const char *err_str = get_coines_error_str(result);
        printf("\n%s", err_str);
        exit(result);
    }

    /* Print the BLE peripheral list */
    printf("\nBLE devices found:");
    for (i = 0; i < peripheral_count; i++)
    {
        printf("\n[%d] %s [%s]", i, ble_info[i].ble_identifier, ble_info[i].ble_address);
    }

    /* Open BLE connection */
    result = coines_open_comm_intf(comm_intf, &ble_config);
    if (result == COINES_SUCCESS)
    {
        printf("\nSuccessfully connected to board!\n");
    }
    else
    {
        printf("\nUnable to connect with board!\n");
        exit(result);
    }

    /* Close BLE connection */
    coines_set_shuttleboard_vdd_vddio_config(0, 0);
    coines_delay_msec(100);

    coines_soft_reset();
    coines_delay_msec(100);

    coines_close_comm_intf(comm_intf, NULL);

    return 0;
}
```

The user must modify BLE settings like MAC address and APP3.X BLE name before executing this example.

### 8.1.11 Configure Serial communication

This example shows how to configure the Serial COM connection.
Target: PC
Target board: APP31

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

#include "coines.h"

#define ROBERT_BOSCH_USB_VID (0x108C)
```

```
#define BST_APP31_CDC_USB_PID (0xAB38)

/*! Variable to hold the communication interface type */
const enum coines_comm_intf comm_intf = COINES_COMM_INTF_USB;

int main(void)
{
   int16_t result;
   struct coines_serial_com_config scom_config;

   scom_config.baud_rate = 38400;
   scom_config.vendor_id = ROBERT_BOSCH_USB_VID;
   scom_config.product_id = BST_APP31_CDC_USB_PID;
   scom_config.com_port_name = "COM4";

   /* Serial com for Linux */
   /*scom_config.com_port_name = "/dev/ttyACM0"; */

   /* Serial com for Mac */
   /*scom_config.com_port_name = "/dev/cu.usbmodemD0F684FC766C1"; */

   scom_config.rx_buffer_size = 2048;

   /* Open serial connection */
   result = coines_open_comm_intf(comm_intf, &scom_config);
   if (result == COINES_SUCCESS)
   {
      printf("\nSuccessfully connected to board!\n");
   }
   else
   {
      printf("\nUnable to connect with board!\n");
      exit(result);
   }

   /* Close serial connection */
   coines_soft_reset();
   coines_delay_msec(100);

   coines_close_comm_intf(comm_intf, NULL);

   return 0;
}
```

The user must modify Serial COM settings like vendor ID, product ID, and COM port name before executing this example.

## 8.2 Python examples

### 8.2.1 Getting board info

The following code snippet shows how to get board information.

```
import coinespy as cpy
from coinespy import ErrorCodes

COM_INTF = cpy.CommInterface.USB
```

```
if __name__ == "__main__":
    board = cpy.CoinesBoard()
    print('COINESPY version - %s' % cpy.__version__)
    board.open_comm_interface(COM_INTF)
    if board.error_code != ErrorCodes.COINES_SUCCESS:
        print(f'Could not connect to board: {board.error_code}')
    else:
        b_info = board.get_board_info()
        print(f"COINES SDK version: {board.lib_version}")
        print('BoardInfo')
          print(f'Software ID: v{(b_info.SoftwareId >> 12) & 0xF}.{(b_info.SoftwareId >>
              6) & 0x3F}.{b_info.SoftwareId & 0x3F}')
          print(f'Type of Board: {hex(b_info.Board)}')
        board.close_comm_interface()
```

### 8.2.2 I2C config and read

This program shows how to configure and perform I2C read.
Sensor: BMI270

```
import sys
import coinespy as cpy
from coinespy import ErrorCodes

COM_INTF = cpy.CommInterface.USB

if __name__ == "__main__":
    BMI2_I2C_PRIM_ADDR = 0x68
    BMI2_CHIP_ID_ADDR = 0x00
    BOARD = cpy.CoinesBoard()

    BOARD.open_comm_interface(COM_INTF)
    if BOARD.error_code == ErrorCodes.COINES_SUCCESS:
        print("Successfully connected to board!")
    else:
        print(f"Open Communication interface: {BOARD.error_code}")
        sys.exit()

    # Power up the board
    BOARD.set_shuttleboard_vdd_vddio_config(vdd_val=3.3, vddio_val=3.3)
    BOARD.delay_milli_sec(200)

    # Pin config
    # Make CSB pin HIGH
    BOARD.set_pin_config(
        cpy.MultiIOPin.SHUTTLE_PIN_21, cpy.PinDirection.OUTPUT, cpy.PinValue.HIGH
    )
    # SDO pin is made low
    BOARD.set_pin_config(
        cpy.MultiIOPin.SHUTTLE_PIN_SDO, cpy.PinDirection.OUTPUT, cpy.PinValue.LOW
    )

    # I2C config
    BOARD.config_i2c_bus(
        cpy.I2CBus.BUS_I2C_0, BMI2_I2C_PRIM_ADDR, cpy.I2CMode.STANDARD_MODE
    )

    # I2C read
```

```
   accel_chip_id = BOARD.read_i2c(
      cpy.I2CBus.BUS_I2C_0, BMI2_CHIP_ID_ADDR, 1, BMI2_I2C_PRIM_ADDR
   )

   print(f"I2C read: Sensor chip ID - {hex(accel_chip_id[0])}")

   # Deinit board
   BOARD.set_shuttleboard_vdd_vddio_config(vdd_val=0, vddio_val=0)
   BOARD.soft_reset()

   BOARD.close_comm_interface()
```

The user must pass GPIO pin numbers, the read register address, and the I2C device address for sensors based on the selected sensor shuttle board. I2C communication requires the proper setting of VDD and VDDIO using `set_shuttleboard_vdd_vddio_config`.

### 8.2.3 SPI config and read

This program shows how to configure and perform SPI read.
Sensor: BMI270

```
import sys
import coinespy as cpy
from coinespy import ErrorCodes

COM_INTF = cpy.CommInterface.USB

if __name__ == "__main__":
   BMI2_CHIP_ID_ADDR = 0x00
   DUMMY_BYTES = 1
   BOARD = cpy.CoinesBoard()

   BOARD.open_comm_interface(COM_INTF)
   if BOARD.error_code == ErrorCodes.COINES_SUCCESS:
      print("Successfully connected to board!")
   else:
      print(f"Open Communication interface: {BOARD.error_code}")
      sys.exit()

   # Power up the board
   BOARD.set_shuttleboard_vdd_vddio_config(vdd_val=3.3, vddio_val=3.3)
   BOARD.delay_milli_sec(200)

   #  SPI config
   BOARD.config_spi_bus(
      cpy.SPIBus.BUS_SPI_0,
      cpy.MultiIOPin.MINI_SHUTTLE_PIN_2_1,
      cpy.SPISpeed.SPI_5_MHZ,
      cpy.SPIMode.MODE3,
   )

   # Pin config
   BOARD.set_pin_config(
      cpy.MultiIOPin.SHUTTLE_PIN_21, cpy.PinDirection.OUTPUT, cpy.PinValue.HIGH
   )

   # Initialize SPI by dummy read
   BOARD.read_spi(cpy.SPIBus.BUS_SPI_0, BMI2_CHIP_ID_ADDR, 1)
```

```
BOARD.delay_micro_sec(450)

# SPI read
chip_id = BOARD.read_spi(
   cpy.SPIBus.BUS_SPI_0,
   BMI2_CHIP_ID_ADDR,
   1 + DUMMY_BYTES,
   cpy.MultiIOPin.MINI_SHUTTLE_PIN_2_1,
)
BOARD.delay_micro_sec(450)

print(f"SPI read: Sensor chip ID - {hex(chip_id[1])}")

# Deinit board
BOARD.set_shuttleboard_vdd_vddio_config(vdd_val=0, vddio_val=0)
BOARD.soft_reset()

BOARD.close_comm_interface()
```

The user must pass GPIO pin numbers, the read register address, and the SPI CS pins for sensors based on the selected sensor shuttle board. SPI communication requires the proper setting of VDD and VDDIO using `set_shuttleboard_vdd_vddio_config`.

## 9 FreeRTOS support

FreeRTOS support is available for the MCU_APP3X targets. For a sample FreeRTOS application, refer to: `examples\c\freeRTOS\Sample\main.c`.

## 10 Debugging via VS code

Follow these steps to debug programs via VS code:

- Download Segger software from https://www.segger.com/downloads/jlink/.
- Refer to https://wiki.segger.com/J-Link_Visual_Studio_Code for using J-link with VS code.
- Download the NRF5 .svd file from Nordic Semiconductor GitHub.
- Connect J-link to the SWD debugger connector.

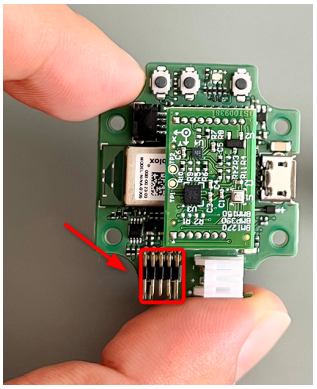


Fig. 12: APP3.1 Debugger connector

- Below is the sample launch.json config for VS code debug.

launch.json

.vscode > launch.json > ...

```
{
    // Use IntelliSense to learn about possible attributes.
    // Hover to view descriptions of existing attributes.
    // For more information, visit: https://go.microsoft.com/fwlink/?linkid=830387
    "version": "0.2.0",
    "configurations": [
        {
            "type": "cortex-debug",
            "request": "launch",
            "name": "Debug J-Link",
            "cwd": "${workspaceRoot}",
            "executable": "[redacted]/examples/c/coines_bridge_firmware/coines_bridge_firmware.elf",
            "serverpath": "C:/Program Files/SEGGER/JLink/JLinkGDBServerCL.exe",
            "servertype": "jlink",
            "armToolchainPath":"C:/Program Files (x86)/Arm GNU Toolchain arm-none-eabi/12.2 mpacbti-rel1/bin",
            "device": "nRF52840_xxAA",
            "interface": "swd",
            "showDevDebugOutput": "raw",
            "svdFile": "[redacted]/nRF52840.svd"

        }
    ]
}
```

Fig. 13: VS code debug launch.json

## 11 Media Transfer Protocol (MTP) firmware for APP3.X

The external memory chip W25M02/W25N02 on APP3.X is based on NAND flash.

FAT filesystem on NAND flash memory results in a complicated solution using a lot of RAM. Moreover, using FAT without a Flash Translation Layer (to save RAM) wears out NAND flash with frequent usage. Therefore, FlogFS, a filesystem optimized for use with NAND flash, was chosen.

Using 'FlogFS' presents a new problem: 'Filesystem access from the PC via USB'. The use of 'FlogFS' with the USB Mass Storage protocol is impossible because the operating system can't recognize 'FlogFS' as a valid file system.

Using a custom protocol for filesystem operations would mean re-inventing the wheel and a lot of effort. Users also would have a different experience than with USB Mass Storage.

The solution is to use the "Media Transfer Protocol" developed initially by Microsoft for portable devices like MP3 players. Starting from Android Kitkat (v4.4), MTP is the only way to access files on an Android device since the whole flash memory (including user storage space) uses filesystems like ext4, YAFFS, F2FS, etc.

Files in APP3.X's NAND flash memory can be viewed using the USB MTP firmware.

Supported on Windows, Linux, macOS, and Android (via USB OTG).

