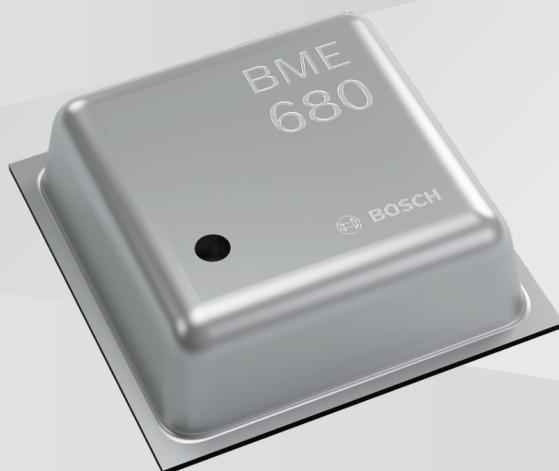




BOSCH

Integration Guide

Bosch Software Environmental Cluster (BSEC)



Contents

1	BSEC Integration Guideline	5
1.1	Overview of BME Family Sensors	5
1.2	The Environmental Fusion Library BSEC	6
1.2.1	BSEC Library Solutions	6
1.2.2	Key Features	6
1.2.3	Supported Virtual Sensor Output Signals	7
1.3	Requirements for Integration	8
1.3.1	Hardware	8
1.3.2	Software Framework	8
1.3.3	Physical Input Sensor Signals	8
1.3.4	Build the Solution	8
2	BSEC Step-by-step Example	9
2.1	Prerequisites	9
2.2	Setting Everything Up	9
2.3	The Example Code	10
2.4	Hello "Indoor-Air-Quality"	10
2.5	Reducing power consumption	13
2.6	Temperature offsets due to heatsources on the board	13
2.7	Conclusion	14
3	FAQ	15
3.1	No Output From bsec_do_steps()	15
3.2	IAQ output does not change or accuracy remains 0	15
3.3	Error Codes and Corrective Measures	15
3.3.1	Errors Returned by bsec_do_steps()	15
3.3.2	Errors Returned by bsec_update_subscription()	17
3.3.3	Errors Returned by bsec_set_configuration() / bsec_set_state()	20
3.3.4	Errors Returned by bsec_sensor_control()	22
4	Module Documentation	23
4.1	BSEC C Interface	23
4.1.1	Interface Usage	23
4.1.2	Interface Functions	24
4.1.3	Enumerations	33
4.1.4	Defines	38
5	Data Structure Documentation	41
5.1	bsec_bme_settings_t Struct Reference	41
5.1.1	Detailed Description	41
5.1.2	Field Documentation	42
5.2	bsec_input_t Struct Reference	43
5.2.1	Detailed Description	43

- 5.2.2 Field Documentation 43
- 5.3 bsec_output_t Struct Reference 44
 - 5.3.1 Detailed Description 45
 - 5.3.2 Field Documentation 45
- 5.4 bsec_sensor_configuration_t Struct Reference 46
 - 5.4.1 Detailed Description 46
 - 5.4.2 Field Documentation 47
- 5.5 bsec_version_t Struct Reference 47
 - 5.5.1 Detailed Description 47
 - 5.5.2 Field Documentation 48

1 BSEC Integration Guideline

1.1 Overview of BME Family Sensors

The BME sensor family has been designed to enable pressure, temperature, humidity and gas measurements (BME680 only). The sensors can be operated in different modes specified in supplied header files. For example, ULP mode offers output data at slow rate thereby minimizing power consumption. In general, higher data rate corresponds to higher power consumption.

This section will provide information about the integrated sensors which are used by the BSEC library and also a brief overview of them.

Temperature Sensor

In order to guarantee fast response times, the temperature sensor within BME280/680 is expected to be mounted at a location in the device that enables good air and temperature exchange. The integrated temperature sensor has been optimized for very low noise and high resolution. It is primarily used for estimating ambient temperature and for temperature compensation of the other sensors present. The temperature measurement accuracy is specified in the corresponding data sheet of the used hardware.

Pressure Sensor

The pressure sensor within BME280/680 is an absolute barometric pressure sensor featuring exceptionally high accuracy and resolution at very low noise. The pressure measurement accuracy is specified in the corresponding data sheet of the used hardware.

Relative Humidity Sensor

The humidity sensor within BME280/680 measures relative humidity from 0 to 100 percent across a temperature range from -40 degrees centigrade to +85 degrees centigrade. The humidity measurement accuracy is specified in the corresponding data sheet of the used hardware.

Gas Sensor

The gas sensor within BME680 can detect a broad range of gases to measure indoor air quality for personal well being. Gases that can be detected by the BME680 include volatile organic compounds (VOC) from paints (such as formaldehyde), lacquers, paint strippers, cleaning supplies, furnishings, office equipment, glues, adhesives and alcohol. The gas measurement accuracy is specified in the corresponding data sheet of the used hardware.

1.2 The Environmental Fusion Library BSEC

General Description

BSEC fusion library has been conceptualized to provide higher-level signal processing and fusion for the BME sensor. The library receives compensated sensor values from the sensor API. It processes the BME sensor signals (in combination with the additional optional device sensors) to provide the requested sensor outputs. Inputs to BSEC signals are commonly called signals from *physical sensors*. For the outputs of BSEC, several denominations are coined for the name of the sensors providing the respective signal: composite sensors, synthetic sensors, software-based sensors and virtual sensors. For BSEC, only the denomination *virtual sensors* shall be used.

Prior to probing into BSEC Library, the entire BSEC system can be understood as a combination of the below mentioned system architecture components

- ▶ BME680 sensor (pressure, temperature, humidity and gas) / BME280 sensor (pressure, temperature, humidity)
- ▶ Device with BME680/280 integrated
- ▶ Sensor driver API
- ▶ BSEC fusion library
- ▶ *Optional*: Additional device sensors (i.e., temperature of other heat sources in the device or position sensors)

1.2.1 BSEC Library Solutions

A BSEC solution can be chosen from a set of pre-defined and tested solutions that have a fixed set of features. Based on customer requests it is technically possible to further customize BSEC to meet specific customer demands.

Available BSEC solutions are

Solution	Included features
ALL	Indoor-air-quality, ambient temperature/humidity estimation, raw signals
IAQ	Indoor-air-quality, sensor heating compensated temperature/humidity, raw signals
BSH	Ambient temperature/humidity estimation, raw signals

1.2.2 Key Features

- ▶ Precise calculation of ambient air temperature outside the device
- ▶ Precise calculation of ambient relative humidity outside the device
- ▶ Precise calculation of atmospheric pressure outside the device
- ▶ Precise calculation of indoor air quality (IAQ) level outside the device

Applications

- ▶ Health monitoring/ well-being (warning regarding dehydration / heat stroke)
- ▶ Home automation control

- ▶ Control heating, venting, air conditioning (HVAC) applications
- ▶ Gaming applications like flying toys
- ▶ Internet of things
- ▶ Context awareness
- ▶ The pressure sensor provides the following features
 - ▶ Enhancement of GPS navigation (e.g., time-to-first-fix improvement, dead-reckoning, slope detection)
 - ▶ Indoor navigation (floor detection, elevator detection)
 - ▶ Outdoor navigation, leisure and sports applications
 - ▶ Weather forecast
 - ▶ Health care applications (e.g., spirometry)
 - ▶ Vertical velocity indication (e.g., rise/sink speed)

Advantages

- ▶ Hardware and software co-design for optimal performance
- ▶ Complete software fusion solution
- ▶ Eliminates need for developing fusion software in customer's side
- ▶ Robust virtual sensor outputs optimized for the application

1.2.3 Supported Virtual Sensor Output Signals

BSEC provides the output signals given in the table below. All signals from virtual sensor outputs are time-continuous signals sampled in equidistant time intervals.

Signal name	Unit	Acc.? ¹	Inc.? ²	ULP mode	LP mode
BSEC_OUTPUT_RAW_PRESSURE	Pa	no	ALL, IAQ, BSH	1/300 [Hz]	1/3 [Hz]
BSEC_OUTPUT_RAW_TEMPERATURE	deg C	no	ALL, IAQ, BSH	1/300 [Hz]	1/3 [Hz]
BSEC_OUTPUT_RAW_HUMIDITY	%	no	ALL, IAQ, BSH	1/300 [Hz]	1/3 [Hz]
BSEC_OUTPUT_IAQ_ESTIMATE	0-500	yes	ALL, IAQ	1/300 [Hz]	1/3 [Hz]
BSEC_OUTPUT_SENSOR_HEAT_COMP↔ ENSATED_TEMPERATURE	deg C	no	ALL, IAQ	1/300 [Hz]	1/3 [Hz]
BSEC_OUTPUT_SENSOR_HEAT_COMP↔ ENSATED_HUMIDITY	%	no	ALL, IAQ	1/300 [Hz]	1/3 [Hz]
::BSEC_OUTPUT_AMBIENT_TEMPERAT↔ URE	deg C	yes	ALL, BSH	1/300 [Hz]	1/3 [Hz]
::BSEC_OUTPUT_AMBIENT_HUMIDITY	%	no	ALL, BSH	1/300 [Hz]	1/3 [Hz]
BSEC_OUTPUT_STABILIZATION_STATUS	y/n	no	ALL, IAQ	1/300 [Hz]	1/3 [Hz]
BSEC_OUTPUT_RUN_IN_STATUS	y/n	no	ALL, IAQ	1/300 [Hz]	1/3 [Hz]

¹ Accuracy status available (see [bsec_output_t::accuracy](#)). ² Included in solution.

1.3 Requirements for Integration

1.3.1 Hardware

BSEC was specifically designed to work together with Bosch environmental sensor of the BMExxx family. No other sensors are supported. To ensure a consistent performance, the sensors shall be configured by BSEC itself by the use of the `bsec_sensor_control()` interface.

1.3.2 Software Framework

The framework must provide the sample rates requested by the user for the virtual sensors to BSEC via `bsec_update_subscription()`, e.g., using an application on the end-user graphical interface like an Android application. BSEC internally configures itself according to the requested output sample rates. The framework must then use `bsec_sensor_control()` periodically to configure the BMExxx sensor. After every call to `bsec_sensor_control()`, the next call to `bsec_sensor_control()` should be scheduled by the framework as specified in the returned sensor settings structure.

For each input data, an exact time stamp shall be provided synchronized to each other when they belong to the same instant in time, i.e., they are "aligned". The processing function requires at least one input signal.

1.3.3 Physical Input Sensor Signals

BSEC is designed to be used exclusively together with sensors of the BMExxx family, such as the BME280 or the BME680.

