

Integration Guide

Bosch Software Environmental Cluster (BSEC)



Contents

1	BSEC Integration Guideline	4
1.1	Overview of BME Family Sensors	4
1.2	The Environmental Fusion Library BSEC	5
1.2.1	BSEC Library Solutions	5
1.2.2	BSEC Configuration Settings	5
1.2.3	Key Features	7
1.2.4	Supported Virtual Sensor Output Signals	8
1.3	Requirements for Integration	9
1.3.1	Hardware	9
1.3.2	Software Framework	9
1.3.3	Physical Input Sensor Signals	10
1.3.4	Build the Solution	10
2	How to integrate BSEC library	11
2.1	Prerequisites	11
2.2	Integration of BSEC Interfaces	11
3	FAQ	14
3.1	No Output From <code>bsec_do_steps()</code>	14
3.2	IAQ output does not change or accuracy remains 0	14
3.3	Error Codes and Corrective Measures	14
3.3.1	Errors Returned by <code>bsec_do_steps()</code>	14
3.3.2	Errors Returned by <code>bsec_update_subscription()</code>	17
3.3.3	Errors Returned by <code>bsec_set_configuration()</code> / <code>bsec_set_state()</code>	20
3.3.4	Errors Returned by <code>bsec_sensor_control()</code>	22
4	Module Documentation	23
4.1	BSEC Enumerations and Definitions	23
4.1.1	Enumerations	23
4.1.2	Defines	29
4.2	BSEC Standard Interfaces	32
4.2.1	Interface Functions	32
4.3	BSEC Multi-instance Interfaces	41
4.3.1	Interface Functions	41
5	Data Structure Documentation	47
5.1	<code>bsec_bme_settings_t</code> Struct Reference	47
5.1.1	Detailed Description	47
5.1.2	Field Documentation	48
5.2	<code>bsec_input_t</code> Struct Reference	49
5.2.1	Detailed Description	50
5.2.2	Field Documentation	50
5.3	<code>bsec_output_t</code> Struct Reference	51
5.3.1	Detailed Description	51

5.3.2	Field Documentation	51
5.4	bsec_sensor_configuration_t Struct Reference	53
5.4.1	Detailed Description	53
5.4.2	Field Documentation	53
5.5	bsec_version_t Struct Reference	54
5.5.1	Detailed Description	54
5.5.2	Field Documentation	54

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. Recent addition to the family has been BME688 sensor. The BME688 sensor hardware is the same as BME680 except that it can measure higher gas resistance. The sensors can be operated in different modes specified in supplied header files. 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 BME688 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 BME688 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 BME688 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 BME688 can detect Volatile Organic Compounds (VOCs), Volatile Sulfer Compounds(VSCs) and other gases such as carbon monoxide and hydrogen in the part per billion (ppb) range.

Additionally to all features of BME680, the BME688 has a gas scanner function. In standard configuration, the presence of VSCs is being detected as an indicator for e.g. bacteria growth. And the gas scanner can be customized with respect to sensitivity, selectivity, data rate and power consumption as well. The BME AI-Studio tool enables customers to train the BME688 gas scanner on their specific application, like in home appliances, IoT products or Smart Home.

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

- ▶ BME688 sensor (pressure, temperature, humidity and gas)
- ▶ Device with BME688 integrated
- ▶ Sensor driver API - Provide software interfaces to get compensated raw data from sensor via SPI/I2C interface
- ▶ BSEC fusion library - Provides fused sensor outputs and AI interpreter for classification of gas classes
- ▶ BME AI Studio - AI toolchain to develop, verify and deploy custom gas classification use cases
- ▶ *Optional*: Additional device sensors (i.e., temperature of other heat sources in the device or position sensors)

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

1.2.1 BSEC Library Solutions

Offered BSEC solutions are

Solution	Included features
SELECTIVITY	Gas estimations, Index for Air Quality, ambient temperature/humidity estimation, raw signals

Based on customer requests it is technically possible to further customize BSEC to meet specific customer demands.