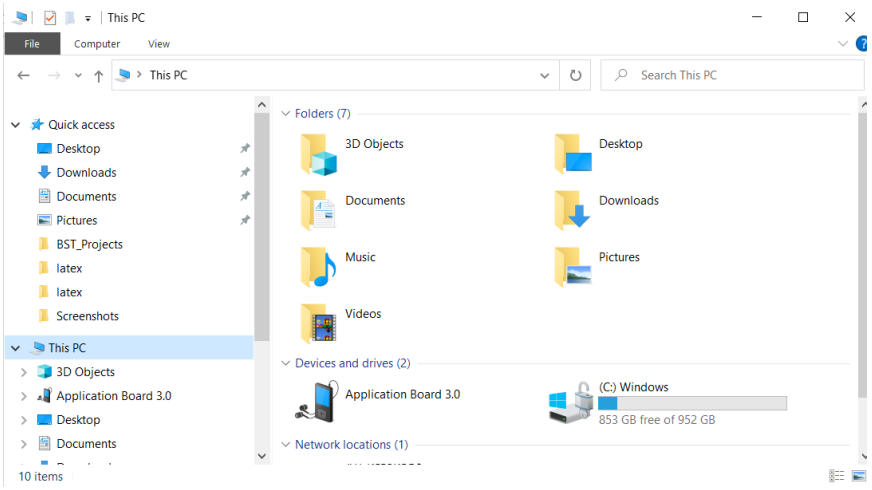



Fig. 14: APP3.X in MTP mode on Windows

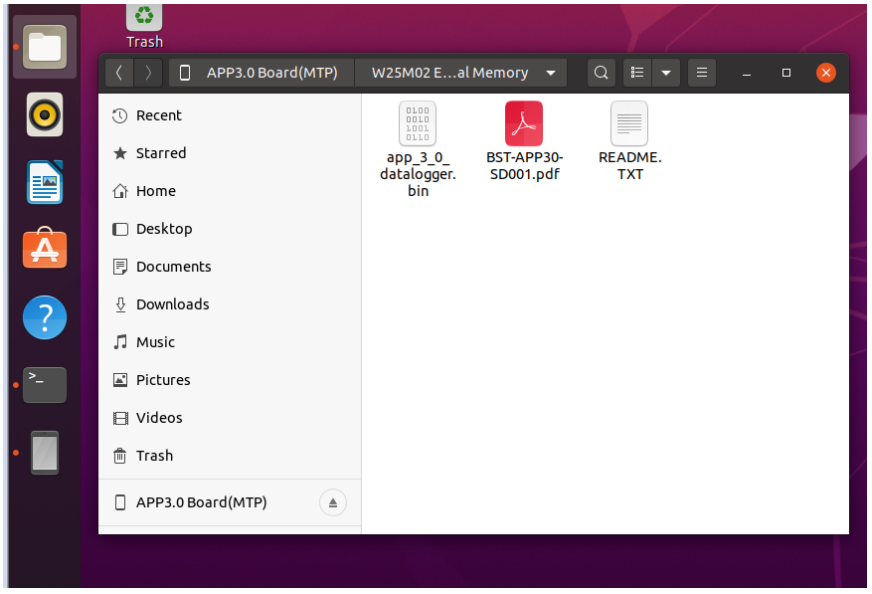



Fig. 15: APP3.X in MTP mode on Linux

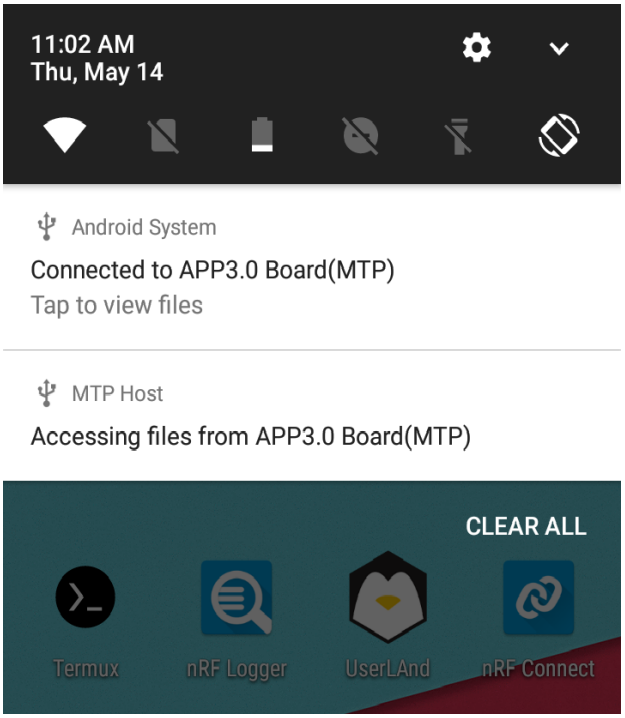



Fig. 16: APP3.X in MTP mode on Andriod

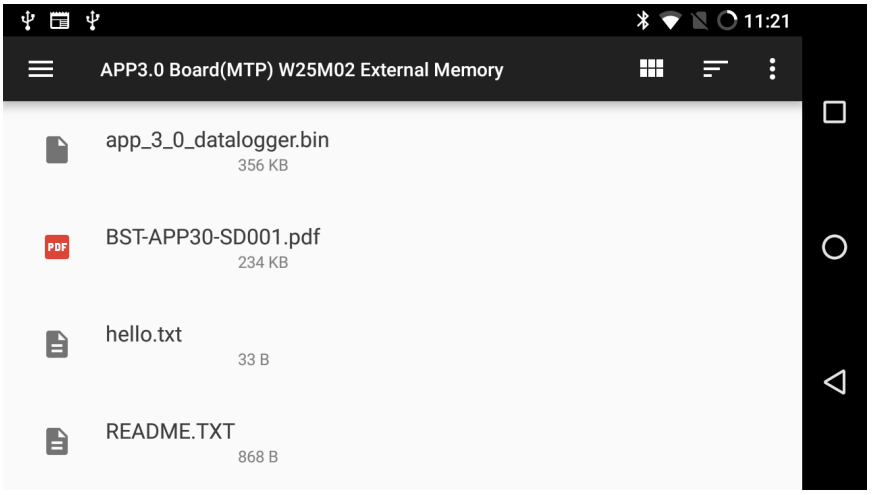



Fig. 17: Files in external memory listed on Andriod device

## 11.1 Copying the files using MTP

Procedure needs the following steps:

- APP3.X comes with the preloaded MTP firmware update package.
- Refer to section Switching to Operating Modes to switch to MTP mode
- The device will enumerate as an MTP device with the name "Application Board 3.X". Click it and select the "W25M02 External Memory". The device will list all the available files and all required files can be copied.

*Note*: Before macOS 12 (Monterey), limited MTP access was possible via third-party tools, though never natively supported. From macOS 12 onward, even that support was removed. To detect MTP devices and transfer files, use mtp-tools executables, which are installed alongside the SDK.

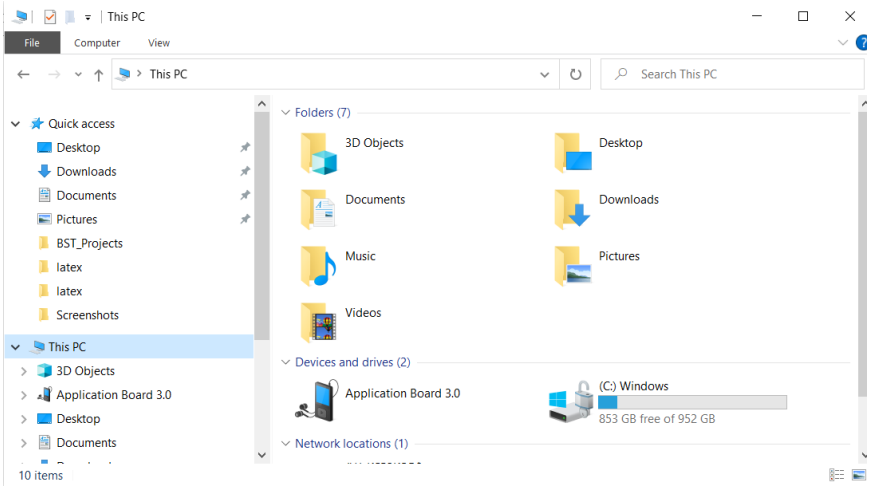



Fig. 18: Copy data log files to the PC over USB MTP

## 11.2 Formatting an MTP Device

**On Windows:** To format an MTP device, right-click it in File Explorer and select the *Format* option. This will erase all files stored in the external memory.

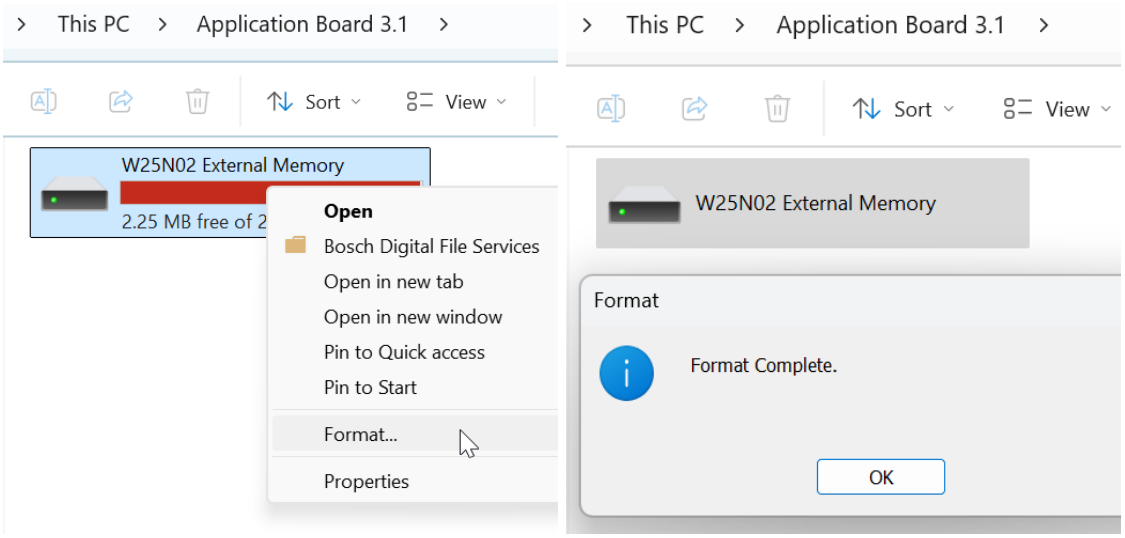



Fig. 19: Steps to format an MTP device in Windows

**On Linux and macOS:** Use the 'mtp-format' command from the `mtp-tools` package to format the MTP device. This utility is typically installed as part of the SDK setup.

### Troubleshooting on Linux

If you encounter an error like the one shown below when attempting to connect to the MTP device, it is likely due to interference from **GVFS**:

```
pkill gvfsd-mtp
```

or

```
pkill gvfs-mtp-volume-monitor
```

Run one of the above commands to stop the interfering process and retry the connection.

```
~$ mtp-format
libmtp version: 1.1.21

Device 0 (VID=108c and PID=ab3a) is UNKNOWN in libmtp v1.1.21.
Please report this VID/PID and the device model to the libmtp development team
libusb_claim_interface() reports device is busy, likely in use by GVFS or KDE MT
P device handling alreadyLIBMTP PANIC: Unable to initialize device
No devices.
~$ pkill gvfsd-mtp
~$ mtp-format
libmtp version: 1.1.21

Device 0 (VID=108c and PID=ab3a) is UNKNOWN in libmtp v1.1.21.
Please report this VID/PID and the device model to the libmtp development team
I will now format your device. This means that
all content (and licenses) will be lost forever.
you will not be able to undo this operation.
Continue? (y/n)
> y
```

Fig. 20: Formatting an MTP device on Linux using mtp-tools

# 12 USB/BLE DFU bootloader

A USB/BLE Bootloader for APP3.X/nRF52840 and a Nicla Sense ME/nRF52832 chip comply with the https://www.usb.org/sites/default/files/DFU_1.1.pdf.

## 12.1 Key Features

### 12.1.1 USB DFU

The key features of USB DFU are as follows:

- Code download to RAM or FLASH
- Code read back (upload) from RAM or FLASH (useful for taking firmware backups)
- Works with Windows, Linux, macOS and Android.

### 12.1.2 BLE DFU

The key features of BLE DFU are as follows:

- Code download to FLASH
- Works with PC and mobile devices with iOS/Android

The bootloader was written taking into account the following aspects:

- Usability
  1. No special driver installation or admin rights should be required.
  2. The update process should be straight forward.
- Maintainability
  1. It's an open source community that takes care of the tools on the PC side. For example, dfu-util is a cross platform tool.
  2. Uses Google Chrome's WebUSB to update firmware. Sample implementation https://devanlai.github.io/webdfu/dfu-util/
- Size
- COINES SDK on MCU

## 12.2 Invoking the Bootloader

To invoke Bootloader from hardware, switch the board to bootloader mode (refer to section 13).

To invoke Bootloader from software, use the snippets below in your program based on the board selected.

- APP3.X
  1. Write 0x4E494F43 ('N','I','O','C') to MAGIC_LOCATION (0x2003FFF4)
  2. Write 0x0 or 0xF0000 to APP_START_ADDR (0x2003FFF8)
  3. Call NVIC_SystemReset()

```
#define MAGIC_LOCATION (0x2003FFF4)
#define APP_START_ADDR (*(uint32_t *)(MAGIC_LOCATION+4)

*((uint32_t *)MAGIC_LOCATION) == 0x4E494F43;
```

```
APP_START_ADDR = 0xF0000;
//APP_START_ADDR = 0x0;
NVIC_SystemReset();
```

- Nicla Sense ME Board
  1. Write 0x544F4F42 ('T','O','O','B') to MAGIC_LOCATION (0x2000F804)
  2. Call NVIC_SystemReset()

```
#define MAGIC_LOCATION (0x2000F804)
#define APP_START_ADDR (*(uint32_t *)(MAGIC_LOCATION+4)

*((uint32_t *)MAGIC_LOCATION) == 0x544F4F42;
NVIC_SystemReset();
```

*Note*: The same feature can also be used to perform application switches (two or more applications can reside in the same flash memory at different address locations). Write the application start address to APP_START_ADDR instead of the bootloader address.

### 12.3 Using the Bootloader via USB

The commands below demonstrate how to use dfu-util for different scenarios:

- Path to dfu-util: `\tools\usb-dfu`

Write firmware to flash memory using the following command:

- dfu-util -a FLASH -D <firmware>.bin -R

Write firmware to RAM memory using the following command:

- dfu-util -a RAM -D <firmware>.bin -R

Read firmware from flash memory using the following command:

- dfu-util -a FLASH -U <firmware>.bin

Read firmware from RAM memory using the following command:

- dfu-util -a RAM -U <firmware>.bin

Read device serial number/ BLE MAC address using the following command:

- dfu-util -l

  *Note*: Not applicable for Nicla Sense ME board

### 12.4 Using the Bootloader via BLE

To update the bootloader firmware via BLE, use the following procedure:

- PC (Windows, Linux or macOS)
  Python script present in following path `\tools\app30-ble-dfu` can use the binary file directly.
  1. Refer to section 13 to switch to Bootloader mode.
  2. Run the command:
     - `pip install -r requirements.txt`
  3. Scan for devices to find BLE MAC address using below the command:
     - `python app30-ble-dfu.py -l`

4. Update firmware by using the MAC address obtained in the previous step and firmware BIN file:
   - `python app30-ble-dfu.py -d D7:A3:CE:8E:36:14 -f <firmware>.bin`

- Android devices

1. Generate the ZIP package using https://pypi.org/project/adafruit-nrfutil/ before using nRF ToolBox for BLE or nRF connect for mobile:
   - `adafruit-nrfutil dfu genpkg -dev-type 0x0052 -application <firmware>.bin dfu-package.zi`

*Note*: Not applicable for Nicla Sense ME board

# 13 Switching to Operating Modes

## 13.1 APP3.X

The process for switching modes for the Application Board involves these steps:

- Bootloader mode - Turn OFF and ON the board with T2 pressed, the blue LED glows indicating that the board switched to bootloader mode.
- MTP mode - Turn OFF and ON the board with T1 pressed, the green LED glows indicating that the board switched to MTP mode.