Moreover, ambient temperature and humidity estimation may require additional inputs from the host system to compensate for self-heating effects caused by the operation of the host device. This may include information such as supply voltage, charging status or display status.

1.3.4 Build the Solution

BSEC is delivered as a pre-compiled static library to be linked against the host integratoin code. The library includes the following header files which need to be included along with BSEC library package.

Header file	Description
<code>bsec_↔ datatypes.h</code>	Data types and defines used by interface functions
<code>bsec_interface.h</code>	Declaration of interface functions

2 BSEC Step-by-step Example

Temperature, humidity, and the presence of certain gases all influence the quality of the air we are breathing. In this walk-through, we will see how to use Bosch Sensortec BME680 sensors together with the BSEC software package to measure indoor-air-quality (IAQ).

2.1 Prerequisites

First of all, you will need a BME680 sensor that is connected to a microcontroller (MCU). The MCU will be used to control the operation of the sensor and to process the sensor signals to derive indoor-air-quality in the end. Of course, you will also need an development environment for the MCU of your choice. In this example, I will use the Arduino-based [Octopus](#) board with BME680 already included on-board. If you plan on following this tutorial with the same hardware, you can find instructions on how to setup Arduino IDE for this board [here](#). If you have trouble downloading the board support package via the board manager, you can manually get the required files from [GitHub](#) directly.

Once we are set with our hardware and development environment, we need two pieces of software to get the most out of our BME680 sensor:

1. BME680 API ([available on GitHub](#)) deals with the low-level communication and basic compensation of sensor data. It saves us from having to fiddle with individual registers on the BME680 ourselves.
2. BSEC ([available from Bosch Sensortec](#)) will be used by us to control the sensor operation and it will provide us with an indoor-air-quality output as well as compensated temperature and relative humidity data. For this it interfaces with the BME680 API we just downloaded. In case you are wondering, BSEC stands for "Bosch Software Environmental Cluster".

While the above might sound somewhat intimidating, we are lucky that BSEC comes with an ready made example code that only requires a small number of modifications to get it running on a new platform.

2.2 Setting Everything Up

To get started, we will first see which files from the packages we just downloaded need to be added to our project. From the BME680 API, we need to add all included .h and .c files. In case of BSEC, we need to add the BSEC interface headers as well the the example that we want to extend:

- ▶ inc
 - ▶ bsec_interface.h
 - ▶ bsec_datatypes.h
- ▶ examples
 - ▶ bsec_integration.h
 - ▶ bsec_integration.c

► bsec_iot_example.c

As BSEC is made available as a pre-compiled binary, we should also get the correct .a file. Since the Octopus board uses an ESP8266 MCU, we need to use the library file found in `binaries/staticlib/ESP8266/libalgobsec.a` of the BSEC release package.

To use our code in an Arduino sketch, we should copy all the above mentioned files into a folder named `bsec_iot_example`.

2.3 The Example Code

Once we are set up, let's have a look at `bsec_iot_example.c` which is the only file we will have to modify to get our project up and running. You will see that it contains a `main()` function as shown below.

```
int main()
{
    /* Call to the function which initializes the BSEC library
     * Switch on low-power mode and provide no temperature offset */
    bsec_iot_init(BSEC_SAMPLE_RATE_LP, 0.0f, bus_write, bus_read, sleep);

    /* Call to endless loop function which reads and processes data based on sensor settings */
    bsec_iot_loop(sleep, get_timestamp_us, output_ready);

    return 0;
}
```

Here, we first initialize both the API and the BSEC library. For this purpose, we provide 3 function pointers `bus_write`, `bus_read`, and `sleep` to `bsec_iot_init()`. These pointers are used by BSEC and the BME680 API to communicate with the sensor and to put the system to sleep to control timings. Moreover, we provide the desired operation mode, in this case, we use low-power (LP) mode. The numerical argument allows us to subtract a temperature offset from the temperature reading and correct the humidity accordingly. More on this later.

Next, `bsec_iot_loop()` is called to enter an endless loop which periodically reads out sensor data, processes the signals, and calls the provided function pointer `output_ready` whenever new data is available. Additionally, we provide a function pointer `get_timestamp_us` that is used to get the system time stamps in microseconds.

Inside `bsec_iot_example.c`, we already find empty implementations of these five functions pointers. All we have to do to get our basic example up and running is to fill in some code into these functions and add a little bit of MCU initialization to the beginning of `main()`.

2.4 Hello "Indoor-Air-Quality"

Since we want to use the example with Arduino IDE, we will have to convert the example C code into an Arduino-compatible sketch file. For this, we change the file name of `bsec_iot_example.c` to `bsec_iot_example.ino` and rename the `int main()` function into `void setup()`. At the top of the function, we add 2 lines to initialize the I2C port used to talk with our sensor as well as the serial line we want to use to report IAQ values back to our host PC. We also add an empty `loop()` to create a complete Arduino sketch.

```
void setup()
{
    /* Init I2C and serial communication */
    Wire.begin();
    Serial.begin(115200);
}
```

```

/* Call to the function which initializes the BSEC library
 * Switch on low-power mode and provide no temperature offset */
bsec_iot_init(BSEC_SAMPLE_RATE_LP, 0.0f, bus_write, bus_read, sleep);

/* Call to endless loop function which reads and processes data based on sensor settings */
bsec_iot_loop(sleep, get_timestamp_us, output_ready);
}

void loop()
{
    /* We do not need to put anything here as we enter our on loop function in setup() */
}

```

To be able to compile to above code, we also need to add the following include at the top of the file.

```
#include <Wire.h>
```

With the setup done, we can now move to implementing the communication with the sensor. The write function takes an array of bytes as well as a register address to start writing to.

```

int8_t bus_write(uint8_t dev_addr, uint8_t reg_addr, uint8_t *reg_data_ptr, uint8_t data_len)
{
    Wire.beginTransaction(dev_addr);
    Wire.write(reg_addr);    /* Set register address to start writing to */

    /* Write the data */
    for (int index = 0; index < data_len; index++) {
        Wire.write(reg_data_ptr[index]);
    }

    return (int8_t)Wire.endTransmission();
}

```

In case of the read function, we burst read from the BME680 register map.

```

int8_t bus_read(uint8_t dev_addr, uint8_t reg_addr, uint8_t *reg_data_ptr, uint8_t data_len)
{
    int8_t comResult = 0;
    Wire.beginTransaction(dev_addr);
    Wire.write(reg_addr);    /* Set register address to start reading from */
    comResult = Wire.endTransmission();

    delayMicroseconds(150);    /* Precautionary response delay */
    Wire.requestFrom(dev_addr, data_len);    /* Request data */

    int index = 0;
    while (Wire.available()) /* The slave device may send less than requested (burst read) */
    {
        reg_data_ptr[index] = Wire.read();
        index++;
    }

    return comResult;
}

```

To implement sleep functionality, we make use of the Arduino delay() function.

```

void sleep(uint32_t t_ms)
{
    delay(t_ms);
}

```

Getting a timestamp is just as easy. Here, we use the millis() function to get a timestamp in milliseconds and multiply by 1000 to get the required microsecond resolution.

```
int64_t get_timestamp_us()
{
    return (int64_t) millis() * 1000;
}
```

Finally, we need to do something with the measurement data we get. The simplest thing is to print out the something on the serial interface and look at it on your host machine. In this case, we print out the temperature, humidity, and IAQ values returned by BSEC. In the future, we could even return the pressure as it is also measured.

```
void output_ready(int64_t timestamp, float iaq, uint8_t iaq_accuracy, float temperature,
    float pressure, float humidity, bsec_library_return_t bsec_status)
{
    Serial.print("[");
    Serial.print(timestamp/1e6);
    Serial.print("] T: ");
    Serial.print(temperature);
    Serial.print(" | rH: ");
    Serial.print(humidity);
    Serial.print(" | IAQ: ");
    Serial.print(iaq);
    Serial.print(" (");
    Serial.print(iaq_accuracy);
    Serial.println(")");
}
```

The last step we need to do is to ensure that the pre-build libalgobsec.a library is linked when we compile our project. Unfortunately, this process is somewhat tricky when it comes to Arduino IDE. We first need to find where the board support package for our board is installed. On Windows, this could be for example in <USER_HOME>\AppData\Local\Arduino15\packages\esp8266\hardware or in <ARDUINO_ROOT>\hardware. Once we found the location, we need to perform the following steps. Please keep in mind that the target paths might differ slightly depending on the ESP8266 package version you are using.

1. We need to copy the file binaries\staticlib\ESP8266\libalgobsec.a from the BSEC package into the hardware\esp8266\2.3.0\tools\sdk\lib folder.
2. The linker file found at hardware\esp8266\2.3.0\tools\sdk\ld\eagle.app.v6.common.ld needs to be modified by inserting the line *libalgobsec.a:(.literal .text .literal.* .text.*) after the line *libm.a:(.literal .text .literal.* .text.*).
3. Finally, we need to change the linker argument, telling the linker to include BSEC. This is achieved by adding the argument -lalgobsec to the line compiler.c.elf.libs=-lm -lgcc ... found in hardware\esp8266\2.3.0\platform.txt.

If we now run our project and check with the Serial Monitor (found under Tools menu in Arduino IDE), we can see a new measurement come in every 3 seconds as shown below.

```
...
[300345.00] T: 33.30| rH: 22.73| IAQ: 25.00 (3)
[303346.00] T: 33.30| rH: 22.76| IAQ: 26.30 (3)
[306346.00] T: 33.26| rH: 22.81| IAQ: 27.90 (3)
[309346.00] T: 33.26| rH: 22.84| IAQ: 19.72 (3)
[312346.00] T: 33.20| rH: 22.93| IAQ: 25.02 (3)
[315346.00] T: 33.19| rH: 22.94| IAQ: 20.70 (3)
[318346.00] T: 33.14| rH: 22.97| IAQ: 28.80 (3)
...
```

Success!

2.5 Reducing power consumption

You will notice that we now get IAQ, temperature and humidity data once every 3 seconds. This is because the example code is pre-configured to use what is called low-power (LP) mode.

For certain applications, we may want to reduce the power consumption (and data rate) and use ultra-low-power mode. Since it only takes around 0.08 mA current on average, this mode is ideal for long-term battery powered operation. But let's see if we can change the code to lower the data rate.