1.2.2 BSEC Configuration Settings

BSEC offers the flexibility to configure the solution based on customer specific needs. The configuration can be loaded to BSEC via [bsec_set_configuration\(\)](#). The following settings can be configured

- ▶ Supply voltage of the sensor. The supply voltage influences the self-heating of the sensor.
 - ▶ 1.8V

- ▶ 3.3V
- ▶ Different power modes for the gas sensor and corresponding data rates are supported by the software solution:
 - ▶ Gas Scan mode (scan) is designed for interactive applications where selectivity of target gases are needed in addition in indoor air-quality monitoring. The standard heater profile has an update rate of 10.8 s and can be finetuned for use-cases using BME AI Studio.
 - ▶ Ultra low power (ULP) mode is designed for battery-powered and/or frequency-coupled devices over extended periods of time. This mode features an update rate of 300 seconds and an average current consumption of <0.1 mA
 - ▶ Low power (LP) mode that is designed for interactive applications where the air quality is tracked and observed at a higher update rate of 3 seconds with a current consumption of <1 mA
 - ▶ Continuous(CONT) mode provides an update rate if 1 Hz and shall only be used short-term for use cases that incorporate very fast events or stimulus. This mode has an average current consumption of <12 mA
- ▶ The history BSEC considers for the automatic background calibration of the IAQ is in days. That means changes in this time period will influence the IAQ value.
 - ▶ 4days, means BSEC will consider the last 4 days of operation for the automatic background calibration.
 - ▶ 28days, means BSEC will consider the last 28 days of operation for the automatic background calibration.

Configurations for BSEC library

The following configuration sets are available for BSEC library:

Configuration	Sensor variant	Supply voltage	Max data rate (IAQ)	Calibration time (IAQ)	Gas estimate output class
bme688_sel_↔ 33v_3s_28d	BME688	3.3 V	3s	28 days	H2S/NonH2S
bme688_sel_↔ 33v_3s_4d	BME688	3.3 V	3s	4 days	H2S/NonH2S
bme688_sel_↔ 18v_3s_28d	BME688	1.8V	3s	28 days	H2S/NonH2S
bme688_sel_↔ 18v_3s_4d	BME688	1.8V	3s	4 days	H2S/NonH2S
bme688_sel_↔ 33v_300s_28d	BME688	3.3 V	300s	28 days	H2S/NonH2S
bme688_sel_↔ 33v_300s_4d	BME688	3.3 V	300s	4 days	H2S/NonH2S
bme688_sel_↔ 18v_300s_28d	BME688	1.8V	300s	28 days	H2S/NonH2S
bme688_sel_↔ 18v_300s_4d	BME688	1.8V	300s	4 days	H2S/NonH2S
bme680_iaq_↔ 33v_3s_28d	BME680	3.3 V	3s	28 days	-
bme680_iaq_↔ 33v_3s_4d	BME680	3.3 V	3s	4 days	-

Configuration	Sensor variant	Supply voltage	Max data rate (IAQ)	Calibration time (IAQ)	Gas estimate output class
bme680_iaq_↔ 18v_3s_28d	BME680	1.8V	3s	28 days	-
bme680_iaq_↔ 18v_3s_4d	BME680	1.8V	3s	4 days	-
bme680_iaq_↔ 33v_300s_28d	BME680	3.3 V	300s	28 days	-
bme680_iaq_↔ 33v_300s_4d	BME680	3.3 V	300s	4 days	-
bme680_iaq_↔ 18v_300s_28d	BME680	1.8V	300s	28 days	-
bme680_iaq_↔ 18v_300s_4d	BME680	1.8V	300s	4 days	-

Note:

- ▶ The above configuration offers either gas estimation outputs in selectivity mode or IAQ outputs in LP/ULP mode. IAQ and gas estimate outputs shall not be enabled at the same time.
- ▶ The built-in configuration settings in BSEC library when no external configuration file is used can be 'bme688↔_sel_18v_300s_4d' settings.
- ▶ It is mandatory to provide configuration setting for getting gas estimate outputs, else output will be all zeros.
- ▶ Users can update the settings for gas estimate outputs using BME AI-Studio. For a BME AI-Studio generated configuration file, the default settings for all outputs other than gas estimates are based on 'bme688_sel_18v↔_300s_28d' file settings.
- ▶ BSEC support a maximum of 4 classes for gas estimate outputs, However above mentioned configuration settings is defined for only two classes.

1.2.3 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 Index for Air Quality(IAQ) level outside the device
- ▶ Provide the probability of the detected target gas(eg: H2S, nonH2S) class

Selectivity Mode with BME688

- ▶ Generally, the idea of the Selectivity /gas scanner is to collect as much information of the present gas mixture as possible to use the measured sensor information together with a trained model to recognize certain gases or gas mixtures and probability of occurrence of target gases.
- ▶ Under selectivity mode of operation, the heater is operated under multiple temperature points and a maximum of 10 heater points is supported by the library.