## 13.2 Nicla Sense ME board

The process for switching modes for the Nicla Sense ME board involves these steps:

- Bootloader mode - Press the reset button three times, the blue LED glows indicating that the board switched to bootloader mode.
- Application Mode - Press the reset button three times to switch to application mode.

# 14 Updating firmware using COINES SDK

To update the firmware, follow these steps:

1. Connect the Engineering Board using USB cable to PC.
2. These boards come preloaded with a bootloader update package.

Run the required .bat/.sh scripts inside `\firmware\` based on the board chosen.

| Function | Script |
|---|---|
| Nicla Sense ME Board - Prepare board | `prepare_nicla.bat` |
| Bootloader update | `\bootloader_update\update_bootloader.bat` |
| MTP firmware update | `\mtp_fw_update\update_mtp_fw.bat` |
| COINES Bridge firmware targetted for RAM memory | `\coines_bridge\update_coines_bridge_ram_fw.bat` |
| COINES Bridge firmware targetted for FLASH memory | `\coines_bridge\update_coines_bridge_flash_fw.bat` |

## 15 FAQs

1. **What to do in case of a communication or initialization failure while running examples?**
   Resetting or rebooting the board will help solve such issues.
2. **Why is there no output in my terminal application after cross-compiling and downloading an example on the MCU?**
   The code example on the MCU waits until the serial port of the board is opened. However, opening the port is not enough, the user has to ensure that also the DTR signal is set (this is required due to higher compatibiliy among different terminal applications).
3. **How to fix libusb not found issue on macOS (arm64)?**

   Try the steps below to fix the issue:

   **a.** Install libusb: Libusb will be automatically installed as part of the COINES SDK installation. However, If it's not installed automatically, you can use Homebrew to install it:
   `brew install libusb`

   After running the above command, libusb should be installed on your system.
   On Intel Mac: `/usr/local/lib`
   On M1 Mac: `/opt/homebrew/lib`

   **b.** Add the path in `<installed_COINES_SDK_location>/coines.mk`

coines.mk

```
135     else
136         DRIVER = LIBUSB_DRIVER
137         ifeq ($(shell uname -m),arm64)
138             LIBPATHS += /opt/homebrew/lib
139         endif
140     endif
```

Fig. 21: COINES SDK file structure

4. **How do I recover the original program when bootloader was erased accidentally on Application Board 3.X?**
   Refer to the section 10 to setup Segger and connect JLink with APP3.X. Follow the steps below to reprogram the APP3.X.
   - Open SEGGER J-Flash lite tool and click 'OK' after verifying the target device.

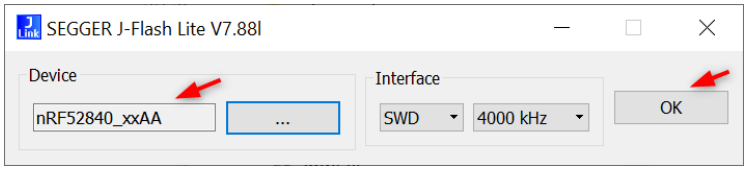



Fig. 22: SEGGER J-Flash Target selection

   - Load `<installed_COINES_SDK_location>/firmware/app3.X/bootloader_update/usb_ble_dfu_bootloader.hex` and click 'Program Device'.

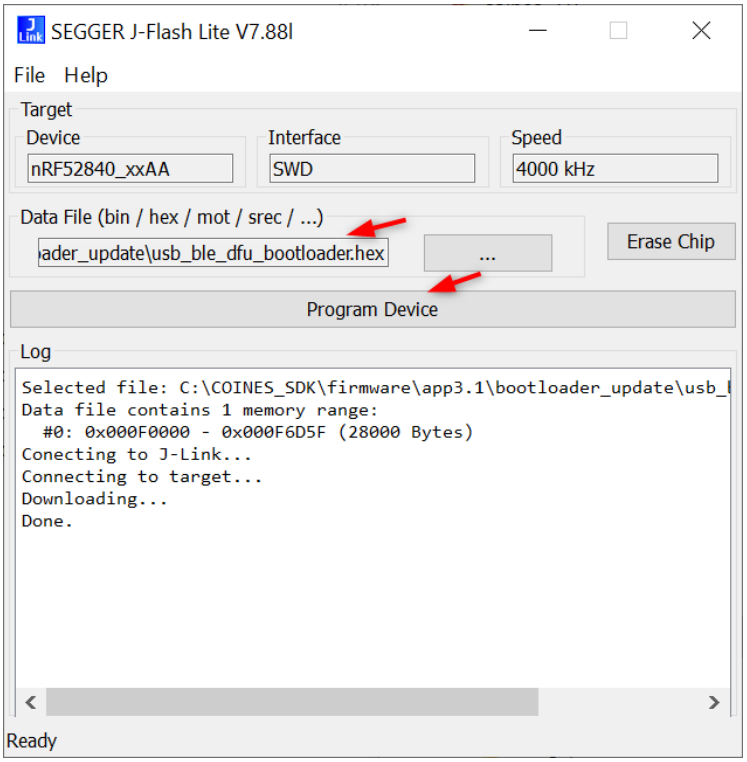



Fig. 23: SEGGER J-Flash Program Device

5. **How do I run multiple application boards using COINES SDK in a single computer?**
   When multiple USB devices are connected to a PC, by configuring Serial COM settings for a script, the user can communicate with them separately. Refer to 8.1.11 for implementation.

For more FAQs, visit Bosch Sensortec MEMS sensors forum.

# 16 Annexure

## 16.1 GPIO mapping

### 16.1.1 GPIO mapping of APP3.0 shuttle board pins

The APP3.0 shuttle board has 16 pins, seven on the left and nine on the right (with shuttle board pins facing downwards).

*Note*:

- In COINES SDK functions, the pins are addressed as on the APP3.0 shuttle board. For example, the GPIO #5 is addressed as `COINES_MINI_SHUTTLE_PIN_2_6`.
- Supported VDD voltages on APP3.0 are 0, 1.8V, and 2.8V
- Supported VDDIO voltage on APP3.0 is 1.8V

| Pin number on shuttle board | Name / function | Pin number on shuttle board | Name / function |
|---|---|---|---|
| 1_1 | VDD (1.8/2.8V) | 2_1 | SPI_CS |
| 1_2 | VDDIO (1.8) | 2_2 | SPI: SCK / $I^2C$: SCL |
| 1_3 | GND | 2_3 | SPI: MISO |
| 1_4 | GPIO0 | 2_4 | SPI: MOSI / $I^2C$: SDA |
| 1_5 | GPIO1 | 2_5 | GPIO4* |
| 1_6 | GPIO2 | 2_6 | GPIO5* |
| 1_7 | GPIO3 | 2_7 | IOXP_INT* |
| | | 2_8 | PlugDet* |
| | | 2_9 | EEPROM_RW |

*SPI pins for secondary interface - CS:GPIO4, SCK:GPIO5, MISO:IOXP_INT, MOSI:PlugDet

Table 1: Overview of APP3.0 shuttle board pins and their functions

### 16.1.2 GPIO mapping of APP3.1 shuttle board pins

The APP3.1 shuttle board has 16 pins, seven on the left and nine on the right (with shuttle board pins facing downwards).

*Note*:

- In COINES SDK functions, the pins are addressed as on the APP3.1 shuttle board. For example, the GPIO #5 is addressed as `COINES_MINI_SHUTTLE_PIN_2_6`.
- Supported VDD voltages on APP3.1 are 0, as well as a range from 1.8V to 3.3V
- Supported VDDIO voltages on APP3.1 are 0, as well as a range from 1.8V to 3.3V

| Pin number on shuttle board | Name / function | Pin number on shuttle board | Name / function |
|---|---|---|---|
| 1_1 | VDD (1.8 to 3.3V) | 2_1 | SPI_CS |
| 1_2 | VDDIO (1.8 to 3.3V) | 2_2 | SPI: SCK / $I^2C$: SCL |
| 1_3 | GND | 2_3 | SPI: MISO |
| 1_4 | GPIO0 | 2_4 | SPI: MOSI / $I^2C$: SDA |
| 1_5 | GPIO1 | 2_5 | GPIO4* |
| 1_6 | GPIO2 | 2_6 | GPIO5* |
| 1_7 | GPIO3 | 2_7 | IOXP_INT* |
| | | 2_8 | PlugDet* |
| | | 2_9 | EEPROM_RW |

*SPI pins for secondary interface - CS:GPIO4, SCK:GPIO5, MISO:IOXP_INT, MOSI:PlugDet

Table 2: Overview of APP3.1 shuttle board pins and their functions

### 16.2 APP3.X Flash memory layout

The APP3.X features an nRF52840 chip with 1MB of flash memory. The COINES SDK utilizes this memory with the following layout:

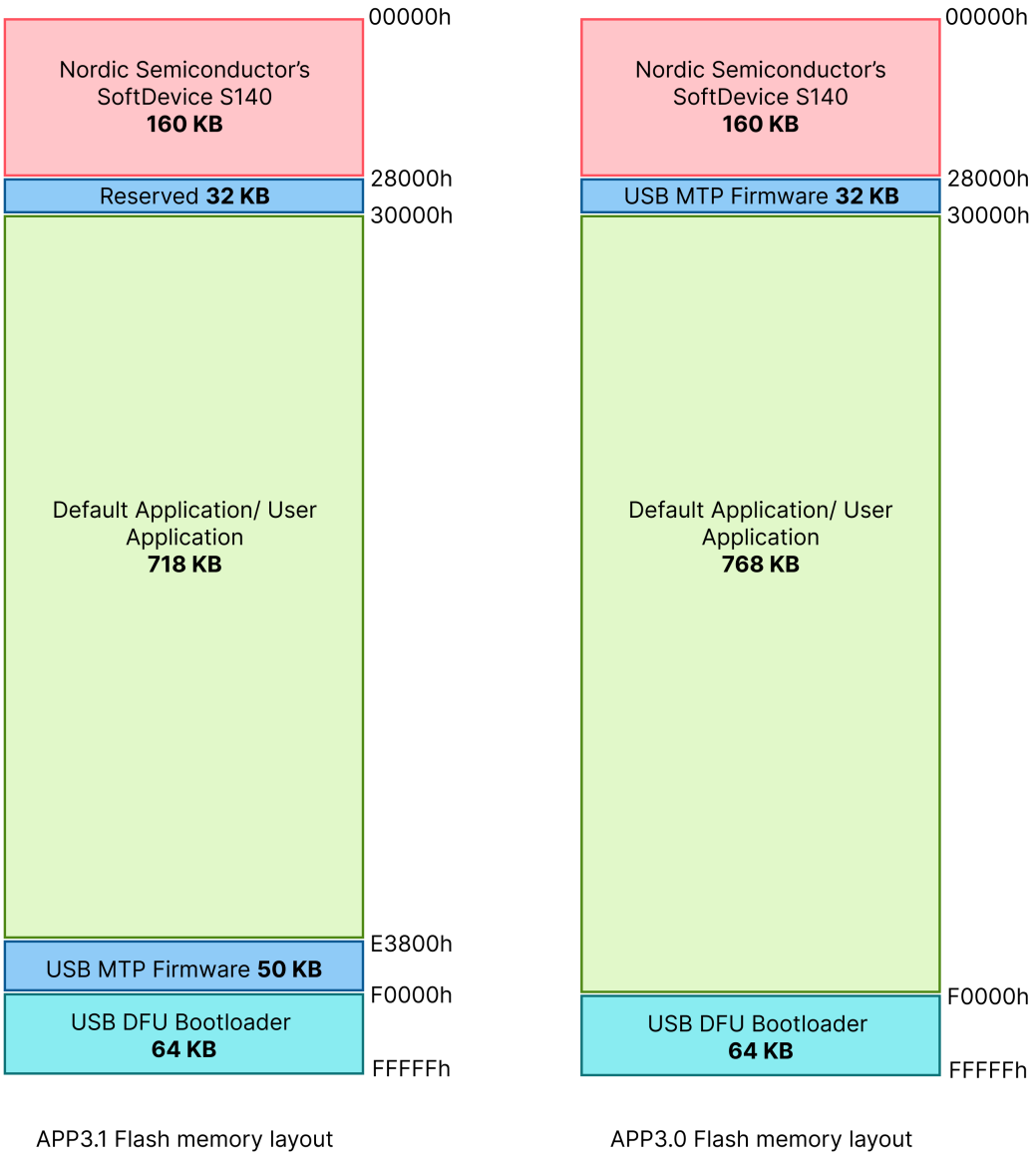



Fig. 24: APP3.X Flash memory layout

## 16.3 COINES SDK C functions

### 16.3.1 coinesAPI calls: Interface and board information

#### 16.3.1.1 coines_open_comm_intf

Opens the communication interface

```
int16_t coines_open_comm_intf(enum coines_comm_intf intf_type,void *arg);
```

In the case of an MCU target, the API waits indefinitely for a serial port or BLE connection (`MCU_APP30` target and `MCU_APP31` target).

In the case of a PC target, the user can configure communication settings either by passing the address of `coines_serial_com_config` or `ble_peripheral_info` to `*arg`.