In order to make this change, we can simply change the first argument we pass to `bsec_iot_init()` to ULP mode:

```
/* Call to the function which initializes the BSEC library
 * Switch on ultra-low-power mode and provide no temperature offset */
bsec_iot_init(BSEC_SAMPLE_RATE_ULP, 0.0f, bus_write, bus_read, sleep);
```

When we run our project now again, we can see the data coming in much slower and with greatly reduced current consumption.

```
...
[1200329.00] T: 30.97 | rH: 25.95 | IAQ: 25.00 (3)
[1500329.00] T: 31.35 | rH: 25.98 | IAQ: 97.96 (3)
[1800330.00] T: 30.86 | rH: 26.72 | IAQ: 131.54 (3)
[2100330.00] T: 30.80 | rH: 26.73 | IAQ: 124.95 (3)
...
```

2.6 Temperature offsets due to heatsources on the board

Last but not least, let's have a look at the temperature and humidity values we are receiving from the board. A temperature of over thirty degrees and such a low relative humidity seems off. Looking at a reference thermometer, we can see that our temperature is indeed a few degrees too high. Does that mean the temperature sensor inside the BME680 is inaccurate?

Actually no, it very accurately measures the temperature exactly where it is located on the board. But there also is the issue: our board as most devices contains some heatsources (e.g., MCU, WiFi chip, display, ...). This means the temperature on our board is actually higher than the ambient temperature. Since the absolute amount of water in the air is approximately constant, this also causes the relative humidity to be lower on our board than elsewhere in the room.

As BSEC cannot know in which kind of device the sensor is integrated, we have to provide some information to the algorithm to enable it to compensate this offset. In the simplest case, we have to deal with an embedded device with a constant workload and approximately constant self-heating. In such a case, we can simply supply a temperature offset to BSEC which will be subtracted from the temperature and will be used to correct the humidity reading as well.

To achieve this, we simply provide a non-zero temperature offset as the second argument to `bsec_iot_init()` as shown below. Here, we subtract a 5 degrees offset, for example.

```
/* Call to the function which initializes the BSEC library
 * Switch on ultra-low-power mode and subtract a 5 degrees temperature offset */
bsec_iot_init(BSEC_SAMPLE_RATE_ULP, 5.0f, bus_write, bus_read, sleep);
```

2.7 Conclusion

As you can see, it is easy to easily BSEC and BME680 API into an Arduino platform to measure indoor-air-quality, temperature, and humidity. We also did some modifications to the existing example code in order to change the sampling rate to ULP and to subtract a temperature offset.

3 FAQ

3.1 No Output From `bsec_do_steps()`

Possible reasons:

- ▶ The virtual sensor was not enabled via `bsec_update_subscription()`
- ▶ The input signals required for that virtual sensor were not provided to `bsec_do_steps()`
- ▶ The timestamps passed to `bsec_do_steps()` are duplicated or are not in nanosecond resolution

3.2 IAQ output does not change or accuracy remains 0

Possible reason:

- ▶ Timing of gas measurements is not within 50% of the target values. For example, when running the sensor in low-power mode the intended sample period is 3 s. In this case the difference between two consecutive measurements must not exceed 150% of 3 s which is 4.5 s. When integrating BSEC, it is crucial to strictly follow the timing between measurements as returned by `bsec_sensor_control()` in the `bsec_sensor_settings_t::next_call` field.

Correction method:

- ▶ Ensure accurate timestamps with ns resolution, especially avoid overflows of the timer or issues with 64-bit arithmetic
- ▶ Ensure that the `bsec_sensor_control()` and `bsec_do_steps()` loop is correctly implemented and the `bsec_sensor_settings_t::next_call` field is used to determine the frequency between measurements.

3.3 Error Codes and Corrective Measures

3.3.1 Errors Returned by `bsec_do_steps()`

3.3.1.1 BSEC_E_DOSTEPS_INVALIDINPUT (-1)

Possible reasons:

- ▶ The sensor id of the input (physical) sensor passed to `bsec_do_steps()` is not in valid range or not valid for the requested output (virtual) sensor.
- ▶ The number of inputs passed to `bsec_do_steps()` is greater than the actual number of populated input structs.

E.g:

```
inputs[0].sensor_id = 100;
inputs[0].signal = 25;
inputs[0].time_stamp= ts;
n_inputs = 3;
status = bsec_do_steps(inputs, n_inputs, outputs, &n_outputs);
```

Correction methods:

- ▶ The sensor_id field in the inputs structure passed to `bsec_do_steps()` should be one among the input (physical) sensors ids returned from `bsec_update_subscription()` stored in `required_sensor_settings` array.
- ▶ The sensor_id field in the inputs structure passed to `bsec_do_steps()` should be in the range of `bsec_physical_sensor_t` enum.
- ▶ `n_inputs` should be equal to the number of inputs passed to `bsec_do_steps()`, i.e. size of inputs structure array.

3.3.1.2 BSEC_E_DOSTEPS_VALUELIMITS (-2)

Possible reasons:

- ▶ The value of the input (physical) sensor signal passed to `bsec_do_steps()` is not in the valid range.

E.g:

```
inputs[0].sensor_id = BSEC_INPUT_TEMPERATURE;
inputs[0].signal = 250;
inputs[0].time_stamp= ts;
n_inputs = 1;
status = bsec_do_steps(inputs, n_inputs, outputs, &n_outputs);
```

Correction methods:

- ▶ The value of signal field in the inputs structure passed to `bsec_do_steps()` should be in a valid range. The allowed input value range for the sensors is listed below.
 - ▶ TEMPERATURE (-65 to 125)
 - ▶ HUMIDITY (0 to 100)
 - ▶ PRESSURE (0 to 2000000)
 - ▶ GASRESISTOR (170 to 13200000)
 - ▶ Other Sensors (-3.4028e+38 to +3.4028e+38)

3.3.1.3 BSEC_E_DOSTEPS_DUPLICATEINPUT (-6)

Possible reasons:

- ▶ Duplicate input (physical) sensor ids are passed to `bsec_do_steps()`.

E.g:

```
inputs[0].sensor_id = BSEC_INPUT_TEMPERATURE;
inputs[0].signal = 25;
inputs[0].time_stamp= ts;
inputs[1].sensor_id = BSEC_INPUT_TEMPERATURE;
inputs[1].signal = 30;
inputs[1].time_stamp= ts;
n_inputs = 2;
status = bsec_do_steps(inputs, n_inputs, outputs, &n_outputs);
```

Correction methods:

- ▶ Each input (physical) sensor id passed to `bsec_do_steps()` should be unique.

3.3.1.4 BSEC_I_DOSTEPS_NOOUTPUTSRETURNABLE (2)

Possible reasons:

- ▶ Some virtual sensors are requested, but no memory is allocated to hold the returned output values corresponding to these virtual sensors from `bsec_do_steps()`.

E.g:

```
n_outputs=0; /*Requested number of virtual sensor is 5*/  
status = bsec_do_steps(inputs, n_inputs, outputs, &n_outputs);
```

Correction methods:

- ▶ `n_outputs` should be assigned the value equal to the maximum number of virtual sensors defined in `bsec_virtual_sensor_t` enum.

3.3.1.5 BSEC_W_DOSTEPS_EXCESSOUTPUTS (3)

Possible reasons:

- ▶ Some virtual sensors are requested, but not enough memory is allocated to hold the returned output values corresponding to these virtual sensors from `bsec_do_steps()`.

E.g:

```
n_outputs = 2 ; /*Requested number of virtual sensor is 5*/  
status=bsec_do_steps(inputs, n_inputs, outputs, &n_outputs);
```

Correction methods:

- ▶ `n_outputs` should be assigned the value equal to the maximum number of virtual sensors defined in `bsec_virtual_sensor_t` enum.

3.3.1.6 BSEC_W_DOSTEPS_TSINTRADIFFOUTOFRANGE (4)

Possible reasons:

- ▶ Current timestamp of the inputs passed to `bsec_do_steps()` is same as the previous one stored for the same inputs.

Correction methods:

- ▶ `time_stamp` field of the inputs structure passed to `bsec_do_steps()` should be unique.

3.3.2 Errors Returned by `bsec_update_subscription()`

3.3.2.1 BSEC_E_SU_WRONGDATARATE (-10)

Possible reasons:

- ▶ Virtual sensors are requested with a sampling rate which is not allowed, e.g. 0.

E.g:

```
requested_virtual_sensors[virtual_sensor_count].sample_rate = 0;
requested_virtual_sensors[virtual_sensor_count].sensor_id = BSEC_OUTPUT_RAW_GAS;
status = bsec_update_subscription(requested_virtual_sensors,
    n_requested_virtual_sensors, required_sensor_settings, &n_required_sensor_settings);
```

Correction methods:

- ▶ The `sample_rate` field in the `requested_virtual_sensors` structure passed to `bsec_update_subscription()` should match with the sampling rate defined for that sensor. The allowed sampling rate(s) in Hertz for each sensor is listed in this [table](#).

3.3.2.2 BSEC_E_SU_SAMPLERATELIMITS (-11)

Possible reasons:

- ▶ Virtual sensors are requested with an incorrect sampling rate.

E.g:

```
requested_virtual_sensors[virtual_sensor_count].sample_rate = 5;
requested_virtual_sensors[virtual_sensor_count].sensor_id = BSEC_OUTPUT_RAW_GAS;
status=bsec_update_subscription(requested_virtual_sensors,
    n_requested_virtual_sensors, required_sensor_settings, &n_required_sensor_settings);
```

Correction methods:

- ▶ The `sample_rate` field in the `requested_virtual_sensors` structure passed to `bsec_update_subscription()` should match with the sampling rate defined for that sensor. The allowed sampling rate(s) in Hertz for each sensor is listed in this [table](#).

3.3.2.3 BSEC_E_SU_DUPLICATEGATE (-13)

Possible reasons:

- ▶ Duplicate virtual sensors are requested through `bsec_update_subscription()` function.

E.g:

```
requested_virtual_sensors[virtual_sensor_count].sample_rate = 1;
requested_virtual_sensors[virtual_sensor_count].sensor_id = BSEC_OUTPUT_RAW_GAS;
virtual_sensor_count++;
requested_virtual_sensors[virtual_sensor_count].sample_rate = 1;
requested_virtual_sensors[virtual_sensor_count].sensor_id = BSEC_OUTPUT_RAW_GAS;
status = bsec_update_subscription(requested_virtual_sensors,
    n_requested_virtual_sensors, required_sensor_settings ,&n_required_sensor_settings);
```

Correction methods:

- ▶ The `sensor_id` field in the `requested_virtual_sensors` structure passed to `bsec_update_subscription()` should be unique.