Advantages

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

1.2.4 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	Min	Max	Unit	Acc ¹	Supporting Modes ²
BSEC_OUTPUT_GAS_ESTIMATE_1	0.00	1	probability ³	yes	scan
BSEC_OUTPUT_GAS_ESTIMATE_2	0.00	1	probability ³	yes	scan
BSEC_OUTPUT_GAS_ESTIMATE_3	0.00	1	probability ³	yes	scan
BSEC_OUTPUT_GAS_ESTIMATE_4	0.00	1	probability ³	yes	scan
BSEC_OUTPUT_RAW_PRESSURE	30000	110000	Pa	no	ULP,LP,CONT,scan
BSEC_OUTPUT_RAW_TEMPERATURE	-40	85	deg C	no	ULP,LP,CONT,scan
BSEC_OUTPUT_RAW_HUMIDITY	0	100	%	no	ULP,LP,CONT,scan
BSEC_OUTPUT_RAW_GAS	170	103000000	Ohm	no	ULP,LP,CONT,scan
BSEC_OUTPUT_RAW_GAS_INDEX	0	9		no	scan
BSEC_OUTPUT_IAQ	0	500		yes	ULP,LP,CONT
BSEC_OUTPUT_STATIC_IAQ	0	-		yes	ULP,LP,CONT
BSEC_OUTPUT_CO2_EQUIVALENT	400	-	ppm	yes	ULP,LP,CONT
BSEC_OUTPUT_BREATH_VOC_EQUIVALENT	0	1000	ppm	yes	ULP,LP,CONT
BSEC_OUTPUT_SENSOR_HEAT_COMPENSATED_TEMPERATURE	-45	85	deg C	no	ULP,LP,CONT
BSEC_OUTPUT_SENSOR_HEAT_COMPENSATED_HUMIDITY	0	100	%	no	ULP,LP,CONT
BSEC_OUTPUT_STABILIZATION_STATUS	False	True		no	ULP,LP,CONT
BSEC_OUTPUT_RUN_IN_STATUS	False	True		no	ULP,LP,CONT
BSEC_OUTPUT_GAS_PERCENTAGE	0	100	%	yes	ULP,LP,CONT

To achieve best gas sensor performance, the user shall not switch between different modes during the lifetime of a given sensor.

¹ Accuracy status available (see [bsec_output_t::accuracy](#)).

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

² The sample rate of ULP, LP, CONT and scan mode are 1/300Hz, 1/3Hz, 1Hz and 1/18Hz respectively.

³ Probability denotes the range 0.00-1

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 family 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.

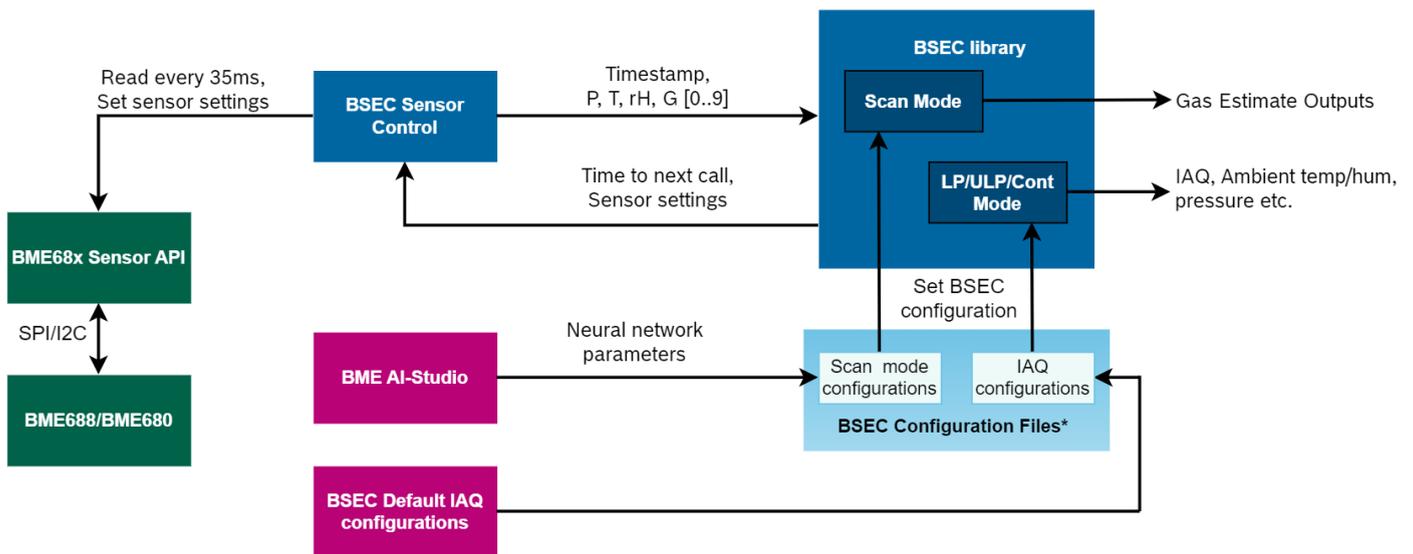


Figure 1.1: BSEC Overview

* Refer [Configurations for BSEC library](#) for modes and configurations supported by the sensors.