Serial com configuration:

If `*arg` is NULL for `COINES_COMM_INTF_USB`, the first com port enumerated will be used for communication. The serial com configuration structure contains the following items. Refer to 8.1.11 for its implementation.

```
struct coines_serial_com_config
{
   uint32_t baud_rate; /*< Baud rate */
   uint16_t vendor_id; /*< vendor Id */
   uint16_t product_id; /*< Product Id */
   char* com_port_name; /*< serial com port name */
   uint16_t rx_buffer_size; /*< RX response buffer size */
};
```

BLE com configuration:

If `*arg` is NULL for `COINES_COMM_INTF_BLE`, the nearest Application board for the host BLE will be used for communication. The ble com configuration structure contains the following items. Refer to 8.1.10 for its implementation.

```
struct ble_peripheral_info
{
   char ble_address[COINES_CHAR_MAX_LEN]; /*< BLE device address */
   char ble_identifier[COINES_CHAR_MAX_LEN]; /*< BLE device identifier */
   uint32_t scan_timeout; /*< BLE device scan timeout */
    uint16_t rx_buffer_size; /*< RX response buffer size */
};
```

#### 16.3.1.2 coines_close_comm_intf

Closes the communication interface

```
int16_t coines_close_comm_intf(enum coines_comm_intf intf_type,void *arg);
```

#### 16.3.1.3 coines_get_board_info

Gets the board information

```
int16_t coines_get_board_info(struct coines_board_info *data);
```

The data structure contains the following items:

```
struct coines_board_info {
   /*!Board hardware ID */
   uint16_t hardware_id;
   /*!Board software ID */
   uint16_t software_id;
   /*!Type of the board like APP3.0*/
   uint8_t board;
   /*!Shuttle ID of the sensor connected*/
   uint16_t shuttle_id;
};
```

### 16.3.2 coinesAPI calls: GPIO oriented calls

#### 16.3.2.1 coines_set_pin_config

Sets the pin direction and the state

```
int16_t coines_set_pin_config(enum coines_multi_io_pin pin_number, enum
    coines_pin_direction direction, enum coines_pin_value pin_value);
```

#### 16.3.2.2 coines_get_pin_config

Gets the pin configuration

```
int16_t coines_get_pin_config(enum coines_multi_io_pin pin_number, enum
    coines_pin_direction *pin_direction, enum coines_pin_value *pin_value);
```

#### 16.3.2.3 coines_set_shuttleboard_vdd_vddio_config

This API configures the VDD and VDDIO of the sensor. For the APP3.0 board, voltage levels of 0, 1800mV, and 2800mV are supported. For the APP3.1 board, voltage levels of 0, as well as a range from 1800mV to 3300mV, are supported.

```
int16_t coines_set_shuttleboard_vdd_vddio_config(uint16_t vdd_millivolt, uint16_t
    vddio_millivolt);
```

### 16.3.3 coinesAPI calls: Sensor communication

#### 16.3.3.1 coines_config_i2c_bus

Configures the $I^2C$ bus

```
int16_t coines_config_i2c_bus(enum coines_i2c_bus bus, enum coines_i2c_mode i2c_mode);
```

The first argument refers to the bus on the board.

The following $I^2C$ modes are available:

```
COINES_I2C_STANDARD_MODE
COINES_I2C_FAST_MODE
COINES_I2C_SPEED_3_4_MHZ
COINES_I2C_SPEED_1_7_MHZ
```

#### 16.3.3.2 coines_config_spi_bus

This API configures the SPI bus of the board. The argument coines_spi_bus refers to the bus on the board. The SPI speed can be chosen in various discrete steps, as defined in enum coines_spi_speed in coines.h. (For example, COINES_SPI_SPEED_2_MHZ sets the SPI speed to 2 MHz.)

```
int16_t coines_config_spi_bus(enum coines_spi_bus bus, uint32_t spi_speed, enum
    coines_spi_mode spi_mode);
```

#### 16.3.3.3 coines_config_i2s_bus

This API is used to configure the $I^2S$ bus to match the TDM configuration.

```
int16_t coines_config_i2s_bus(uint16_t data_words, coines_tdm_callback callback);
```

Arguments:

- data_words: The number of words to use in the buffer. Max is set at COINES_TDM_BUFFER_SIZE_WORDS.
- callback: register a callback to be called to process and copy the data

#### 16.3.3.4 coines_deconfig_spi_bus

This API is used to de-configure the SPI bus.

```
int16_t coines_deconfig_spi_bus(enum coines_spi_bus bus);
```

#### 16.3.3.5 coines_deconfig_i2c_bus

This API is used to de-configure the $I^2C$ bus.

```
int16_t coines_deconfig_i2c_bus(enum coines_i2c_bus bus);
```

#### 16.3.3.6 coines_deconfig_i2s_bus

This API is used to stop the $I^2S$/TDM interface from reading data from the sensor.

```
void coines_deconfig_i2s_bus(void);
```

#### 16.3.3.7 coines_write_i2c

This API writes 8-bit register data to the I²C device at `COINES_I2C_BUS_0`.

```
int8_t coines_write_i2c(enum coines_i2c_bus bus,uint8_t dev_addr, uint8_t reg_addr,
    uint8_t *reg_data, uint16_t count);
```

Arguments:

- `bus`: I²C bus to be used
- `dev_addr`: I²C device address
- `reg_addr`: Starting address for writing the data
- `reg_data`: Data to be written
- `count`: Number of bytes to write

#### 16.3.3.8 coines_read_i2c

This API reads 8-bit register data from the I²C device at `COINES_I2C_BUS_0`.

```
int8_t coines_read_i2c(enum coines_i2c_bus bus,uint8_t dev_addr, uint8_t reg_addr,
    uint8_t *reg_data, uint16_t count);
```

Arguments:

- `bus`: I²C bus to be used
- `dev_addr`: I²C device address
- `reg_addr`: Starting address for reading the data
- `reg_data`: Buffer to take up the read data
- `count`: Number of bytes to read

#### 16.3.3.9 coines_write_16bit_i2c

This API writes 16-bit register data to the I²C device.

```
int8_t coines_write_16bit_i2c(enum coines_i2c_bus bus, uint8_t dev_addr, uint16_t
    reg_addr, void *reg_data, uint16_t count, enum coines_i2c_transfer_bits
    i2c_transfer_bits);
```

Arguments:

- `bus`: I²C bus to be used
- `dev_addr`: I²C device address
- `reg_addr`: Starting address for writing the data
- `reg_data`: Data to be written
- `count`: Number of bytes to write
- `i2c_transfer_bits`: bits to transfer

#### 16.3.3.10 coines_read_16bit_i2c

This API reads 16-bit register data from the I²C device.

```
int8_t coines_read_16bit_i2c(enum coines_i2c_bus bus, uint8_t dev_addr, uint16_t
    reg_addr, void *reg_data, uint16_t count, enum coines_i2c_transfer_bits
    i2c_transfer_bits);
```

Arguments:

- `bus`: I²C bus to be used
- `dev_addr`: I²C device address
- `reg_addr`: Starting address for reading the data
- `reg_data`: Data read from the sensor
- `count`: Number of bytes to read
- `i2c_transfer_bits`: bits to transfer

#### 16.3.3.11 coines_i2c_set

This API is used to write the data in I2C communication.

```
int8_t coines_i2c_set(enum coines_i2c_bus bus, uint8_t dev_addr, uint8_t *data, uint8_t
    count);
```

Arguments:

- `bus`: I²C bus to be used
- `dev_addr`: I²C device address
- `data`: Data to be written
- `count`: Number of bytes to write

#### 16.3.3.12 coines_i2c_get

This API is used to read the data in I2C communication.

```
int8_t coines_i2c_get(enum coines_i2c_bus bus, uint8_t dev_addr, uint8_t *data, uint8_t
    count);
```

Arguments:

- `bus`: I²C bus to be used
- `dev_addr`: I²C device address
- `data`: Data read from the sensor
- `count`: Number of bytes to read

#### 16.3.3.13 coines_write_spi

This API writes 8-bit register data to the SPI device at `COINES_SPI_BUS_0`.

```
int8_t coines_write_spi(enum coines_spi_bus bus,uint8_t cs_pin, uint8_t reg_addr,
    uint8_t *reg_data, uint16_t count);
```

Arguments:

- bus: SPI bus to be used
- cs_pin: Chip select pin number
- reg_addr: Starting address for writing the data
- reg_data: Data to be written.
- count: Number of bytes to write

#### 16.3.3.14 coines_read_spi

This API reads 8-bit register data from the SPI device at COINES_SPI_BUS_0.

```
int8_t coines_read_spi(enum coines_spi_bus bus, uint8_t cs_pin, uint8_t reg_addr,
    uint8_t *reg_data, uint16_t count);
```

Arguments:

- bus: SPI bus to be used
- cs_pin: Chip select pin number
- reg_addr: Starting address for reading the data
- reg_data: Buffer to take up the read data
- count: Number of bytes to read

#### 16.3.3.15 coines_write_16bit_spi

This API writes 16-bit register data to the SPI device.

```
int8_t coines_write_16bit_spi(enum coines_spi_bus bus, uint8_t cs_pin, uint16_t
    reg_addr, void *reg_data, uint16_t count, enum coines_spi_transfer_bits
    spi_transfer_bits);
```

Arguments:

- bus: SPI bus to be used
- cs_pin: Chip select pin number
- reg_addr: Starting address for writing the data
- reg_data: Data to be written
- count: Number of bytes to write
- spi_transfer_bits: bits to transfer

#### 16.3.3.16 coines_read_16bit_spi

This API reads 16-bit register data from the SPI device.

```
int8_t coines_read_16bit_spi(enum coines_spi_bus bus, uint8_t cs_pin, uint16_t
    reg_addr, void *reg_data, uint16_t count, enum coines_spi_transfer_bits
    spi_transfer_bits);
```

Arguments:

- `bus`: SPI bus to be used
- `cs_pin`: Chip select pin number
- `reg_addr`: Starting address for reading the data
- `reg_data`: Data read from the sensor
- `count`: Number of bytes to read
- `spi_transfer_bits`: bits to transfer

#### 16.3.3.17 coines_delay_msec

This API introduces the delay in milliseconds.

```
void coines_delay_msec(uint32_t delay_ms);
```

#### 16.3.3.18 coines_delay_usec

This API introduces the delay in microseconds.

```
void coines_delay_usec(uint32_t delay_us);
```

#### 16.3.3.19 coines_uart_init

This API is used to initialize the UART communication.

```
int8_t coines_uart_init(enum coines_uart_instance uart_instance, enum
    coines_uart_parity parity, enum coines_uart_flow_control flow_control, uint32_t
    baud_rate);
```

Arguments:

- `uart_instance`: Specifies the UART instance
- `parity`: UART parity
- `flow_control`: UART flow control mode
- `baud_rate`: UART baud rate

#### 16.3.3.20 coines_uart_read

This API is used to read the data in UART communication.

```
uint16_t coines_uart_read(enum coines_uart_instance uart_instance, uint8_t *buffer,
    uint16_t length);
```

Arguments:

- `uart_instance`: Specifies the UART instance
- `buffer`: Pointer to the buffer to store the data
- `length`: Length of the buffer

#### 16.3.3.21 coines_uart_write

This API is used to write the data in UART communication.

```
int8_t coines_uart_write(enum coines_uart_instance uart_instance, uint8_t *buffer,
    uint16_t length);
```

Arguments:

- `uart_instance`: Specifies the UART instance
- `buffer`: Pointer to the data buffer that need to be written
- `length`: Length of the buffer

### 16.3.4 coinesAPI calls: Streaming feature

*Note*:

1. The APIs below are supported only on a PC Target.
2. A simpler approach to using `coines_attach_interrupt()` API is available for MCU.

#### 16.3.4.1 coines_config_streaming

This API sets the configuration for streaming sensor data.

```
int16_t coines_config_streaming(uint8_t channel_id, struct coines_streaming_config
    *stream_config, struct coines_streaming_blocks *data_blocks);
```

Arguments:

- `channel_id`: An integer number that can be used as an identifier/index to the sensor data that is streamed for this setting.
- `stream_config`: Contains information regarding interface settings and streaming configuration.
- `coines_streaming_blocks`: Contains information regarding the numbers of blocks to read, register address, and size for each block

*Note*:
The parameters below should always be set:

- `data_block.no_of_blocks`: The number of blocks to stream (must at least be one)
- For each block b:
  - `data_block.reg_start_addr[b]`: start address of the block in the register map
  - `stream_block.no_of_data_bytes[b]`: number of bytes to read, starting from the start address
  - `data_block.block_type[b]`: Specifies the type of block — `COINES_READ_BLOCK` or `COINES_WRITE_BLOCK`.
    The write type is valid only with streaming modes `COINES_STREAMING_MODE_BLOCK_IO_POLLING` and `COINES_STREAMING_MODE_BLOCK_IO_INTERRUPT`.

- data_block.wait_time_us[b]: wait time after each block IO operation in microseconds
- data_block.write_data[b]: data buffer to be written (if the block type is COINES_WRITE_BLOCK). The size of the buffer should be equal to stream_block.no_of_data_bytes[b].

For reading data from the $I^2C$ bus, set the parameters below:

- stream_config.intf = COINES_SENSOR_INTF_I2C;
- stream_config.i2c_bus: $I^2C$ bus
- stream_config.dev_addr: $I^2C$ address of the sensor

For reading data from the SPI bus, set the parameters below:

- stream_config.intf = COINES_SENSOR_INTF_SPI;
- stream_config.spi_bus: SPI bus
- stream_config.cs_pin: The CS pin of the sensor. Information can be obtained from the shuttle board documentation for the sensor.