3.3.2.4 BSEC_E_SU_INVALIDSAMPLERATE (-14)

Possible reasons:

- ▶ The sampling rate of the requested virtual sensors is not within the valid limit.

E.g:

```
requested_virtual_sensors[virtual_sensor_count].sample_rate = 100;
requested_virtual_sensors[virtual_sensor_count].sensor_id = BSEC_OUTPUT_RAW_GAS;
status = bsec_update_subscription(requested_virtual_sensors,
    n_requested_virtual_sensors, required_sensor_settings, &n_required_sensor_settings);
```

Correction methods:

- ▶ The `sample_rate` field in the `requested_virtual_sensors` structure passed to `bsec_update_subscription()` should match with the sampling rate defined for that sensor. The allowed sampling rate(s) in Hertz for each sensor is listed in this [table](#).

3.3.2.5 BSEC_E_SU_GATECOUNTEXCEEDSARRAY (-15)

Possible reasons:

- ▶ Enough memory is not allocated to hold the returned physical sensor data from `bsec_update_subscription()`.

E.g:

```
n_required_sensor_settings = 0;
status = bsec_update_subscription(requested_virtual_sensors,
    n_requested_virtual_sensors, required_sensor_settings, &n_required_sensor_settings);
```

Correction methods:

- ▶ `n_required_sensor_settings` passed to `bsec_update_subscription()` should be equal to `BSEC_MAX_PHYSICAL_SENSOR`.

3.3.2.6 BSEC_E_SU_SAMPLINTVLINTEGERMULT (-16)

Possible reasons:

- ▶ The sampling rate of an output requested via `bsec_update_subscription()` is not an integer multiple of the other active sampling rates.

Correction methods:

- ▶ Use one of the sampling rates listed in this [table](#).

3.3.2.7 BSEC_E_SU_MULTGASSAMPLINTVL (-17)

Possible reasons:

- ▶ The sampling rate of the requested output requires the gas sensor, which is currently running at a different sampling rate.

Correction methods:

- ▶ The outputs that require the gas sensor must have equal sampling rates.

3.3.2.8 BSEC_E_SU_HIGHHEATERONDURATION (-18)

Possible reasons:

- ▶ The sampling period of the requested output is lower than the duration of a complete measurement.

Correction methods:

- ▶ Use a slower sampling rate.

3.3.2.9 BSEC_W_SU_UNKNOWNOUTPUTGATE (10)

Possible reasons:

- ▶ Requested virtual sensor id is not valid.
- ▶ Number of virtual sensors passed to `bsec_update_subscription()` is greater than the actual number of output(virtual) sensors requested.

E.g:

```
requested_virtual_sensors[virtual_sensor_count].sample_rate = 1;
requested_virtual_sensors[virtual_sensor_count].sensor_id = 100;

n_requested_virtual_sensors = 3;
status = bsec_update_subscription(requested_virtual_sensors,
    n_requested_virtual_sensors, required_sensor_settings, &n_required_sensor_settings);
```

Correction methods:

- ▶ The `sensor_id` field in the `requested_virtual_sensors` structure passed to `bsec_update_subscription()` should be in the range of `bsec_virtual_sensor_t` enum.
- ▶ `n_requested_virtual_sensors` should be equal to the number of output (virtual) sensors requested through `bsec_update_subscription()` i.e. size of `requested_virtual_sensors` structure array.

3.3.2.10 BSEC_I_SU_SUBSCRIBEDOUTPUTGATES (12)

Possible reasons:

- ▶ No output (virtual) sensor requested through `bsec_update_subscription()`
- ▶ Number of requested output (virtual) sensors passed to `bsec_update_subscription()` is zero even when there are some output (virtual) sensors requested.

E.g:

```
requested_virtual_sensors[virtual_sensor_count].sample_rate = 1/300;
requested_virtual_sensors[virtual_sensor_count].sensor_id = BSEC_OUTPUT_RAW_GAS;
n_requested_virtual_sensors = 0;
status = bsec_update_subscription(requested_virtual_sensors,
    n_requested_virtual_sensors, required_sensor_settings, &n_required_sensor_settings);
```

Correction methods:

- ▶ `requested_virtual_sensors` structure to `bsec_update_subscription()` should be populated with the data of the required output (virtual) sensors. It should not be empty.
- ▶ `n_requested_virtual_sensors` should be equal to the number of output (virtual) sensors requested via `bsec_update_subscription()`, i.e., size of `requested_virtual_sensors` structure array. It should not be zero.

3.3.3 Errors Returned by `bsec_set_configuration()` / `bsec_set_state()`

3.3.3.1 BSEC_E_CONFIG_VERSIONMISMATCH (-34)

Possible reasons:

- ▶ Version mentioned in the configuration string or state string passed to `bsec_set_configuration()` or `bsec_set_state()`, respectively, does not match with the current version.

Correction methods:

- ▶ Obtain a compatible string.
- ▶ A call to `bsec_get_version()` would give the current version information.

3.3.3.2 BSEC_E_CONFIG_FEATUREMISMATCH (-35)

Possible reasons:

- ▶ Enabled features encoded in configuration/state strings passed to `bsec_set_configuration()/bsec_set_state()` does not match with current library implementation.

Correction methods:

- ▶ Ensure the configuration/state strings generated for current library implementation is passed to `bsec_set_configuration()/bsec_set_state()`.

3.3.3.3 BSEC_E_CONFIG_CRCMISMATCH (-36)

Possible reasons:

- ▶ Difference in configuration/state strings passed to `bsec_set_configuration()/bsec_set_state()` from what is generated for current library implementation.
- ▶ String was corrupted during storage or loading.
- ▶ String was truncated.
- ▶ String is shorter than the size argument provided to the setter function.

Correction methods:

- ▶ Ensure the configuration/state strings generated for current library implementation is passed to `bsec_set_configuration()/bsec_set_state()`.

3.3.3.4 BSEC_E_CONFIG_EMPTY (-37)

Possible reasons:

- ▶ String passed to the setter function is too short to be able to be a valid string.

Correction methods:

- ▶ Obtain a valid config string.

3.3.3.5 BSEC_E_CONFIG_INSUFFICIENTWORKBUFFER (-38)

Possible reasons:

- ▶ Length of work buffer passed to `bsec_set_configuration()` or `bsec_set_state()` is less than required value.
- ▶ Length of work buffer passed to `bsec_get_configuration()` or `bsec_get_state()` is less than required value.

Correction methods:

- ▶ Value of `n_work_buffer_size` passed to `bsec_set_configuration()` and `bsec_set_state()` should be assigned the maximum value `BSEC_MAX_PROPERTY_BLOB_SIZE`.
- ▶ Value of `n_work_buffer` passed to `bsec_get_configuration()` and `bsec_get_state()` should be assigned the maximum value `BSEC_MAX_PROPERTY_BLOB_SIZE`.

3.3.3.6 BSEC_E_CONFIG_INVALIDSTRINGSIZE (-40)

Possible reasons:

- ▶ String size encoded in configuration/state strings passed to `bsec_set_configuration()` / `bsec_set_state()` does not match with the actual string size `n_serialized_settings/n_serialized_state` passed to these functions.

Correction methods:

- ▶ Ensure the configuration/state strings generated for current library implementation is passed to `bsec_set_configuration()`/`bsec_set_state()`.

3.3.3.7 BSEC_E_CONFIG_INSUFFICIENTBUFFER (-41)

Possible reasons:

- ▶ Value of `n_serialized_settings_max/n_serialized_state_max` passed to `bsec_get_configuration()`/`bsec_get_state()` is insufficient to hold serialized data from BSEC library.

Correction methods:

- ▶ Value of `n_serialized_settings_max/n_serialized_state_max` passed to `bsec_get_configuration()` /`bsec_get_state()` should be equal to `BSEC_MAX_PROPERTY_BLOB_SIZE`.

3.3.4 Errors Returned by `bsec_sensor_control()`

3.3.4.1 BSEC_W_SC_CALL_TIMING_VIOLATION (100)

Possible reasons:

- ▶ The timestamp at which `bsec_sensor_control(timestamp)` is called differs from the target timestamp which was returned during the previous call in the `.next_call` struct member by more than 6.25%.

Correction methods:

- ▶ Ensure that your system calls `bsec_sensor_control()` at the time instructed in the previous call.
- ▶ To ensure proper operation, a maximum jitter of 6.25% is allowed.

4 Module Documentation

4.1 BSEC C Interface

4.1.1 Interface Usage

Interfaces of BSEC signal processing library.

Interface usage The following provides a short overview on the typical operation sequence for BSEC.

- Initialization of the library

Steps	Function
Initialization of library	bsec_init()
Update configuration settings (optional)	bsec_set_configuration()
Restore the state of the library (optional)	bsec_set_state()

- The following function is called to enable output signals and define their sampling rate / operation mode.

Steps	Function
Enable library outputs with specified mode	bsec_update_subscription()

- This table describes the main processing loop.

Steps	Function
Retrieve sensor settings to be used	bsec_sensor_control()
Configure sensor and trigger measurement	See BME680 API and example codes
Read results from sensor	See BME680 API and example codes
Perform signal processing	bsec_do_steps()

- Before shutting down the system, the current state of BSEC can be retrieved and can then be used during re-initialization to continue processing.

Steps	Function
To retrieve the current library state	bsec_get_state()

Configuration and state Values of variables belonging to a BSEC instance are divided into two groups:

- Values **not updated by processing** of signals belong to the **configuration group**. If available, BSEC can be configured before use with a customer specific configuration string.

- ▶ Values **updated during processing** are member of the **state group**. Saving and restoring of the state of BSEC is necessary to maintain previously estimated sensor models and baseline information which is important for best performance of the gas sensor outputs.

Note

BSEC library consists of adaptive algorithms which models the gas sensor which improves its performance over the time. These will be lost if library is initialised due to system reset. In order to avoid this situation library state shall be stored in non volatile memory so that it can be loaded after system reset.

4.1.2 Interface Functions

4.1.2.1 bsec_do_steps()

```
bsec_library_return_t bsec_do_steps (
    const bsec_input_t *const inputs,
    const uint8_t n_inputs,
    bsec_output_t * outputs,
    uint8_t * n_outputs )
```

Main signal processing function of BSEC.

Processing of the input signals and returning of output samples is performed by `bsec_do_steps()`.