Typical durations for the "Sleep until measurement is finished" are 0.190 seconds for LP mode, 2 seconds for ULP mode, 0.9 seconds for CONT mode and 0.190 seconds for scan mode. Typical durations for the "Sleep until next time `bsec_sensor_control()` asked to be called" are 2.8 seconds for LP mode, 298 seconds for ULP mode, 0.1 seconds for CONT mode, 35ms for ON scan cycle of scan mode and sleep duration for OFF scan cycle for scan

mode. Sleep duration for scan mode is as configured in case of the selected temperature profiles. Typically sleep duration is the multiplication of total profile duration and number of OFF scan cycle. Sleep duration of standard heater profile (Heater Profile #354 with 5/10 duty cycle) is 107.8 seconds.

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. In selectivity mode, more than 1 data could be read from sensor using 3 member FIFO, which could be passed with same timestamp but with different [BSEC_INPUT_PROFILE_PART](#) updated.

1.3.3 Physical Input Sensor Signals

BSEC is designed to be used exclusively together with sensors of the BMExxx family, BME680/BME688.

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 integration code. The library includes the following header files which need to be included along with BSEC library package.

Header file	Description
bsec_datatypes.h	Data types and defines used by interface functions
bsec_interface.h/bsec_interface_multi.h	Declaration of standard interface or multi-interface functions

2 How to integrate BSEC library

This section describes how you can integrate BSEC library into your own environment and prerequisites for its integration into frameworks on supported platforms. Furthermore, a brief explanation of sequence of API calls with example code is also provided.

2.1 Prerequisites

First of all, you require a BME688/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. Of course, you will also need a development environment for the MCU of your choice. Key software tools for BME688 are listed below-

Software components	Type	Function	Link
BME68x sensor API	C code	Provide software interfaces to get compensated raw data from sensor via SPI/I2C interface	BME68x -sensor-API
BSEC	C static library	Provides fused sensor outputs and AI interpreter for classification of gas classes	BME688 -Software
BME AI Studio	PC Application	AI toolchain to develop, verify and deploy custom gas classification use cases	BME688 -Software

2.2 Integration of BSEC Interfaces

Integration of BSEC Standard Interface

Integration of BSEC library requires following the below mentioned steps-

Steps	Function	APIs to be called
Initialization of BME68x sensor	Initialization of sensor	bme68x_init() Refer BME68x -sensor-API
Initialization of the BSEC library	Initialization of library Update configuration settings (optional) Restore state of library (optional)	bsec_init() bsec_set_configuration() bsec_set_state()
Subscribe outputs	Enable library outputs with specified mode	bsec_update_subscription()
Signal processing with BSEC library	Retrieve BME68x sensor instructions Main signal processing function	bsec_sensor_control() bsec_do_steps()
Retrieval of BSEC library state on power cycle	Retrieve the current internal library state (optional)	bsec_get_state()



The following sequence diagram gives the sequence of standard interface API calls.