When polling mode is requested, set the parameters below:

- stream_config.sampling_units:
  either milliseconds (COINES_SAMPLING_TIME_IN_MILLI_SEC)
  or microseconds (COINES_SAMPLING_TIME_IN_MICRO_SEC)
- stream_config.sampling_time: sampling period in the unit as defined in stream_config.sampling_units

When interrupt mode is requested, set the parameters below:

- stream_config.int_pin: The interrupt pin that triggers the sensor read-out. If the interrupt output of the sensor is used, the required information about the pin number can be obtained from the shuttle board documentation for the sensor.
- stream_config.int_timestamp: It can be configured if the sensor data is tagged with a timestamp (COINES_TIMESTAMP_ENABLE) or not (COINES_TIMESTAMP_DISABLE).

#### 16.3.4.2 coines_start_stop_streaming

This API starts or stops sensor data streaming.

```
int16_t coines_start_stop_streaming(enum coines_streaming_mode stream_mode, uint8_t
    start_stop);
```

Arguments:

- stream_mode: Supported streaming modes:
  - COINES_STREAMING_MODE_POLLING
  - COINES_STREAMING_MODE_INTERRUPT
  - COINES_STREAMING_MODE_DMA_INTERRUPT
  - COINES_STREAMING_MODE_BLOCK_IO_POLLING
  - COINES_STREAMING_MODE_BLOCK_IO_INTERRUPT
- start_stop: flag to either start (COINES_STREAMING_START) or stop (COINES_STREAMING_STOP) the streaming

#### 16.3.4.3 coines_read_stream_sensor_data

This API reads the data streamed from the sensor.

```
int16_t coines_read_stream_sensor_data(uint8_t sensor_id, uint32_t number_of_samples,
    uint8_t *data, uint32_t *valid_samples_count);
```

Arguments:

- `sensor_id`: id of the sensor
- `number_of_samples`: number of samples the user wishes to read (not implemented)
- `data`: data buffer
  - Interrupt streaming - Packet counter + Register data + Timestamp
  - Polling streaming - Register data
- `valid_samples_count`: number of samples the user has actually received (may be less than `number_of_samples`)

Example of a packet:

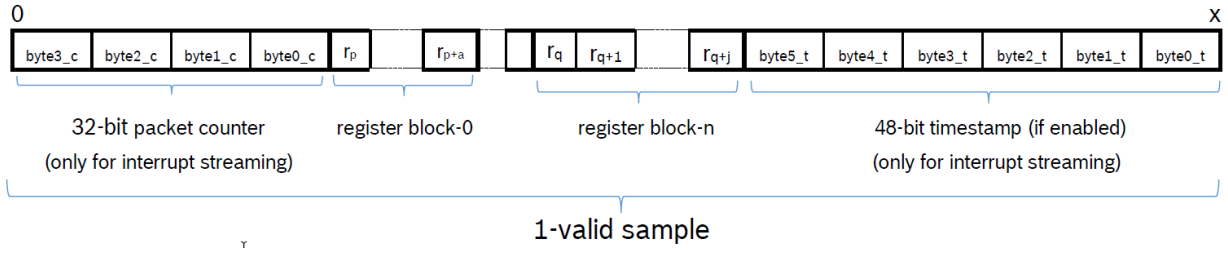



Fig. 25: Format of streaming packages

The following definitions are for the abbreviations in Figure 25:

- $r_p$: Value at register address p
- a: Size of register block−0
- $r_{p+a}$: Value at register address p

In the same manner, definitions are applicable to $r_q$, j and $r_{q+j}$. See the `coines_streaming_blocks` structure for information regarding register blocks.

Determine the packet counter and timestamp as follows:

```
packet_counter = (byte3_c « 24) | (byte2_c « 16) | (byte1_c « 8) | (byte0_c)
```

```
timestamp = (byte5_t « 40) | (byte4_t « 32) | (byte3_t « 24) | (byte2_t « 16) |
(byte1_t « 8) | (byte0_t)
```

The 48-bit timestamp is enabled by using
`coines_trigger_timer(COINES_TIMER_START, COINES_TIMESTAMP_ENABLE);`

Determine the Timestamp in microseconds using the formula below:

$$Timestamp\ (\mu s) = \frac{48bit\_timestamp}{30}$$

#### 16.3.4.4 coines_trigger_timer

This API triggers the firmware timer and enables or disables the time stamp feature.

```
int16_t coines_trigger_timer(enum coines_timer_config tmr_cfg,enum
    coines_time_stamp_config ts_cfg);
```

Arguments:

- `tmr_cfg`: start, stop or reset the timer (`COINES_TIMER_START`, `COINES_TIMER_STOP` or `COINES_TIMER_RESET`)
- `ts_cfg`: Enables/disables the microcontroller timestamp (`COINES_TIMESTAMP_ENABLE` or `COINES_TIMESTAMP_DISABLE`)

### 16.3.5 coinesAPI calls: Other useful APIs

#### 16.3.5.1 coines_get_millis

This API returns the number of milliseconds passed since the program started.

```
uint32_t coines_get_millis();
```

#### 16.3.5.2 coines_get_micro_sec

Returns the number of microseconds passed since the program started

```
uint64_t coines_get_micro_sec();
```

#### 16.3.5.3 coines_attach_interrupt

This API attaches an interrupt to a Multi-IO pin. It only works on an MCU target.

```
void coines_attach_interrupt(enum coines_multi_io_pin pin_number,void
    (*callback)(uint32_t, uint32_t),enum coines_pin_interrupt_mode int_mode);
```

Arguments:

- `pin_number`: Multi-IO pin
- `callback`: Name of the function to be called on interrupt detection
- `int_mode`: Trigger modes - change (`COINES_PIN_INTERRUPT_CHANGE`),
  rising edge (`COINES_PIN_INTERRUPT_RISING_EDGE`),
  falling edge (`COINES_PIN_INTERRUPT_FALLING_EDGE`)

#### 16.3.5.4 coines_detach_interrupt

This API detaches interrupt from a Multi-IO pin. It only works on an MCU target.

```
void coines_detach_interrupt(enum coines_multi_io_pin pin_number);
```

Arguments:

- `pin_number`: Multi-IO pin

#### 16.3.5.5 coines_intf_available

Return the number of bytes available in the read buffer of the interface. It only works on an APP3.x MCU target.

```
uint16_t coines_intf_available(enum coines_comm_intf intf);
```

Arguments:

- `intf`: Type of interface (USB, COM, or BLE)

#### 16.3.5.6 coines_intf_connected

This API checks if the interface is connected. It only works on an APP3.x MCU target.

```
bool coines_intf_connected(enum coines_comm_intf intf);
```

Arguments:

- `intf`: Type of interface (USB, COM, or BLE)

#### 16.3.5.7 coines_flush_intf

This API flushes the write buffer. It only works on an APP3.x MCU target.

```
void coines_flush_intf(enum coines_comm_intf intf);
```

Arguments:

- `intf`: Type of interface (USB, COM, or BLE)

#### 16.3.5.8 coines_read_intf

This API reads data over the specified interface. It only works on an APP3.x MCU target.

```
uint16_t coines_read_intf(enum coines_comm_intf intf, void *buffer, uint16_t len);
```

Arguments:

- `intf`: Type of interface (USB, COM, or BLE)
- `buffer`: Pointer to the buffer to store the data
- `len`: Length of the buffer

#### 16.3.5.9 coines_read_intf_non_block

This API Reads data from the specified communication interface in a non-blocking manner.Callback should be register with coines_open_comm_intf API.It only works on an APP3.x MCU target.

```
uint16_t coines_read_intf_non_block(enum coines_comm_intf intf, void *buffer, uint16_t
    len);
```

Arguments:

- `intf`: Type of interface (USB, COM, or BLE)

- buffer: Pointer to the buffer storing the data
- len: Length of the buffer

#### 16.3.5.10 coines_write_intf

This API writes data over the specified interface. It only works on an APP3.x MCU target.

```
uint16_t coines_write_intf(enum coines_comm_intf intf, void *buffer, uint16_t len);
```

Arguments:

- intf: Type of interface (USB, COM, or BLE)
- buffer: Pointer to the buffer storing the data
- len: Length of the buffer

#### 16.3.5.11 coines_write_intf_non_block

This API Writes data to the specified communication interface in a non-blocking manner.Callback should be register with coines_open_comm_intf API.It only works on an APP3.x MCU target.

```
uint16_t coines_write_intf_non_block(enum coines_comm_intf intf, void *buffer, uint16_t
    len);
```

Arguments:

- intf: Type of interface (USB, COM, or BLE)
- buffer: Pointer to the buffer storing the data
- len: Length of the buffer

#### 16.3.5.12 coines_get_version

Returns pointer to COINES version string

```
char* coines_get_version(void);
```

#### 16.3.5.13 coines_soft_reset

Resets the device. After the device is reset, it opens to the address specified in makefile (APP_START_ADDRESS).

```
void coines_soft_reset(void);
```

#### 16.3.5.14 coines_read_temp_data

This API is used to read the temperature sensor data.

```
int16_t coines_read_temp_data(float *temp_data);
```

Arguments:

- `temp_data`: Buffer to retrieve the sensor data in degree Celsius.

#### 16.3.5.15 coines_read_bat_status

This API is used to read the battery status.

```
int16_t coines_read_bat_status(uint16_t *bat_status_mv, uint8_t *bat_status_percent);
```

Arguments:

- `bat_status_mv`: Buffer to retrieve the battery status in millivolt.
- `bat_status_percent`: Buffer to retrieve the battery status in .

#### 16.3.5.16 coines_ble_config

This API is used to configure BLE name and power. Call this API before calling coines_open_comm_intf API.

```
int16_t coines_ble_config(struct coines_ble_config *ble_config);
```

Arguments:

- `ble_config`: structure holding the ble name and power details

#### 16.3.5.17 coines_set_led

This API is used to set the led state (on or off).

```
int16_t coines_set_led(enum coines_led led,enum coines_led_state led_state);
```

Arguments:

- `led`: led to which the state has to be set
- `led_state`: state to be set to the given led

#### 16.3.5.18 coines_timer_config

This API is used to configure the hardware timer.

```
int16_t coines_timer_config(enum coines_timer_instance instance, void* handler);
```

Arguments:

- `instance`: timer instance
- `handler`: callback to be called when timer expires

#### 16.3.5.19 coines_timer_deconfig

This API is used to de-configure the hardware timer.

```
int16_t coines_timer_deconfig(enum coines_timer_instance instance);
```

Arguments:

- `instance`: timer instance

#### 16.3.5.20 coines_timer_start

This API is used to start the configured hardware timer.

```
int16_t coines_timer_start(enum coines_timer_instance instance, uint32_t timeout);
```

Arguments:

- `instance`: timer instance
- `timeout`: timeout in microseconds

#### 16.3.5.21 coines_timer_stop

This API is used to stop the hardware timer.

```
int16_t coines_timer_stop(enum coines_timer_instance instance);
```

Arguments:

- `instance`: timer instance

#### 16.3.5.22 coines_get_realtime_usec

This API is used to get the current counter (RTC) reference time in usec.

```
uint32_t coines_get_realtime_usec(void);
```

#### 16.3.5.23 coines_delay_realtime_usec

This API is used to introduce delay based on high precision RTC (LFCLK crystal) with the resolution of 30.517 usec.

```
void coines_delay_realtime_usec(uint32_t period);
```

Arguments:

- `period`: required delay in microseconds

#### 16.3.5.24 coines_attach_timed_interrupt

This API Attaches a timed interrupt to a Multi-IO pin.

```
int16_t coines_attach_timed_interrupt(enum coines_multi_io_pin pin_number, void
    (*timed_interrupt_cb)(uint64_t,uint32_t,uint32_t), enum coines_pin_interrupt_mode
    int_mode);
```

Arguments:

- `pin_number`: Multi-IO pin.

- `timed_interrupt_cb`: Name of the function to be called on detection of interrupt.
- `int_mode`: Trigger modes - change, rising edge, falling edge.

#### 16.3.5.25 coines_detach_timed_interrupt

Detaches a timed interrupt from a Multi-IO pin.

```
int16_t coines_detach_timed_interrupt(enum coines_multi_io_pin pin_number);
```

Arguments:

- `pin_number`: Multi-IO pin

#### 16.3.5.26 coines_echo_test

This API is used to test the communication.

```
int16_t coines_echo_test(uint8_t *data, uint16_t length);
```

Arguments:

- `data`: Data to be sent for testing
- `length`: Length of the data

#### 16.3.5.27 coines_shuttle_eeprom_write

This API is used to write the content into shuttle eeprom.

```
int16_t coines_shuttle_eeprom_write(uint16_t start_addr, uint8_t *buffer, uint16_t
    length);
```

Arguments:

- `start_addr`: EEPROM write address
- `buffer`: Pointer to the buffer
- `length`: Length of the buffer

#### 16.3.5.28 coines_shuttle_eeprom_read

This API is used to read the content from shuttle eeprom.

```
int16_t coines_shuttle_eeprom_read(uint16_t start_addr, uint8_t *buffer, uint16_t
    length);
```

Arguments:

- `start_addr`: EEPROM read address
- `buffer`: Pointer to the buffer
- `length`: Length of the buffer

#### 16.3.5.29 coines_yield

This API can be defined to perform a task when yielded from an ongoing blocking call.

```
void coines_yield(void);
```

#### 16.3.5.30 coines_execute_critical_region

This API is used to execute the function inside a critical region.

```
void coines_execute_critical_region(coines_critical_callback callback);
```

Arguments:

- `callback`: Function to execute

#### 16.3.5.31 coines_scan_ble_devices

This API is used to connect to the BLE Adapter and return a list of BLE peripherals found during the BLE scan.

```
int16_t coines_scan_ble_devices(struct ble_peripheral_info *ble_info, uint8_t
    *peripheral_count, uint32_t scan_timeout_ms)
```

Arguments:

- `ble_info`: an array of structs with information regarding detected BLE peripheral devices
- `peripheral_count`: The number of BLE peripherals found
- `scan_timeout_ms`: timeout for BLE scan

## 16.4 COINES SDK Python functions

As coinespy is only a wrapper on top of coinesAPI, the following API documentation is limited to the wrapper only. Details about the meaning of variables and functionality can be found in the corresponding coinesAPI documentation in the chapter above. The following function calls

are defined within the class `CoinesBoard`. Therefore, to access the functions, the user must first create an object of that class.

```
import coinespy as cpy
coinesboard = cpy.CoinesBoard()
```

### 16.4.1 coinespy API calls: Interface and board information

#### 16.4.1.1 open_comm_interface

Sets the communication interface between the board and PC to USB, Serial or BLE

```
coinesboard.open_comm_interface(interface=CommInterface.USB, serial_com_config:
    SerialComConfig = None,
ble_com_config: BleComConfig = None) -> ErrorCodes
```

For the definition of `CommInterface`, refer to 16.4.6.3.