- ▶ The samples of all library inputs must be passed with unique identifiers representing the input signals from physical sensors where the order of these inputs can be chosen arbitrary. However, all input have to be provided within the same time period as they are read. A sequential provision to the library might result in undefined behaviour.
- ▶ The samples of all library outputs are returned with unique identifiers corresponding to the output signals of virtual sensors where the order of the returned outputs may be arbitrary.
- ▶ The samples of all input as well as output signals of physical as well as virtual sensors use the same representation in memory with the following fields:
 - ▶ Sensor identifier:
 - ▶ For inputs: required to identify the input signal from a physical sensor
 - ▶ For output: overwritten by `bsec_do_steps()` to identify the returned signal from a virtual sensor
 - ▶ Time stamp of the sample

Calling `bsec_do_steps()` requires the samples of the input signals to be provided along with their time stamp when they are recorded and only when they are acquired. Repetition of samples with the same time stamp are ignored and result in a warning. Repetition of values of samples which are not acquired anew by a sensor result in deviations of the computed output signals. Concerning the returned output samples, an important feature is, that a value is returned for an output only when a new occurrence has been computed. A sample of an output signal is returned only once.

Parameters

in	<i>inputs</i>	Array of input data samples. Each array element represents a sample of a different physical sensor.
in	<i>n_inputs</i>	Number of passed input data structs.
out	<i>outputs</i>	Array of output data samples. Each array element represents a sample of a different virtual sensor.

Parameters

in, out	<i>n_outputs</i>	[in] Number of allocated output structs, [out] number of outputs returned
---------	------------------	---

Returns

Zero when successful, otherwise an error code

```
// Example //
```

```
// Allocate input and output memory
bsec_input_t input[3];
uint8_t n_input = 3;
bsec_output_t output[2];
uint8_t n_output=2;

bsec_library_return_t status;

// Populate the input structs, assuming the we have timestamp (ts),
// gas sensor resistance (R), temperature (T), and humidity (rH) available
// as input variables
input[0].sensor_id = BSEC_INPUT_GASRESISTOR;
input[0].signal = R;
input[0].time_stamp= ts;
input[1].sensor_id = BSEC_INPUT_TEMPERATURE;
input[1].signal = T;
input[1].time_stamp= ts;
input[2].sensor_id = BSEC_INPUT_HUMIDITY;
input[2].signal = rH;
input[2].time_stamp= ts;

// Invoke main processing BSEC function
status = bsec_do_steps( input, n_input, output, &n_output );

// Iterate through the BSEC output data, if the call succeeded
if(status == BSEC_OK)
{
    for(int i = 0; i < n_output; i++)
    {
        switch(output[i].sensor_id)
        {
            case BSEC_OUTPUT_IAQ_ESTIMATE:
                // Retrieve the IAQ results from output[i].signal
                // and do something with the data
                break;
            case BSEC_OUTPUT_AMBIENT_TEMPERATURE:
                // Retrieve the ambient temperature results from output[i].signal
                // and do something with the data
                break;
        }
    }
}
```

4.1.2.2 bsec_get_configuration()

```
bsec_library_return_t bsec_get_configuration (
    const uint8_t config_id,
    uint8_t * serialized_settings,
    const uint32_t n_serialized_settings_max,
    uint8_t * work_buffer,
```

```
const uint32_t n_work_buffer,
uint32_t * n_serialized_settings )
```

Retrieve the current library configuration.

BSEC allows to retrieve the current configuration using `bsec_get_configuration()`. Returns a binary blob encoding the current configuration parameters of the library in a format compatible with `bsec_set_configuration()`.

Note

The function `bsec_get_configuration()` is required to be used for debugging purposes only. A work buffer with sufficient size is required and has to be provided by the function caller to decompose the serialization and apply it to the library and its modules. Please use `BSEC_MAX_PROPERTY_BLOB_SIZE` for allotting the required size.

Parameters

in	<i>config_id</i>	Identifier for a specific set of configuration settings to be returned; shall be zero to retrieve all configuration settings.
out	<i>serialized_settings</i>	Buffer to hold the serialized config blob
in	<i>n_serialized_settings_max</i>	Maximum available size for the serialized settings
in,out	<i>work_buffer</i>	Work buffer used to parse the binary blob
in	<i>n_work_buffer</i>	Length of the work buffer available for parsing the blob
out	<i>n_serialized_settings</i>	Actual size of the returned serialized configuration blob

Returns

Zero when successful, otherwise an error code

```
// Example //

// Allocate variables
uint8_t serialized_settings[BSEC_MAX_PROPERTY_BLOB_SIZE];
uint32_t n_serialized_settings_max = BSEC_MAX_PROPERTY_BLOB_SIZE;
uint8_t work_buffer[BSEC_MAX_PROPERTY_BLOB_SIZE];
uint32_t n_work_buffer = BSEC_MAX_PROPERTY_BLOB_SIZE;
uint32_t n_serialized_settings = 0;

// Configuration of BSEC algorithm is stored in 'serialized_settings'
bsec_get_configuration(0, serialized_settings, n_serialized_settings_max, work_buffer
, n_work_buffer, &n_serialized_settings);
```

4.1.2.3 bsec_get_state()

```
bsec_library_return_t bsec_get_state (
    const uint8_t state_set_id,
    uint8_t * serialized_state,
    const uint32_t n_serialized_state_max,
    uint8_t * work_buffer,
    const uint32_t n_work_buffer,
    uint32_t * n_serialized_state )
```

Retrieve the current internal library state.

BSEC allows to retrieve the current states of all signal processing modules and the BSEC module using `bsec_get_state()`. This allows a restart of the processing after a reboot of the system by calling `bsec_set_state()`.

Note

A work buffer with sufficient size is required and has to be provided by the function caller to decompose the serialization and apply it to the library and its modules. Please use `BSEC_MAX_PROPERTY_BLOB_SIZE` for allotting the required size.

Parameters

in	<code>state_set_id</code>	Identifier for a specific set of states to be returned; shall be zero to retrieve all states.
out	<code>serialized_state</code>	Buffer to hold the serialized config blob
in	<code>n_serialized_state_max</code>	Maximum available size for the serialized states
in,out	<code>work_buffer</code>	Work buffer used to parse the blob
in	<code>n_work_buffer</code>	Length of the work buffer available for parsing the blob
out	<code>n_serialized_state</code>	Actual size of the returned serialized blob

Returns

Zero when successful, otherwise an error code

// Example //

// Allocate variables

```
uint8_t serialized_state[BSEC_MAX_PROPERTY_BLOB_SIZE];
uint32_t n_serialized_state_max=BSEC_MAX_PROPERTY_BLOB_SIZE;
uint32_t n_serialized_state = BSEC_MAX_PROPERTY_BLOB_SIZE;
uint8_t work_buffer_state[BSEC_MAX_PROPERTY_BLOB_SIZE];
uint32_t n_work_buffer_size = BSEC_MAX_PROPERTY_BLOB_SIZE;
```

// Algorithm state is stored in 'serialized_state'

```
bsec_get_state(0, serialized_state, n_serialized_state_max, work_buffer_state,
n_work_buffer_size, &n_serialized_state);
```

4.1.2.4 bsec_get_version()

```
bsec_library_return_t bsec_get_version (
    bsec_version_t * bsec_version_p )
```

Return the version information of BSEC library.

Parameters

out	<code>bsec_version↔_p</code>	pointer to struct which is to be populated with the version information
-----	------------------------------	---

Returns

Zero if successful, otherwise an error code

See also: `bsec_version_t`

```
// Example //
bsec_version_t version;
bsec_get_version(&version);
printf("BSEC version: %d.%d.%d", version.major, version.minor, version.
    major_bugfix, version.minor_bugfix);
```

4.1.2.5 bsec_init()

```
bsec_library_return_t bsec_init (
    void )
```

Initialize the library.

Initialization and reset of BSEC is performed by calling `bsec_init()`. Calling this function sets up the relation among all internal modules, initializes run-time dependent library states and resets the configuration and state of all BSEC signal processing modules to defaults.

Before any further use, the library must be initialized. This ensure that all memory and states are in defined conditions prior to processing any data.

Returns

Zero if successful, otherwise an error code

```
// Initialize BSEC library before further use
bsec_init();
```

4.1.2.6 bsec_reset_output()

```
bsec_library_return_t bsec_reset_output (
    uint8_t sensor_id )
```

Reset a particular virtual sensor output.

This function allows specific virtual sensor outputs to be reset. The meaning of "reset" depends on the specific output. In case of the IAQ output, reset means zeroing the output to the current ambient conditions.

Parameters

in	<i>sensor_id</i>	Virtual sensor to be reset
----	------------------	----------------------------

Returns

Zero when successful, otherwise an error code

```
// Example //
bsec_reset_output(BSEC_OUTPUT_IAQ_ESTIMATE);
```

4.1.2.7 bsec_sensor_control()

```
bsec_library_return_t bsec_sensor_control (
```

```
const int64_t time_stamp,
bsec_bme_settings_t * sensor_settings )
```

Retrieve BMExxx sensor instructions.

The `bsec_sensor_control()` interface is a key feature of BSEC, as it allows an easy way for the signal processing library to control the operation of the BME sensor. This is important since gas sensor behaviour is mainly determined by how the integrated heater is configured. To ensure an easy integration of BSEC into any system, `bsec_sensor_control()` will provide the caller with information about the current sensor configuration that is necessary to fulfill the input requirements derived from the current outputs requested via `bsec_update_subscription()`.

In practice the use of this function shall be as follows:

- ▶ Call `bsec_sensor_control()` which returns a `bsec_bme_settings_t` struct.
- ▶ Based on the information contained in this struct, the sensor is configured and a forced-mode measurement is triggered if requested by `bsec_sensor_control()`.
- ▶ Once this forced-mode measurement is complete, the signals specified in this struct shall be passed to `bsec_do_steps()` to perform the signal processing.
- ▶ After processing, the process should sleep until the `bsec_bme_settings_t::next_call` timestamp is reached.

Parameters

in	<code>time_stamp</code>	Current timestamp in [ns]
out	<code>sensor_settings</code>	Settings to be passed to API to operate sensor at this time instance

Returns

Zero when successful, otherwise an error code

4.1.2.8 bsec_set_configuration()

```
bsec_library_return_t bsec_set_configuration (
    const uint8_t *const serialized_settings,
    const uint32_t n_serialized_settings,
    uint8_t * work_buffer,
    const uint32_t n_work_buffer_size )
```

Update algorithm configuration parameters.

BSEC uses a default configuration for the modules and common settings. The initial configuration can be customized by `bsec_set_configuration()`. This is an optional step.

Note

A work buffer with sufficient size is required and has to be provided by the function caller to decompose the serialization and apply it to the library and its modules. Please use `BSEC_MAX_PROPERTY_BLOB_SIZE` for allotting the required size.