#### 16.4.1.2 close_comm_interface

Disposes the resources used by the USB/serial/BLE communication

```
coinesboard.close_comm_interface(arg=None) -> ErrorCodes
```

#### 16.4.1.3 get_board_info

Obtains board-specific information

```
BoardInfo = coinesboard.get_board_info()

# Return:
BoardInfo.HardwareId # Hardware ID
BoardInfo.SoftwareId # Firmware version information
BoardInfo.Board      # Board type
BoardInfo.ShuttleID  # ID of shuttle, in case a shuttle is detected
```

#### 16.4.1.4 scan_ble_devices

This API connects to the BLE Adapter and returns a list of BLE peripherals found during the BLE scan.

```
ble_info, peripheral_count = coinesboard.scan_ble_devices(scan_timeout_ms=0) ->
    Tuple[list, int]
```

For the definition of parameters, refer to 16.3.5.31.

#### 16.4.1.5 echo_test

This API is used to test the communication.

```
coinesboard.echo_test(data: List[int]) -> ErrorCodes
```

Arguments:

- `data`: Data to be sent for testing.

### 16.4.2 coinespy API calls: GPIO oriented calls

#### 16.4.2.1 set_pin_config

Configures the state, level, and direction of a GPIO pin

```
coinesboard.set_pin_config(pin_number: MultiIOPin, direction: PinDirection,
    output_state: PinValue) -> ErrorCodes
```

For the definition of `MultiIOPin`, refer to 16.4.6.9. For the definition of `PinDirection`, refer to 16.4.6.1. For `PinValue`, refer to 16.4.6.2.

#### 16.4.2.2 get_pin_config

Obtains information regarding the Pin's state, level, and direction

```
PinConfigInfo = coinesboard.get_pin_config(pin_number: MultiIOPin)

# Return:
PinConfigInfo.direction     # 0: INPUT, 1: OUTPUT
PinConfigInfo.switch_state   # 0: OFF, 1: ON
PinConfigInfo.level         # 1: HIGH, 0: LOW
```

#### 16.4.2.3 set_shuttleboard_vdd_vddio_config

Sets the VDD and VDDIO voltage level

```
coinesboard.set_shuttleboard_vdd_vddio_config(vdd_val: float = None, vddio_val: float =
    None) -> ErrorCodes

# Example: coinesboard.set_shuttleboard_vdd_vddio_config(3.3, 3.3)
```

#### 16.4.2.4 set_vdd

Sets the VDD voltage level

```
coinesboard.set_vdd(vdd_val: float = None) -> ErrorCodes

# Example: coinesboard.set_vdd(3.3)
```

#### 16.4.2.5 set_vddio

Sets the VDDIO voltage level

```
coinesboard.set_vddio(vdd_val: float = None) -> ErrorCodes

# Example: coinesboard.set_vddio(3.3)
```

### 16.4.3 coinespy API calls: Sensor communication

For the definition of SPIBus, refer to 16.4.6.12. For the definition of I2CBus, refer to 16.4.6.11.

#### 16.4.3.1 config_i2c_bus

Configures the $I^2C$ bus

```
coinesboard.config_i2c_bus(bus: I2CBus, i2c_address: int, i2c_mode: I2CMode) ->
    ErrorCodes
```

For the definition of I2CMode, refer to 16.4.6.4.

#### 16.4.3.2 config_spi_bus

Configures the SPI bus of the board

```
coinesboard.config_spi_bus(bus: SPIBus, cs_pin: MultiIOPin, spi_speed=SPISpeed,
    spi_mode=SPIMode) -> ErrorCodes
```

For the definition of MultiIOPin, refer to 16.4.6.9. For the definition of SPISpeed, refer to 16.4.6.5. For the definition of SPIMode, refer to 16.4.6.8.

#### 16.4.3.3 deconfig_i2c_bus

This API is used to de-configure the $I^2C$ bus.

```
coinesboard.deconfig_i2c_bus(bus: I2CBus) -> ErrorCodes
```

#### 16.4.3.4 deconfig_spi_bus

This API is used to de-configure the SPI bus.

```
coinesboard.deconfig_spi_bus(bus: SPIBus) -> ErrorCodes
```

#### 16.4.3.5 write_i2c

Writes 8-bit register data to the $I^2C$

```
coinesboard.write_i2c(bus: I2CBus, register_address: int, register_value: int,
    sensor_interface_detail: int = None) -> ErrorCodes
```

For the definition of parameters, refer to 16.3.3.7.

#### 16.4.3.6 read_i2c

Reads 8-bit register data from the $I^2C$

```
register_data = coinesboard.read_i2c(bus: I2CBus, register_address: int,
    number_of_reads=1, sensor_interface_detail: int = None)
```

For the definition of parameters, refer to 16.3.3.8.

#### 16.4.3.7 write_16bit_i2c

Writes 16-bit register data to the $I^2C$

```
coinesboard.write_16bit_i2c(bus: I2CBus, register_address: int,
register_value: int, sensor_interface_detail: int = None, i2c_transfer_bits:
    I2CTransferBits = I2CTransferBits.I2C16BIT) -> ErrorCodes
```

For the definition of parameters, refer to 16.3.3.9.

#### 16.4.3.8 read_16bit_i2c

Reads 16-bit register data from the $I^2C$

```
register_data = coinesboard.read_16bit_i2c(bus: I2CBus, register_address: int,
    number_of_reads=2,
sensor_interface_detail: int = None, i2c_transfer_bits: I2CTransferBits =
    I2CTransferBits.I2C16BIT)
```

For the definition of parameters, refer to 16.3.3.10.

#### 16.4.3.9 write_spi

Writes 8-bit register data to the SPI device

```
coinesboard.write_spi(bus: SPIBus, register_address: int, register_value: int,
    sensor_interface_detail: int = None) -> ErrorCodes
```

For the definition of parameters, refer to 16.3.3.13.

#### 16.4.3.10 read_spi

Reads 8-bit register data from the SPI device.

```
register_data = coinesboard.read_spi(bus: SPIBus, register_address: int,
    number_of_reads=1, sensor_interface_detail: int = None)
```

For the definition of parameters, refer to 16.3.3.14.

#### 16.4.3.11 write_16bit_spi

Writes 16-bit register data to the SPI device

```
coinesboard.write_16bit_spi(bus: SPIBus, register_address: int, register_value: list,
    sensor_interface_detail: int = None, spi_transfer_bits: SPITransferBits =
    SPITransferBits.SPI16BIT) -> ErrorCodes
```

For the definition of parameters, refer to 16.3.3.15.

#### 16.4.3.12 read_16bit_spi

Reads 16-bit register data from the SPI device

```
register_data = coinesboard.read_16bit_spi(bus: SPIBus, register_address: int,
    number_of_reads=2, sensor_interface_detail: int = None, spi_transfer_bits:
    SPITransferBits = SPITransferBits.SPI16BIT)
```

For the definition of parameters, refer to 16.3.3.16.

#### 16.4.3.13 delay_milli_sec

Introduces delay in millisecond.

```
coinesboard.delay_milli_sec(time_in_milli_sec=100)
```

#### 16.4.3.14 delay_micro_sec

Introduces delay in microsecond.

```
coinesboard.delay_micro_sec(time_in_micro_sec=1)
```

### 16.4.4 coinespy API calls: Streaming feature

#### 16.4.4.1 config_streaming

Sets the configuration for streaming sensor data.

```
coinesboard.config_streaming(sensor_id: int,
   stream_config: StreamingConfig, data_blocks: StreamingBlocks) -> ErrorCodes
```

Arguments:

- `sensor_id`: An integer number that can be used as an identifier/index to the sensor data that will be streamed for this setting.
- `stream_config`: Contains information regarding interface settings and streaming configuration
- `data_blocks`: Contains information regarding numbers of blocks to read, register address, and size for each block

The parameters below should always be set:

- `data_blocks.NoOfBlocks`: number of blocks to stream (must at least be one)

- For each block b:
  - `data_blocks.RegStartAddr[b]`: start address of the block in the register map
  - `data_blocks.NoOfDataBytes[b]`: number of bytes to read, starting from the start address
  - `data_block.BlockType[b]`: Specifies the type of block — read or write. Refer to 16.4.6.15 for the definition of BlockType.
    The write type is valid only with streaming modes `COINES_STREAMING_MODE_BLOCK_IO_POLLING` and `COINES_STREAMING_MODE_BLOCK_IO_INTERRUPT`.
  - `data_block.WaitTimeUs[b]`: wait time after each block IO operation in microseconds
  - `data_block.WriteData[b]`: data buffer to be written (if the block type is `BlockType.WRITE_BLOCK`). The size of the buffer should be equal to `stream_block.NoOfDataBytes[b]`.

For reading data from the $I^2C$ bus, set the parameters below:

- `stream_config.Intf = cpy.SensorInterface.I2C.value`
- `stream_config.I2CBus`: $I^2C$ bus
- `stream_config.DevAddr`: $I^2C$ address of the sensor

For reading data from the SPI bus, set the parameters below:

- `stream_config.Intf = cpy.SensorInterface.SPI.value;`
- `stream_config.SPIBus`: SPI bus
- `stream_config.CSPin`: CS pin of the sensor, information can be obtained from the shuttle board documentation for the sensor.
- `stream_config.SPIType`: 0 : 8-bit SPI; 1 : 16-bit SPI

When polling mode is requested, set the parameters below:

- `stream_config.SamplingUnits`: Either milliseconds or microseconds. Refer to 16.4.6.17.
- `stream_config.SamplingTime`: sampling period in the unit as defined in `stream_config.SamplingUnits`

When interrupt mode is requested, set the parameters below:

- `stream_config.IntPin`: Pin of the interrupt, which triggers the sensor read-out. If the interrupt output of the sensor is used, the required information about the pin number can be obtained from the shuttle board documentation for the sensor.
- `stream_config.IntTimeStamp`: It can be configured if the sensor data is tagged with a timestamp - 1 or not - 0.
- `stream_config.HwPinState`: State of the hardware pin connected to the interrupt line - 0/1: Low/high

The parameters below are common for both streaming types:

- `stream_config.IntlineCount`: Number of interrupt lines to be used for monitoring interrupts.
- `stream_config.IntlineInfo`: List of pin numbers that correspond to interrupt lines being used for interrupt monitoring.
- `stream_config.ClearOnWrite`: 0/1 : Disable/enable "clear on write" feature

The parameters below should only be set when stream_config.ClearOnWrite = 1:

- `stream_config.ClearOnWriteConfig.StartAddress`: Address of the sensor register at which the process of clearOnWrite should initiate.
- `stream_config.ClearOnWriteConfig.DummyByte`: Number of padding bytes that must be added before clearing the bytes starting from the designated address.
- `stream_config.ClearOnWriteConfig.NumBytesToClear`: Number of bytes that need to be cleared.

Below is the Python code snippet for interrupt streaming.

```
# Store streaming settings in local variables
accel_stream_settings = dict(
   I2C_ADDR_PRIMARY=0x18,
   NO_OF_BLOCKS = 2,
   REG_X_LSB= [0x12, 0x00],
   NO_OF_DATA_BYTES= [6, 1],
   CHANNEL_ID=1,
   CS_PIN=cpy.MultiIOPin.SHUTTLE_PIN_8.value,
   INT_PIN=cpy.MultiIOPin.SHUTTLE_PIN_21.value,
   INT_TIME_STAMP=1,
)
gyro_stream_settings = dict(
   I2C_ADDR_PRIMARY=0x68,
   NO_OF_BLOCKS = 2,
   REG_X_LSB= [0x02,0x00],
   NO_OF_DATA_BYTES = [6, 1],
   CHANNEL_ID=2,
   CS_PIN=cpy.MultiIOPin.SHUTTLE_PIN_14.value,
   INT_PIN=cpy.MultiIOPin.SHUTTLE_PIN_22.value,
   INT_TIME_STAMP=1,
)


# set the config_streaming parameters
stream_config = cpy.StreamingConfig()
data_blocks = cpy.StreamingBlocks()
if self.interface == cpy.SensorInterface.I2C:
   stream_config.Intf = cpy.SensorInterface.I2C.value
   stream_config.I2CBus = cpy.I2CBus.BUS_I2C_0.value
   stream_config.DevAddr = sensor["I2C_ADDR_PRIMARY"]

elif self.interface == cpy.SensorInterface.SPI:
   stream_config.Intf = cpy.SensorInterface.SPI.value
   stream_config.SPIBus = cpy.SPIBus.BUS_SPI_0.value
   stream_config.CSPin = sensor["CS_PIN"]

if sensor_type == bmi08x.SensorType.ACCEL and self.interface == cpy.SensorInterface.SPI:
   # extra dummy byte for SPI
   dummy_byte_offset = 1
else:
   dummy_byte_offset = 0

data_blocks.NoOfBlocks = sensor["NO_OF_BLOCKS"]
for i in range(0, data_blocks.NoOfBlocks):
   data_blocks.RegStartAddr[i] = sensor["REG_X_LSB"][i]
   data_blocks.NoOfDataBytes[i] = sensor["NO_OF_DATA_BYTES"][i] + dummy_byte_offset

stream_config.IntTimeStamp = sensor["INT_TIME_STAMP"]
stream_config.IntPin = sensor["INT_PIN"]
```

```
# call config_streaming API for each sensor to configure the streaming settings
ret = coinesboard.config_streaming(
   accel_sensor_id, self.accel_stream_config, self.accel_data_blocks)
ret = coinesboard.config_streaming(
   gyro_sensor_id, self.accel_stream_config, self.accel_data_blocks)
```

#### 16.4.4.2 start_stop_streaming

Starts or stops sensor data streaming.

```
coinesboard.start_stop_streaming(stream_mode: StreamingMode, start_stop:
    StreamingState) -> ErrorCodes
```

For the definition of `StreamingMode`, refer to 16.4.6.14. For the definition of `StreamingState`, refer to 16.4.6.16.