Parameters

in	<code>serialized_settings</code>	Settings serialized to a binary blob
in	<code>n_serialized_settings</code>	Size of the settings blob

Parameters

in,out	<i>work_buffer</i>	Work buffer used to parse the blob
in	<i>n_work_buffer_size</i>	Length of the work buffer available for parsing the blob

Returns

Zero when successful, otherwise an error code

```
// Example //

// Allocate variables
uint8_t serialized_settings[BSEC_MAX_PROPERTY_BLOB_SIZE];
uint32_t n_serialized_settings_max = BSEC_MAX_PROPERTY_BLOB_SIZE;
uint8_t work_buffer[BSEC_MAX_PROPERTY_BLOB_SIZE];
uint32_t n_work_buffer = BSEC_MAX_PROPERTY_BLOB_SIZE;

// Here we will load a provided config string into serialized_settings

// Apply the configuration
bsec_set_configuration(serialized_settings, n_serialized_settings_max, work_buffer,
    n_work_buffer);
```

4.1.2.9 bsec_set_state()

```
bsec_library_return_t bsec_set_state (
    const uint8_t *const serialized_state,
    const uint32_t n_serialized_state,
    uint8_t * work_buffer,
    const uint32_t n_work_buffer_size )
```

Restore the internal state of the library.

BSEC uses a default state for all signal processing modules and the BSEC module. To ensure optimal performance, especially of the gas sensor functionality, it is recommended to retrieve the state using `bsec_get_state()` before unloading the library, storing it in some form of non-volatile memory, and setting it using `bsec_set_state()` before resuming further operation of the library.

Note

A work buffer with sufficient size is required and has to be provided by the function caller to decompose the serialization and apply it to the library and its modules. Please use `BSEC_MAX_PROPERTY_BLOB_SIZE` for allotting the required size.

Parameters

in	<i>serialized_state</i>	States serialized to a binary blob
in	<i>n_serialized_state</i>	Size of the state blob
in,out	<i>work_buffer</i>	Work buffer used to parse the blob
in	<i>n_work_buffer_size</i>	Length of the work buffer available for parsing the blob

Returns

Zero when successful, otherwise an error code

```
// Example //
```

```
// Allocate variables
uint8_t serialized_state[BSEC_MAX_PROPERTY_BLOB_SIZE];
uint32_t n_serialized_state = BSEC_MAX_PROPERTY_BLOB_SIZE;
uint8_t work_buffer_state[BSEC_MAX_PROPERTY_BLOB_SIZE];
uint32_t n_work_buffer_size = BSEC_MAX_PROPERTY_BLOB_SIZE;

// Here we will load a state string from a previous use of BSEC

// Apply the previous state to the current BSEC session
bsec_set_state(serialized_state, n_serialized_state, work_buffer_state, n_work_buffer_size);
```

4.1.2.10 bsec_update_subscription()

```
bsec_library_return_t bsec_update_subscription (
    const bsec_sensor_configuration_t *const requested_virtual_sensors,
    const uint8_t n_requested_virtual_sensors,
    bsec_sensor_configuration_t * required_sensor_settings,
    uint8_t * n_required_sensor_settings )
```

Subscribe to library virtual sensors outputs.

Use `bsec_update_subscription()` to instruct BSEC which of the processed output signals are requested at which sample rates. See `bsec_virtual_sensor_t` for available library outputs.

Based on the requested virtual sensors outputs, BSEC will provide information about the required physical sensor input signals (see `bsec_physical_sensor_t`) with corresponding sample rates. This information is purely informational as `bsec_sensor_control()` will ensure the sensor is operated in the required manner. To disable a virtual sensor, set the sample rate to `BSEC_SAMPLE_RATE_DISABLED`.

The subscription update using `bsec_update_subscription()` is apart from the signal processing one of the most important functions. It allows to enable the desired library outputs. The function determines which physical input sensor signals are required at which sample rate to produce the virtual output sensor signals requested by the user. When this function returns with success, the requested outputs are called subscribed. A very important feature is the retaining of already subscribed outputs. Further outputs can be requested or disabled both individually and group-wise in addition to already subscribed outputs without changing them unless a change of already subscribed outputs is requested.

Note

The state of the library concerning the subscribed outputs cannot be retained among reboots.

The interface of `bsec_update_subscription()` requires the usage of arrays of sensor configuration structures. Such a structure has the fields sensor identifier and sample rate. These fields have the properties:

- ▶ Output signals of virtual sensors must be requested using unique identifiers (Member of `bsec_virtual_sensor_t`)
- ▶ Different sets of identifiers are available for inputs of physical sensors and outputs of virtual sensors
- ▶ Identifiers are unique values defined by the library, not from external
- ▶ Sample rates must be provided as value of
 - ▶ An allowed sample rate for continuously sampled signals
 - ▶ 65535.0f (`BSEC_SAMPLE_RATE_DISABLED`) to turn off outputs and identify disabled inputs

Note

The same sensor identifiers are also used within the functions `bsec_do_steps()`.

The usage principles of `bsec_update_subscription()` are:

- ▶ Differential updates (i.e., only asking for outputs that the user would like to change) is supported.
- ▶ Invalid requests of outputs are ignored. Also if one of the requested outputs is unavailable, all the requests are ignored. At the same time, a warning is returned.
- ▶ To disable BSEC, all outputs shall be turned off. Only enabled (subscribed) outputs have to be disabled while already disabled outputs do not have to be disabled explicitly.

Parameters

in	<i>requested_virtual_sensors</i>	Pointer to array of requested virtual sensor (output) configurations for the library
in	<i>n_requested_virtual_sensors</i>	Number of virtual sensor structs pointed by <i>requested_virtual_sensors</i>
out	<i>required_sensor_settings</i>	Pointer to array of required physical sensor configurations for the library
in,out	<i>n_required_sensor_settings</i>	[in] Size of allocated <i>required_sensor_settings</i> array, [out] number of sensor configurations returned

Returns

Zero when successful, otherwise an error code

See also

[bsec_sensor_configuration_t](#)
[bsec_physical_sensor_t](#)
[bsec_virtual_sensor_t](#)

// Example //

```
// Change 3 virtual sensors (switch IAQ and raw temperature -> on / pressure -> off)
bsec_sensor_configuration_t requested_virtual_sensors[3];
uint8_t n_requested_virtual_sensors = 3;
```

```
requested_virtual_sensors[0].sensor_id = BSEC_OUTPUT_IAQ_ESTIMATE;
requested_virtual_sensors[0].sample_rate = BSEC_SAMPLE_RATE_ULP;
requested_virtual_sensors[1].sensor_id = BSEC_OUTPUT_RAW_TEMPERATURE;
requested_virtual_sensors[1].sample_rate = BSEC_SAMPLE_RATE_ULP;
requested_virtual_sensors[2].sensor_id = BSEC_OUTPUT_RAW_PRESSURE;
requested_virtual_sensors[2].sample_rate = BSEC_SAMPLE_RATE_DISABLED;
```

```
// Allocate a struct for the returned physical sensor settings
bsec_sensor_configuration_t required_sensor_settings[
    BSEC_MAX_PHYSICAL_SENSOR];
uint8_t n_required_sensor_settings = BSEC_MAX_PHYSICAL_SENSOR;
```

```
// Call bsec_update_subscription() to enable/disable the requested virtual sensors
bsec_update_subscription(requested_virtual_sensors, n_requested_virtual_sensors,
    required_sensor_settings, &n_required_sensor_settings);
```


4.1.3 Enumerations

4.1.3.1 bsec_library_return_t

enum `bsec_library_return_t`

Enumeration for function return codes.

Enumerator

BSEC_OK	Function execution successful
BSEC_E_DOSTEPS_INVALIDINPUT	Input (physical) sensor id passed to <code>bsec_do_steps()</code> is not in the valid range or not valid for requested virtual sensor
BSEC_E_DOSTEPS_VALUELIMITS	Value of input (physical) sensor signal passed to <code>bsec_do_steps()</code> is not in the valid range
BSEC_E_DOSTEPS_DUPLICATEINPUT	Duplicate input (physical) sensor ids passed as input to <code>bsec_do_steps()</code>
BSEC_I_DOSTEPS_NOOUTPUTSRETURNABLE	No memory allocated to hold return values from <code>bsec_do_steps()</code> , i.e., <code>n_outputs == 0</code>
BSEC_W_DOSTEPS_EXCESSOUTPUTS	Not enough memory allocated to hold return values from <code>bsec_do_steps()</code> , i.e., <code>n_outputs < maximum number of requested output (virtual) sensors</code>
BSEC_W_DOSTEPS_TSINTRADIFFOUTOFRANGE	Duplicate timestamps passed to <code>bsec_do_steps()</code>
BSEC_E_SU_WRONGDATARATE	The <code>sample_rate</code> of the requested output (virtual) sensor passed to <code>bsec_update_subscription()</code> is zero
BSEC_E_SU_SAMPLERATELIMITS	The <code>sample_rate</code> of the requested output (virtual) sensor passed to <code>bsec_update_subscription()</code> does not match with the sampling rate allowed for that sensor
BSEC_E_SU_DUPLICATEGATE	Duplicate output (virtual) sensor ids requested through <code>bsec_update_subscription()</code>
BSEC_E_SU_INVALIDSAMPLERATE	The <code>sample_rate</code> of the requested output (virtual) sensor passed to <code>bsec_update_subscription()</code> does not fall within the global minimum and maximum sampling rates
BSEC_E_SU_GATECOUNTEXCEEDSARRAY	Not enough memory allocated to hold returned input (physical) sensor data from <code>bsec_update_subscription()</code> , i.e., <code>n_required_sensor_settings < BSEC_MAX_PHYSICAL_SENSOR</code>
BSEC_E_SU_SAMPLINTVLINTEGERMULT	The <code>sample_rate</code> of the requested output (virtual) sensor passed to <code>bsec_update_subscription()</code> is not correct
BSEC_E_SU_MULTGASSAMPLINTVL	The <code>sample_rate</code> of the requested output (virtual), which requires the gas sensor, is not equal to the <code>sample_rate</code> that the gas sensor is being operated

Enumerator

BSEC_E_SU_HIGHHEATERONDURATION	The duration of one measurement is longer than the requested sampling interval
BSEC_W_SU_UNKNOWNOUTPUTGATE	Output (virtual) sensor id passed to bsec_update_subscription() is not in the valid range; e.g., <code>n_requested_virtual_sensors > actual number of output (virtual) sensors requested</code>
BSEC_I_SU_SUBSCRIBEDOUTPUTGATES	No output (virtual) sensor data were requested via bsec_update_subscription()
BSEC_E_PARSE_SECTIONEXCEEDSWORKBU↔ FFER	<code>n_work_buffer_size</code> passed to <code>bsec_set_[configuration/state]()</code> not sufficient
BSEC_E_CONFIG_FAIL	Configuration failed
BSEC_E_CONFIG_VERSIONMISMATCH	Version encoded in serialized_[settings/state] passed to <code>bsec_set_[configuration/state]()</code> does not match with current version
BSEC_E_CONFIG_FEATUREMISMATCH	Enabled features encoded in serialized_[settings/state] passed to <code>bsec_set_[configuration/state]()</code> does not match with current library implementation
BSEC_E_CONFIG_CRCMISMATCH	serialized_[settings/state] passed to <code>bsec_set_[configuration/state]()</code> is corrupted
BSEC_E_CONFIG_EMPTY	<code>n_serialized_[settings/state]</code> passed to <code>bsec_set_[configuration/state]()</code> is too short to be valid
BSEC_E_CONFIG_INSUFFICIENTWORKBUFFER	Provided work_buffer is not large enough to hold the desired string
BSEC_E_CONFIG_INVALIDSTRINGSIZE	String size encoded in configuration/state strings passed to <code>bsec_set_[configuration/state]()</code> does not match with the actual string size <code>n_serialized_[settings/state]</code> passed to these functions
BSEC_E_CONFIG_INSUFFICIENTBUFFER	String buffer insufficient to hold serialized data from BSEC library
BSEC_E_SET_INVALIDCHANNELIDENTIFIER	Internal error code
BSEC_E_SET_INVALIDLENGTH	Internal error code
BSEC_W_SC_CALL_TIMING_VIOLATION	Difference between actual and defined sampling intervals of bsec_sensor_control() greater than allowed

4.1.3.2 `bsec_physical_sensor_t`

enum [bsec_physical_sensor_t](#)

Enumeration for input (physical) sensors.

Used to populate [bsec_input_t::sensor_id](#). It is also used in [bsec_sensor_configuration_t::sensor_id](#) structs returned in the parameter `required_sensor_settings` of [bsec_update_subscription\(\)](#).

See also

[bsec_sensor_configuration_t](#)
[bsec_input_t](#)

Enumerator

BSEC_INPUT_PRESSURE	Pressure sensor output of BMExxx [Pa].
BSEC_INPUT_HUMIDITY	Humidity sensor output of BMExxx [%]. <p>Note</p> <p>Relative humidity strongly depends on the temperature (it is measured at). It may required a conversion to the temperature outside of the device.</p> <p>See also</p> <p>bsec_virtual_sensor_t</p>
BSEC_INPUT_TEMPERATURE	Temperature sensor output of BMExxx [degrees Celcius]. <p>Note</p> <p>The BME680 is factory trimmed, thus the temperature sensor of the BME680 is very accurate. The temperature value is a very local measurement value and can be influenced by external heat sources.</p> <p>See also</p> <p>bsec_virtual_sensor_t</p>
BSEC_INPUT_GASRESISTOR	Gas sensor resistance output of BMExxx [Ohm]. The restistance value changes due to varying VOC concentrations (the higher the concentration of reducing VOCs, the lower the resistance and vice versa).
BSEC_INPUT_HEATSOURCE	Additional input for device heat compensation. IAQ solution: The value is substracted from BSEC_INPUT_TEMPERATURE to compute BSEC_OUTPUT_SENSOR_HEAT_COMPENSATED_TEMPERATURE . ALL solution: Generic heatsource 1 <p>See also</p> <p>bsec_virtual_sensor_t</p>

4.1.3.3 bsec_virtual_sensor_t

enum [bsec_virtual_sensor_t](#)

Enumeration for output (virtual) sensors.

Used to populate [bsec_output_t::sensor_id](#). It is also used in [bsec_sensor_configuration_t::sensor_id](#) structs

passed in the parameter requested_virtual_sensors of [bsec_update_subscription\(\)](#).

See also

[bsec_sensor_configuration_t](#)

[bsec_output_t](#)

Enumerator

BSEC_OUTPUT_IAQ_ESTIMATE	<p>Indoor-air-quality estimate [0-500]. Indoor-air-quality (IAQ) gives an indication of the relative change in ambient TVOCs detected by BME680.</p> <p>Note</p> <p>The IAQ scale ranges from 0 (clean air) to 500 (heavily polluted air). During operation, algorithms automatically calibrate and adapt themselves to the typical environments where the sensor is operated (e.g., home, workplace, inside a car, etc.). This automatic background calibration ensures that users experience consistent IAQ performance. The calibration process considers the recent measurement history (typ. up to four days) to ensure that IAQ=25 corresponds to typical good air and IAQ=250 indicates typical polluted air.</p>
BSEC_OUTPUT_RAW_TEMPERATURE	<p>Temperature sensor signal [degrees Celcius]. Temperature directly measured by BME680 in degree Celcius.</p> <p>Note</p> <p>This value is cross-influenced by the sensor heating and device specific heating.</p>
BSEC_OUTPUT_RAW_PRESSURE	<p>Pressure sensor signal [Pa]. Pressure directly measured by the BME680 in Pa.</p>
BSEC_OUTPUT_RAW_HUMIDITY	<p>Relative humidity sensor signal [%]. Relative humidity directly measured by the BME680 in %.</p> <p>Note</p> <p>This value is cross-influenced by the sensor heating and device specific heating.</p>
BSEC_OUTPUT_RAW_GAS	<p>Gas sensor signal [Ohm]. Gas resistance measured directly by the BME680 in Ohm. The resistance value changes due to varying VOC concentrations (the higher the concentration of reducing VOCs, the lower the resistance and vice versa).</p>

Enumerator

BSEC_OUTPUT_STABILIZATION_STATUS	Gas sensor stabilization status [boolean]. Indicates initial stabilization status of the gas sensor element: stabilization is ongoing (0) or stabilization is finished (1).
BSEC_OUTPUT_RUN_IN_STATUS	Gas sensor run-in status [boolean]. Indicates power-on stabilization status of the gas sensor element: stabilization is ongoing (0) or stabilization is finished (1).
BSEC_OUTPUT_SENSOR_HEAT_COMPENSATED_TEMPERATURE	<p>Sensor heat compensated temperature [degrees Celcius]. Temperature measured by BME680 which is compensated for the influence of sensor (heater) in degree Celcius. The self heating introduced by the heater is depending on the sensor operation mode and the sensor supply voltage.</p> <p>Note</p> <p>IAQ solution: In addition, the temperature output can be compensated by an user defined value (BSEC_INPUT_HEATSOURCE in degrees Celcius), which represents the device specific self-heating.</p> <p>Thus, the value is calculated as follows:</p> <ul style="list-style-type: none"> ▶ IAQ solution: $BSEC_OUTPUT_SENSOR_HEAT_COMPENSATED_TEMPERATURE = BSEC_INPUT_TEMPERATURE - \text{function}(\text{sensor operation mode, sensor supply voltage}) - BSEC_INPUT_HEATSOURCE$ ▶ other solutions: $BSEC_OUTPUT_SENSOR_HEAT_COMPENSATED_TEMPERATURE = BSEC_INPUT_TEMPERATURE - \text{function}(\text{sensor operation mode, sensor supply voltage})$ <p>The self-heating in operation mode BSEC_SAMPLE_RATE_ULP is negligible. The self-heating in operation mode BSEC_SAMPLE_RATE_LP is supported for 1.8V by default (no config file required). If the BME680 sensor supply voltage is 3.3V, the IoT_LP_3_3V.config shall be used.</p>

Enumerator

BSEC_OUTPUT_SENSOR_HEAT_COMPENSATED_HUMIDITY	<p>Sensor heat compensated humidity [%]. Relative measured by BME680 which is compensated for the influence of sensor (heater) in %.</p> <p>It converts the BSEC_INPUT_HUMIDITY from temperature BSEC_INPUT_TEMPERATURE to temperature BSEC_OUTPUT_SENSOR_HEAT_COMPENSATED_TEMPERATURE.</p> <p>Note</p> <p>IAQ solution: If BSEC_INPUT_HEATSOURCE is used for device specific temperature compensation, it will be effective for BSEC_OUTPUT_SENSOR_HEAT_COMPENSATED_HUMIDITY too.</p>
---	--

4.1.4 Defines

4.1.4.1 BSEC_MAX_PHYSICAL_SENSOR

```
#define BSEC_MAX_PHYSICAL_SENSOR (5)
```

Number of physical sensors that need allocated space before calling [bsec_update_subscription\(\)](#)

4.1.4.2 BSEC_MAX_PROPERTY_BLOB_SIZE

```
#define BSEC_MAX_PROPERTY_BLOB_SIZE (352)
```

Maximum size (in bytes) of the data blobs returned by [bsec_get_state\(\)](#) and [bsec_get_configuration\(\)](#)

4.1.4.3 BSEC_NUMBER_OUTPUTS

```
#define BSEC_NUMBER_OUTPUTS (10)
```

Number of outputs, depending on solution

4.1.4.4 BSEC_OUTPUT_INCLUDED

```
#define BSEC_OUTPUT_INCLUDED (13947)
```

bitfield that indicates which outputs are included in the solution

4.1.4.5 BSEC_PROCESS_GAS

```
#define BSEC_PROCESS_GAS (1 << (BSEC_INPUT_GASRESISTOR-1))
```

process_data bitfield constant for gas sensor

See also

[bsec_bme_settings_t](#)

4.1.4.6 BSEC_PROCESS_HUMIDITY

```
#define BSEC_PROCESS_HUMIDITY (1 << (BSEC_INPUT_HUMIDITY-1))
```

process_data bitfield constant for humidity

See also

[bsec_bme_settings_t](#)

4.1.4.7 BSEC_PROCESS_PRESSURE

```
#define BSEC_PROCESS_PRESSURE (1 << (BSEC_INPUT_PRESSURE-1))
```

process_data bitfield constant for pressure

See also

[bsec_bme_settings_t](#)

4.1.4.8 BSEC_PROCESS_TEMPERATURE

```
#define BSEC_PROCESS_TEMPERATURE (1 << (BSEC_INPUT_TEMPERATURE-1))
```

process_data bitfield constant for temperature

See also

[bsec_bme_settings_t](#)

4.1.4.9 BSEC_SAMPLE_RATE_DISABLED

```
#define BSEC_SAMPLE_RATE_DISABLED (65535.0f)
```

Sample rate of a disabled sensor

4.1.4.10 BSEC_SAMPLE_RATE_LP

```
#define BSEC_SAMPLE_RATE_LP (0.33333f)
```

Sample rate in case of Low Power Mode

4.1.4.11 BSEC_SAMPLE_RATE_ULP

```
#define BSEC_SAMPLE_RATE_ULP (0.0033333f)
```

Sample rate in case of Ultra Low Power Mode

4.1.4.12 BSEC_STRUCT_NAME

```
#define BSEC_STRUCT_NAME Bsec
```

Internal struct name

5 Data Structure Documentation

5.1 bsec_bme_settings_t Struct Reference

Data Fields

- ▶ `int64_t next_call`
Time stamp of the next call of the sensor_control.
- ▶ `uint32_t process_data`
Bit field describing which data is to be passed to `bsec_do_steps()`
- ▶ `uint16_t heater_temperature`
Heating temperature [degrees Celsius].
- ▶ `uint16_t heating_duration`
Heating duration [ms].
- ▶ `uint8_t run_gas`
Enable gas measurements [0/1].
- ▶ `uint8_t pressure_oversampling`
Pressure oversampling settings [0-5].
- ▶ `uint8_t temperature_oversampling`
Temperature oversampling settings [0-5].
- ▶ `uint8_t humidity_oversampling`
Humidity oversampling settings [0-5].
- ▶ `uint8_t trigger_measurement`
Trigger a forced measurement with these settings now [0/1].