#### 16.4.4.3 read_stream_sensor_data

Reads the data streamed from the sensor.

```
coinesboard.read_stream_sensor_data(sensor_id: int, number_of_samples: int,
   buffer_size=STREAM_RSP_BUF_SIZE) -> Tuple[ErrorCodes, list, int]
```

Return:
Tuple of ErrorCodes, data and valid_samples_count
For the detailed definition of parameters, refer to 16.3.4.3.

### 16.4.5 coinespy API calls: Other useful APIs

#### 16.4.5.1 shuttle_eeprom_write

Writes data to the shuttle EEPROM.

```
coinesboard.shuttle_eeprom_write(start_addr: int, data: List[int]) -> ErrorCodes
```

Arguments:

- `start_addr`: Starting address in the EEPROM.
- `data`: List of data bytes to write.

#### 16.4.5.2 shuttle_eeprom_read

Reads data from the shuttle EEPROM.

```
data = coinesboard.shuttle_eeprom_read(start_addr: int, length: int)
```

Arguments:

- `start_addr`: Starting address in the EEPROM.
- `length`: Number of bytes to read.

Return:

- `data`: List of data bytes read.

#### 16.4.5.3 flush_interface

Flush the write buffer.

```
coinesboard.flush_interface()
```

#### 16.4.5.4 soft_reset

Resets the device.

```
coinesboard.soft_reset()
```

#### 16.4.5.5 get_millis

Returns the number of milliseconds passed since the program started.

```
milli_sec = coinesboard.get_millis()
```

#### 16.4.5.6 get_micro_sec

Returns the number of microseconds passed since the program started.

```
micro_sec = coinesboard.get_micro_sec()
```

#### 16.4.5.7 get_coines_error_str

Returns a descriptive error string corresponding to the given error code.

```
error_str = coinesboard.get_coines_error_str(error_code: ErrorCodes) -> str
```

Arguments:

- `error_code`: The error code to be converted to a string.

### 16.4.6 Definition of constants

#### 16.4.6.1 PinDirection

Pin mode definitions

```
class PinDirection:
    INPUT = 0 # COINES_PIN_DIRECTION_IN = 0
    OUTPUT = 1
```

#### 16.4.6.2 PinValue

Pin level definitions

```
class PinValue:
    LOW = 0 # COINES_PIN_VALUE_LOW = 0
    HIGH = 1
```

#### 16.4.6.3 CommInterface

Definition of the Communication interface

```
class CommInterface:
    USB = 0
    SERIAL = 1
    BLE = 2
```

#### 16.4.6.4 I2CMode

Definition of the I2C bus speed

```
class I2CMode:
   STANDARD_MODE = 0 # Standard mode - 100kHz
   FAST_MODE = 1 # Fast mode - 400kHz
   SPEED_3_4_MHZ = 2 # High Speed mode - 3.4 MHz
   SPEED_1_7_MHZ = 3 # High Speed mode 2 - 1.7 MHz
```

#### 16.4.6.5 SPISpeed

Definition of the SPI bus speed

```
class SPISpeed:
   SPI_10_MHZ = 6
   SPI_7_5_MHZ = 8
   SPI_6_MHZ = 10
   SPI_5_MHZ = 12
   SPI_3_75_MHZ = 16
   SPI_3_MHZ = 20
   SPI_2_5_MHZ = 24
   SPI_2_MHZ = 30
   SPI_1_5_MHZ = 40
   SPI_1_25_MHZ = 48
   SPI_1_2_MHZ = 50
   SPI_1_MHZ = 60
   SPI_750_KHZ = 80
   SPI_600_KHZ = 100
   SPI_500_KHZ = 120
   SPI_400_KHZ = 150
   SPI_300_KHZ = 200
   SPI_250_KHZ = 240
```

#### 16.4.6.6 SPITransferBits

Definition of the SPI bits

```
class SPITransferBits:
   SPI8BIT = 8 # 8 bit register read/write
   SPI16BIT = 16 # 16 bit register read/write
```

#### 16.4.6.7 I2CTransferBits

Definition of the I2C bits

```
class I2CTransferBits:
   I2C8BIT = 8 # 8 bit register read/write
   I2C16BIT = 16 # 16 bit register read/write
```

#### 16.4.6.8 SPIMode

Definition of the SPI mode

```
class SPIMode:
   MODE0 = 0x00 # SPI Mode 0: CPOL=0; CPHA=0
   MODE1 = 0x01 # SPI Mode 1: CPOL=0; CPHA=1
   MODE2 = 0x02 # SPI Mode 2: CPOL=1; CPHA=0
   MODE3 = 0x03 # SPI Mode 3: CPOL=1; CPHA=1
```

#### 16.4.6.9 MultiIOPin

Definition of the shuttle board pin(s)

```
class MultiIOPin(Enum):
   SHUTTLE_PIN_7 = 0x09 # CS pin
   SHUTTLE_PIN_8 = 0x05 # Multi-IO 5
   SHUTTLE_PIN_9 = 0x00 # Multi-IO 0
   SHUTTLE_PIN_14 = 0x01 # Multi-IO 1
   SHUTTLE_PIN_15 = 0x02 # Multi-IO 2
   SHUTTLE_PIN_16 = 0x03 # Multi-IO 3
   SHUTTLE_PIN_19 = 0x08 # Multi-IO 8
   SHUTTLE_PIN_20 = 0x06 # Multi-IO 6
   SHUTTLE_PIN_21 = 0x07 # Multi-IO 7
   SHUTTLE_PIN_22 = 0x04 # Multi-IO 4
   SHUTTLE_PIN_SDO = 0x1F

   # APP3.x pins
   MINI_SHUTTLE_PIN_1_4 = 0x10 # GPIO0
   MINI_SHUTTLE_PIN_1_5 = 0x11 # GPIO1
   MINI_SHUTTLE_PIN_1_6 = 0x12 # GPIO2/INT1
   MINI_SHUTTLE_PIN_1_7 = 0x13 # GPIO3/INT2
   MINI_SHUTTLE_PIN_2_5 = 0x14 # GPIO4
   MINI_SHUTTLE_PIN_2_6 = 0x15 # GPIO5
   MINI_SHUTTLE_PIN_2_1 = 0x16 # CS
   MINI_SHUTTLE_PIN_2_3 = 0x17 # SDO
   MINI_SHUTTLE_PIN_2_7 = 0x1D # GPIO6
```

```
   MINI_SHUTTLE_PIN_2_8 = 0x1E # GPIO7
```

#### 16.4.6.10 SensorInterface

Used to define the sensor interface

```
class SensorInterface(Enum):
   SPI = 0
   I2C = 1
```

#### 16.4.6.11 I2CBus

Used to define the I2C type

```
   class I2CBus(Enum):
   BUS_I2C_0 = 0
   BUS_I2C_1 = 1
   BUS_I2C_MAX = 2
```

#### 16.4.6.12 SPIBus

Used to define the SPI type.

```
   class SPIBus(Enum):
   BUS_SPI_0 = 0
   BUS_SPI_1 = 1
   BUS_SPI_MAX = 2
```

#### 16.4.6.13 PinInterruptMode

Used to define pin interrupt modes

```
class PinInterruptMode(Enum):
   # Trigger interrupt on pin state change
   PIN_INTERRUPT_CHANGE = 0
   # Trigger interrupt when pin changes from low to high
   PIN_INTERRUPT_RISING_EDGE = 1
   # Trigger interrupt when pin changes from high to low
   PIN_INTERRUPT_FALLING_EDGE = 2
   PIN_INTERRUPT_MODE_MAXIMUM = 4
```

#### 16.4.6.14 StreamingMode

Streaming mode definitions

```
class StreamingMode:
    STREAMING_MODE_POLLING = 0
    STREAMING_MODE_INTERRUPT = 1
    STREAMING_MODE_DMA_INTERRUPT = 2
    STREAMING_MODE_BLOCK_IO_POLLING = 3
    STREAMING_MODE_BLOCK_IO_INTERRUPT = 4
```

#### 16.4.6.15 BlockType

Block type definitions

```
class BlockType:
    READ_BLOCK = 0
    WRITE_BLOCK = 1
```

#### 16.4.6.16 StreamingState

Streaming state definitions

```
class StreamingState:
   STREAMING_START = 1
   STREAMING_STOP = 0
```

#### 16.4.6.17 SamplingUnits

Sampling Unit definitions

```
class SamplingUnits:
    SAMPLING_TIME_IN_MICRO_SEC = 0x01 # sampling unit in micro second
    SAMPLING_TIME_IN_MILLI_SEC = 0x02 # sampling unit in milli second
```

### 16.5 Error Codes

Error codes are not always returned by the different function calls. Internally, an `error_code` variable is maintained and updated after the function call. It can be read out and checked by the user afterwards.

C Example

```c
#include <stdio.h>
#include <stdlib.h>

#include "coines.h"

int main(void)
{
    int16_t error_code = coines_open_comm_intf(COINES_COMM_INTF_USB, NULL);
    if (error_code != COINES_SUCCESS)
    {
        const char *err_str = get_coines_error_str(error_code);
        printf("\n%s", err_str);
        exit(error_code);
    }

    coines_close_comm_intf(COINES_COMM_INTF_USB, NULL);
    return 0;
}
```

Python Example

```python
import coinespy as cpy
board = cpy.CoinesBoard()
try:
   board.open_comm_interface(cpy.CommInterface.USB)
   board.close_comm_interface()
except:
   print(f'Could not connect to board: {board.error_code}')
   exit(board.error_code)
```

Error code definitions

```
    COINES_SUCCESS = 0
    COINES_E_FAILURE = -1
    COINES_E_COMM_IO_ERROR = -2
    COINES_E_COMM_INIT_FAILED = -3
    COINES_E_UNABLE_OPEN_DEVICE = -4
    COINES_E_DEVICE_NOT_FOUND = -5
    COINES_E_UNABLE_CLAIM_INTERFACE = -6
    COINES_E_MEMORY_ALLOCATION = -7
    COINES_E_NOT_SUPPORTED = -8
    COINES_E_NULL_PTR = -9
    COINES_E_COMM_WRONG_RESPONSE = -10
    COINES_E_SPI16BIT_NOT_CONFIGURED = -11
    COINES_E_SPI_INVALID_BUS_INTERFACE = -12
    COINES_E_SPI_CONFIG_EXIST = -13
    COINES_E_SPI_BUS_NOT_ENABLED = -14
    COINES_E_SPI_CONFIG_FAILED = -15
    COINES_E_I2C_INVALID_BUS_INTERFACE = -16
    COINES_E_I2C_BUS_NOT_ENABLED = -17
    COINES_E_I2C_CONFIG_FAILED = -18
```

```
COINES_E_I2C_CONFIG_EXIST = -19
COINES_E_TIMER_INIT_FAILED = -20
COINES_E_TIMER_INVALID_INSTANCE = -21
COINES_E_TIMER_CC_CHANNEL_NOT_AVAILABLE = -22
COINES_E_EEPROM_RESET_FAILED = -23
COINES_E_EEPROM_READ_FAILED = -24
COINES_E_INIT_FAILED = -25
COINES_E_STREAM_NOT_CONFIGURED = -26
COINES_E_STREAM_INVALID_BLOCK_SIZE = -27
COINES_E_STREAM_SENSOR_ALREADY_CONFIGURED = -28
COINES_E_STREAM_CONFIG_MEMORY_FULL = -29
COINES_E_INVALID_PAYLOAD_LEN = -30
COINES_E_CHANNEL_ALLOCATION_FAILED = -31
COINES_E_CHANNEL_DE_ALLOCATION_FAILED = -32
COINES_E_CHANNEL_ASSIGN_FAILED = -33
COINES_E_CHANNEL_ENABLE_FAILED = -34
COINES_E_CHANNEL_DISABLE_FAILED = -35
COINES_E_INVALID_PIN_NUMBER = -36
COINES_E_MAX_SENSOR_COUNT_REACHED = -37
COINES_E_EEPROM_WRITE_FAILED = -38
COINES_E_INVALID_EEPROM_RW_LENGTH = -39
COINES_E_SCOM_INVALID_CONFIG = -40
COINES_E_BLE_INVALID_CONFIG = -41
COINES_E_SCOM_PORT_IN_USE = -42
COINES_E_UART_INIT_FAILED = -43
COINES_E_UART_WRITE_FAILED = -44
COINES_E_UART_INSTANCE_NOT_SUPPORT = -45
COINES_E_BLE_ADAPTOR_NOT_FOUND = -46
COINES_E_BLE_ADAPTER_BLUETOOTH_NOT_ENABLED = -47
COINES_E_BLE_PERIPHERAL_NOT_FOUND = -48
COINES_E_BLE_LIBRARY_NOT_LOADED = -49
COINES_E_BLE_APP_BOARD_NOT_FOUND = -50
COINES_E_BLE_COMM_FAILED = -51
COINES_E_INCOMPATIBLE_FIRMWARE = -52
COINES_E_UNDEFINED_CODE = -100
```