5.1.1 Detailed Description

Structure returned by `bsec_sensor_control()` to configure BMExxx sensor.

This structure contains settings that must be used to configure the BMExxx to perform a forced-mode measurement. A measurement should only be executed if `bsec_bme_settings_t::trigger_measurement` is 1. If so, the oversampling settings for temperature, humidity, and pressure should be set to the provided settings provided in `bsec_bme_settings_t::temperature_oversampling`, `bsec_bme_settings_t::humidity_oversampling`, and `bsec_bme_settings_t::pressure_oversampling`, respectively.

In case of `bsec_bme_settings_t::run_gas` = 1, the gas sensor must be enabled with the provided `bsec_bme_settings_t::heater_temperature` and `bsec_bme_settings_t::heating_duration` settings.

5.1.2 Field Documentation

5.1.2.1 heater_temperature

`uint16_t bsec_bme_settings_t::heater_temperature`

Heating temperature [degrees Celsius].

5.1.2.2 heating_duration

`uint16_t bsec_bme_settings_t::heating_duration`

Heating duration [ms].

5.1.2.3 humidity_oversampling

`uint8_t bsec_bme_settings_t::humidity_oversampling`

Humidity oversampling settings [0-5].

5.1.2.4 next_call

`int64_t bsec_bme_settings_t::next_call`

Time stamp of the next call of the sensor_control.

5.1.2.5 pressure_oversampling

`uint8_t bsec_bme_settings_t::pressure_oversampling`

Pressure oversampling settings [0-5].

5.1.2.6 process_data

`uint32_t bsec_bme_settings_t::process_data`

Bit field describing which data is to be passed to [bsec_do_steps\(\)](#)

See also

`BSEC_PROCESS_*`

5.1.2.7 run_gas

`uint8_t bsec_bme_settings_t::run_gas`

Enable gas measurements [0/1].

5.1.2.8 temperature_oversampling

uint8_t bsec_bme_settings_t::temperature_oversampling

Temperature oversampling settings [0-5].

5.1.2.9 trigger_measurement

uint8_t bsec_bme_settings_t::trigger_measurement

Trigger a forced measurement with these settings now [0/1].

5.2 bsec_input_t Struct Reference

Data Fields

- ▶ int64_t [time_stamp](#)
Time stamp in nanosecond resolution [ns].
- ▶ float [signal](#)
Signal sample in the unit defined for the respective sensor_id.
- ▶ uint8_t [signal_dimensions](#)
Signal dimensions (reserved for future use, shall be set to 1)
- ▶ uint8_t [sensor_id](#)
Identifier of physical sensor.

5.2.1 Detailed Description

Structure describing an input sample to the library.

Each input sample is provided to BSEC as an element in a struct array of this type. Timestamps must be provided in nanosecond resolution. Moreover, duplicate timestamps for subsequent samples are not allowed and will result in an error code being returned from [bsec_do_steps\(\)](#).

The meaning unit of the signal field are determined by the [bsec_input_t::sensor_id](#) field content. Possible [bsec_input_t::sensor_id](#) values and their meaning are described in [bsec_physical_sensor_t](#).

See also

[bsec_physical_sensor_t](#)

5.2.2 Field Documentation

5.2.2.1 sensor_id

uint8_t bsec_input_t::sensor_id

Identifier of physical sensor.

See also

[bsec_physical_sensor_t](#)

5.2.2.2 signal

`float bsec_input_t::signal`

Signal sample in the unit defined for the respective `sensor_id`.

See also

[bsec_physical_sensor_t](#)

5.2.2.3 signal_dimensions

`uint8_t bsec_input_t::signal_dimensions`

Signal dimensions (reserved for future use, shall be set to 1)

5.2.2.4 time_stamp

`int64_t bsec_input_t::time_stamp`

Time stamp in nanosecond resolution [ns].

Timestamps must be provided as non-repeating and increasing values. They can have their 0-points at system start or at a defined wall-clock time (e.g., 01-Jan-1970 00:00:00)

5.3 bsec_output_t Struct Reference

Data Fields

► `int64_t` [time_stamp](#)

Time stamp in nanosecond resolution as provided as input [ns].

► `float` [signal](#)

Signal sample in the unit defined for the respective [bsec_output_t::sensor_id](#).

► `uint8_t` [signal_dimensions](#)

Signal dimensions (reserved for future use, shall be set to 1)

► `uint8_t` [sensor_id](#)

Identifier of virtual sensor.

► `uint8_t` [accuracy](#)

Accuracy status 0-4.

5.3.1 Detailed Description

Structure describing an output sample of the library.

Each output sample is returned from BSEC by populating the element of a struct array of this type. The contents of the signal field is defined by the supplied `bsec_output_t::sensor_id`. Possible output `bsec_output_t::sensor_id` values are defined in `bsec_virtual_sensor_t`.

See also

[bsec_virtual_sensor_t](#)

5.3.2 Field Documentation

5.3.2.1 accuracy

`uint8_t bsec_output_t::accuracy`

Accuracy status 0-4.

Some virtual sensors provide a value in the accuracy field. If this is the case, the meaning of the field is as follows:

Name	Value	Accuracy description
UNRELIABLE	0	Sensor data is unreliable, the sensor must be calibrated
LOW_ACCURACY	1	Low accuracy, sensor should be calibrated
MEDIUM_ACCURACY	2	Medium accuracy, sensor calibration may improve performance
HIGH_ACCURACY	3	High accuracy

For example:

- Ambient temperature accuracy is derived from change in the temperature in 1 minute.

Virtual sensor	Value	Accuracy description
Ambient temperature	0	The difference in ambient temperature is greater than 4 degree in one minute
	1	The difference in ambient temperature is less than 4 degree in one minute
	2	The difference in ambient temperature is less than 3 degree in one minute
	3	The difference in ambient temperature is less than 2 degree in one minute

- IAQ accuracy indicator will notify the user when she/he should initiate a calibration process. Calibration is performed automatically in the background if the sensor is exposed to clean and polluted air for approximately 30 minutes each.

Virtual sensor	Value	Accuracy description
IAQ	0	The sensor is not yet stabilized or in a run-in status
	1	Calibration required
	2	Not applicable
	3	Calibration is done, now IAQ estimate is highly accurate

5.3.2.2 sensor_id

`uint8_t bsec_output_t::sensor_id`

Identifier of virtual sensor.

See also

[bsec_virtual_sensor_t](#)

5.3.2.3 signal

`float bsec_output_t::signal`

Signal sample in the unit defined for the respective [bsec_output_t::sensor_id](#).

See also

[bsec_virtual_sensor_t](#)

5.3.2.4 signal_dimensions

`uint8_t bsec_output_t::signal_dimensions`

Signal dimensions (reserved for future use, shall be set to 1)

5.3.2.5 time_stamp

`int64_t bsec_output_t::time_stamp`

Time stamp in nanosecond resolution as provided as input [ns].

5.4 bsec_sensor_configuration_t Struct Reference

Data Fields

- ▶ float [sample_rate](#)
Sample rate of the virtual or physical sensor in Hertz [Hz].
- ▶ uint8_t [sensor_id](#)
Identifier of the virtual or physical sensor.

5.4.1 Detailed Description

Structure describing sample rate of physical/virtual sensors.

This structure is used together with [bsec_update_subscription\(\)](#) to enable BSEC outputs and to retrieve information about the sample rates used for BSEC inputs.

5.4.2 Field Documentation

5.4.2.1 sample_rate

`float bsec_sensor_configuration_t::sample_rate`

Sample rate of the virtual or physical sensor in Hertz [Hz].

Only supported sample rates are allowed.

5.4.2.2 sensor_id

`uint8_t bsec_sensor_configuration_t::sensor_id`

Identifier of the virtual or physical sensor.

The meaning of this field changes depending on whether the structs are as the `requested_virtual_sensors` argument to [bsec_update_subscription\(\)](#) or as the `required_sensor_settings` argument.

bsec_update_subscription() argument	sensor_id field interpretation
<code>requested_virtual_sensors</code>	bsec_virtual_sensor_t
<code>required_sensor_settings</code>	bsec_physical_sensor_t

See also

[bsec_physical_sensor_t](#)
[bsec_virtual_sensor_t](#)

5.5 bsec_version_t Struct Reference

Data Fields

- ▶ `uint8_t major`
Major version.
- ▶ `uint8_t minor`
Minor version.
- ▶ `uint8_t major_bugfix`
Major bug fix version.
- ▶ `uint8_t minor_bugfix`
Minor bug fix version.

5.5.1 Detailed Description

Structure containing the version information.

Please note that configuration and state strings are coded to a specific version and will not be accepted by other versions of BSEC.

5.5.2 Field Documentation

5.5.2.1 major

`uint8_t bsec_version_t::major`

Major version.

5.5.2.2 major_bugfix

`uint8_t bsec_version_t::major_bugfix`

Major bug fix version.

5.5.2.3 minor

`uint8_t bsec_version_t::minor`

Minor version.

5.5.2.4 minor_bugfix

`uint8_t bsec_version_t::minor_bugfix`

Minor bug fix version.