## 16.6 COINES SDK structure

- coines-api - Contains the source code for low-level interface to Bosch Sensortec's Engineering Boards
- doc - Contains the COINES SDK user manual
- driver - Contains the USB driver for Application Boards
- examples - Contains C and python examples
- installer_scripts - Contains the Windows batch files used internally for install and uninstall functionalities.
- libraries and thirdparty - Contains libraries and SDKs used for communication APIs.
- tools - Contains tools for the Application switch, firmware update, and BLE connect

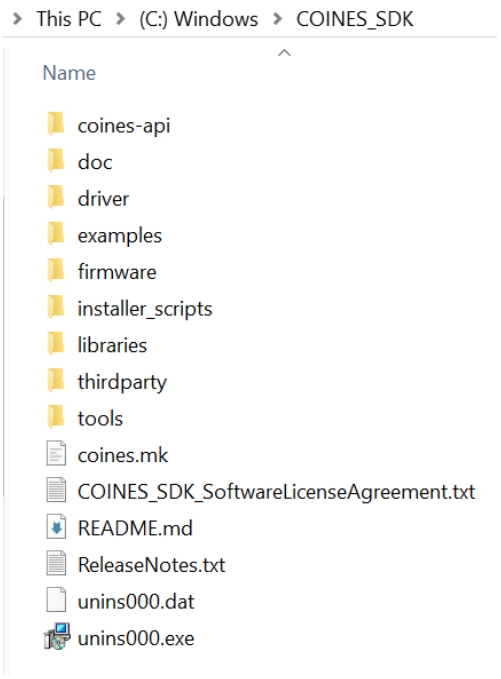



Fig. 26: COINES SDK file structure

## 16.7 Running BSEC on COINES SDK

### Preparation

- Download the BSEC package from the Bosch Sensortec website: https://www.bosch-sensortec.com/software-tools/software/bme688-and-bme690-software/

### Integration Steps

- After installing the COINES SDK, navigate to the `examples` directory: `C:\COINES_SDK\<version>\examples`
- Since there is no default project for BME sensors, create a new folder named `bme69x`.

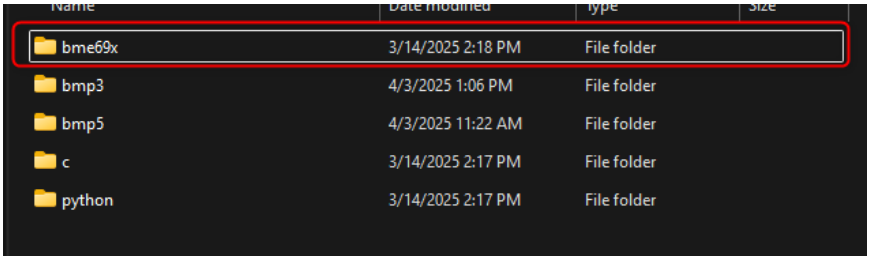



Fig. 27: Creating a BME69x project folder

- Inside the `bme69x` folder, create a subfolder called `examples` and place the required API files in it. Obtain the API files from one of the following sources:
  - From BSEC package: `bsec_v3-2-1-0\examples\BSEC_Integration_Examples\src\bme69x`
  - From GitHub repository: https://github.com/boschsensortec/BME690_SensorAPI

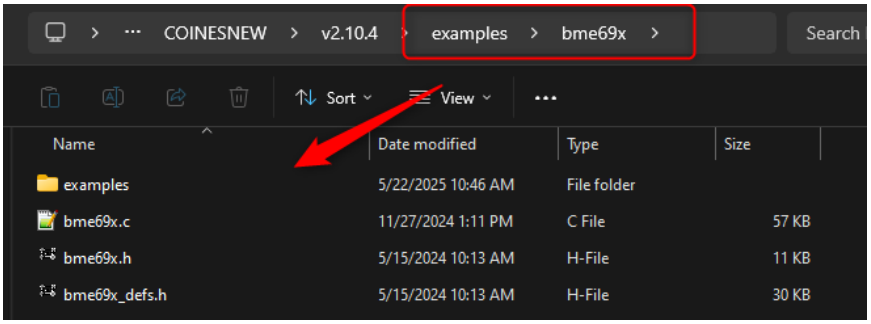



Fig. 28: Creating an examples subfolder inside the bme69x directory

- Inside the new `examples` directory, create another folder named `BSEC3210`.

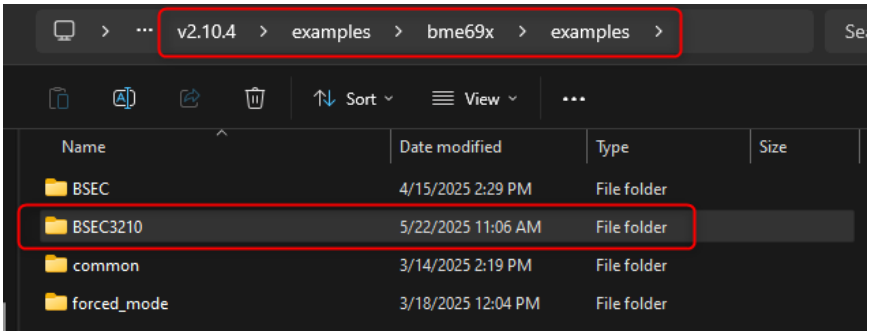



Fig. 29: Creating the BSEC3210 folder inside the examples directory

- Inside `BSEC3210` directory, Copy the relevant project files from: `bsec_v3-2-1-0\examples\BSEC_Integration_Examples`

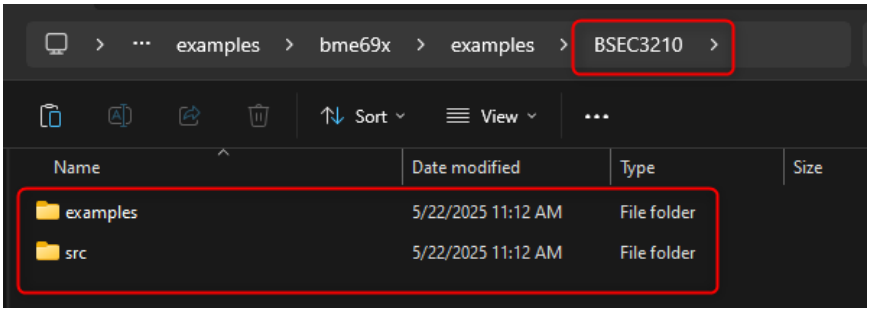



Fig. 30: Example files required for BSEC integration

- Navigate to an example directory containing the `Makefile`.

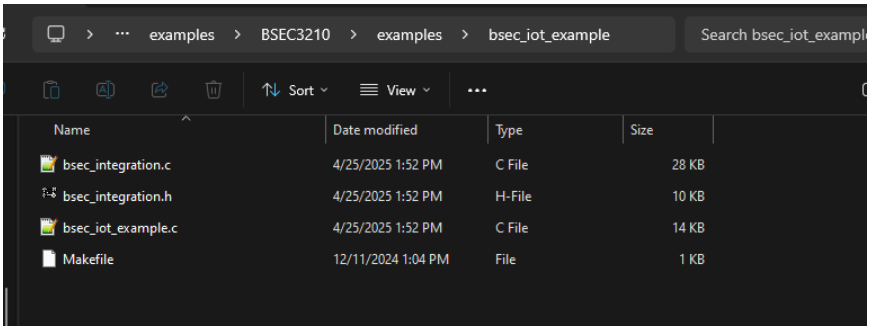



Fig. 31: Navigating to the example directory with the Makefile

- Modify the `Makefile` if needed, based on your COINES SDK version and installation path.

```
COINES_INSTALL_PATH ?= C:/COINESNEW/v2.10.4/

EXAMPLE_FILE ?= bsec_iot_example.c

EXAMPLE_CODE_SRC = ../../src

C_SRCS += bsec_integration.c
C_SRCS += $(EXAMPLE_CODE_SRC)/bme69x/bme69x.c
C_SRCS += $(EXAMPLE_CODE_SRC)/config/bsec_selectivity.c

INCLUDEPATHS += $(EXAMPLE_CODE_SRC)/bme69x
INCLUDEPATHS += $(EXAMPLE_CODE_SRC)/config
INCLUDEPATHS += $(EXAMPLE_CODE_SRC)/inc

# Additional paths to look for library files
LIBPATHS += $(EXAMPLE_CODE_SRC)/cortex-m4/fpv4-sp-d16-hard

# Additional static/shared libraries
LIBS += algobsec

LOCATION?=FLASH
TARGET?=MCU_APP31

include $(COINES_INSTALL_PATH)/coines.mk
```

Fig. 32: Modifying the Makefile for BSEC integration

- Configure settings in `bsec_integration.h`:
  - Set `NUM_OF_SENS` to 1 (for a single BME690 shuttle).
  - Choose your preferred `OUTPUT_MODE`.
  - Set `SAMPLE_RATE`; e.g., use `BSEC_SAMPLE_RATE_LP` in IAQ mode to output data every 3 seconds.

```
#include <stdbool.h>
#include "bme69x.h"
#include "bsec_interface.h"
#include "bsec_datatypes.h"

#define NUM_OF_SENS                     UINT8_C(8)
#define BSEC_INSTANCE_SIZE              UINT16_C(3272)
#define BSEC_CHECK_INPUT(x, shift)      (x & (1 << (shift-1)))
#define BSEC_TOTAL_HEAT_DUR             UINT16_C(140)

/* Note :
          For the classification output from BSEC algorithm set OUTPUT_MODE to CLASS
          For the regression output from BSEC algorithm set OUTPUT_MODE to REGRESSIO
          For the LP, ULP, CONT output from BSEC algorithm set OUTPUT_MODE to IAQ
*/
#define CLASSIFICATION  1
#define REGRESSION      2
#define IAQ             3
#define OUTPUT_MODE     IAQ

/* Note :
        Set the appropriate "SAMPLE_RATE" based on configured "OUTPUT_MODE".
            * If the "OUTPUT_MODE"  value is "CLASSIFICATION" (or) "REGRESSION", the
            * If the "OUTPUT_MODE" value is "IAQ", the "SAMPLE_RATE" assigned is "BS
            * For the "OUTPUT_MODE" as "IAQ" the other supported "SAMPLE_RATE" is "B
*/
#if (OUTPUT_MODE == CLASSIFICATION || OUTPUT_MODE == REGRESSION)
#define SAMPLE_RATE         BSEC_SAMPLE_RATE_SCAN
#define NUM_USED_OUTPUTS    UINT8_C(9)
#elif (OUTPUT_MODE == IAQ)
#define SAMPLE_RATE         BSEC_SAMPLE_RATE_ULP
#define NUM_USED_OUTPUTS    UINT8_C(14)
#endif
```

Fig. 33: Adjusting configuration in `bsec_integration.h`

### Compiling and Flashing the Firmware

- Connect the APP3.1 board to your PC via USB, and attach the BME690 shuttle board.
- Open a terminal window in the directory with makefile.

- Compile and flash the firmware using the command: `mingw32-make TARGET=MCU_APP31 download`

```
C:\COINESNEW\v2.10.4\examples\bme69x\examples\BSEC3210\examples\bsec_iot_example>mingw32-make TARGET=MCU_APP31 download
```

Fig. 34: Compiling and flashing firmware

- If successful, confirmation message will be printed:

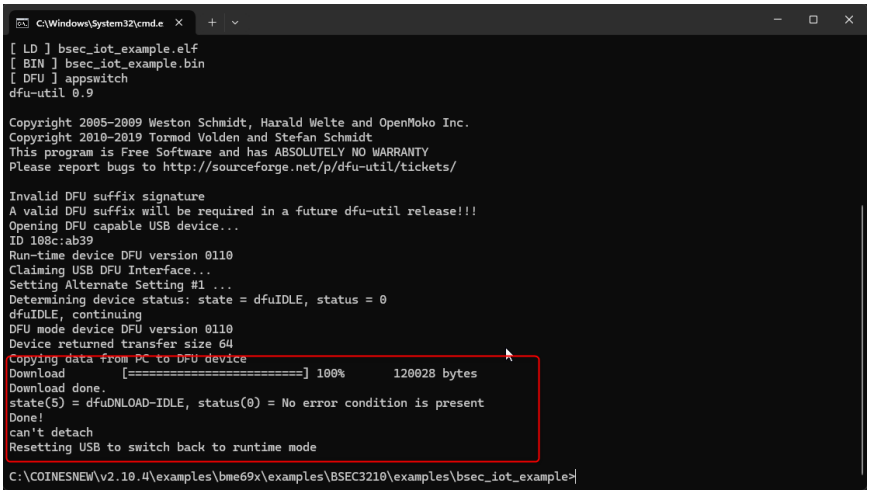



Fig. 35: Firmware flash success message

- Launch a serial port terminal to view the BME690's real-time output.

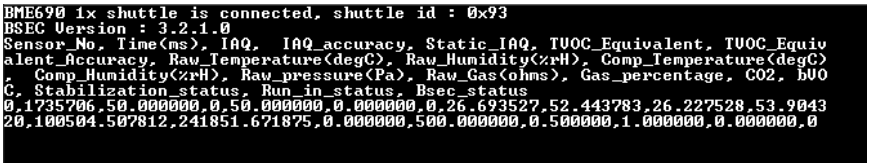



Fig. 36: Output from the BME690 sensor via serial port

Bosch Sensortec GmbH
Gerhard-Kindler-Straße 9
72770 Reutlingen / Germany

www.bosch-sensortec.com


Document number: BST-DHW-AN013
Revision 2.1