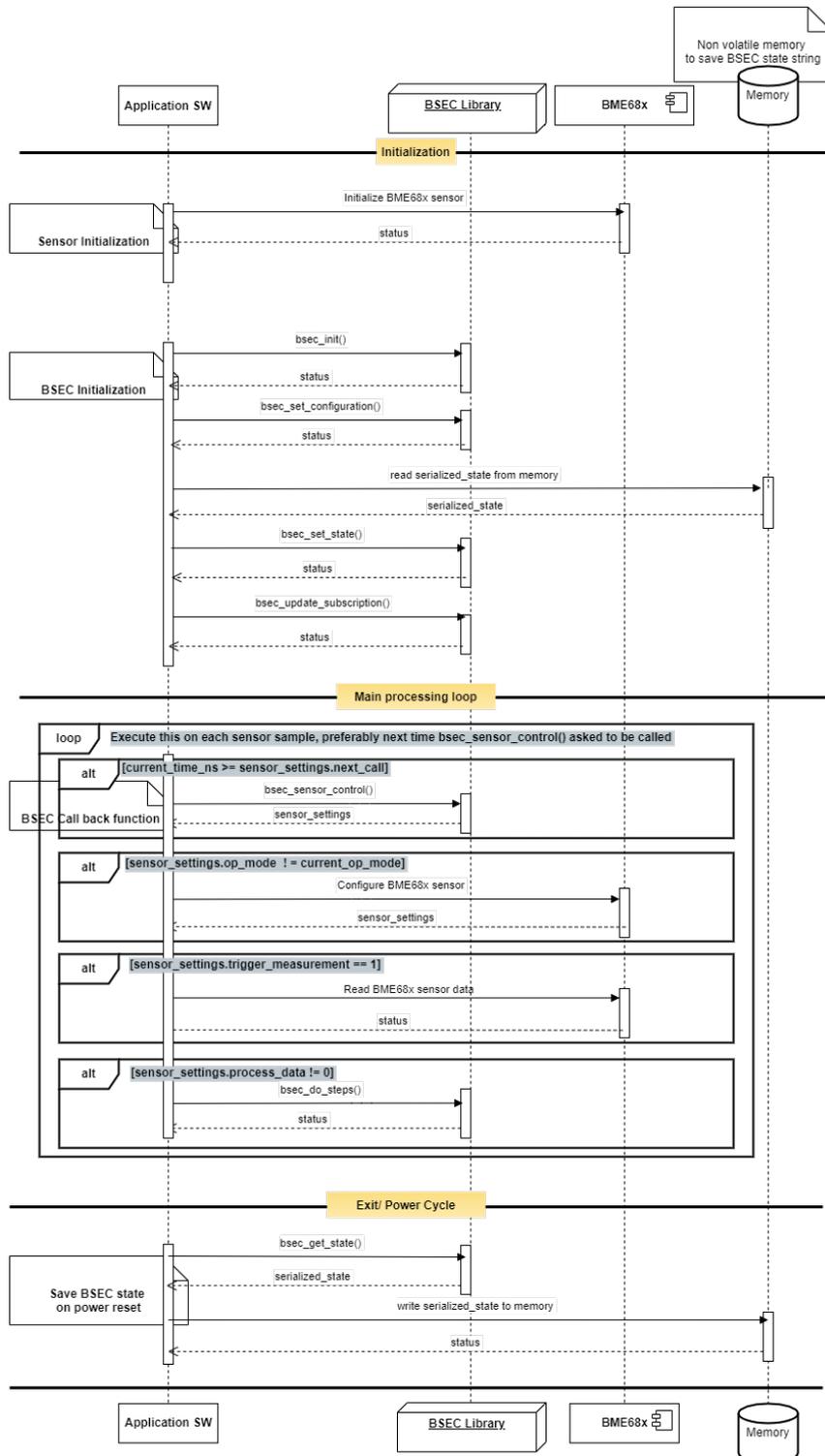


Figure 2.1: Sequence Diagram

Note: Please refer to `bsec_interface.h` for detailed description, interface definitions and sample usage of above-mentioned APIs

Integration of BSEC Multi-Instance Interface

For using more than one BME68x sensor, each sensor must be initialized with BSEC library using multi-instance support. The multi-instance integration requires following the below mentioned steps-

Steps	Function	APIs to be called
Initialization of BME68x sensor	Initialization of sensor	bme68x_init() Refer BME68x -senor-API
Initialization of the BSEC library instance	Initialization of library instance Update instance configuration settings (optional) Restore the instance state of the library (optional)	bsec_init_m() bsec_set_configuration_m() bsec_set_state_m()
Subscribe outputs for the instance	Enable library instance outputs with specified mode	bsec_update_subscription_m()
Signal processing with BSEC library instance	Retrieve BME68x sensor instructions for the instance Main signal processing function of BSEC instance	bsec_sensor_control_m() bsec_do_steps_m()
Retrieval of BSEC library instance state on power cycle	Retrieve the current internal library instance state (optional)	bsec_get_state_m()

Note: Please refer to `bsec_interface_multi.h` for detailed description, interface definitions and sample usage of above-mentioned APIs

Example code for BSEC Interfaces

Please find abstracted example code using [Bosch-BSEC2-Library](#) in <https://github.com/BoschSensortec/Bosch-BSEC2-Library/tree/master/examples>

Please refer to <https://github.com/BoschSensortec/Bosch-BSEC2-Library/blob/master/README.md> for prerequisites and instructions for using those examples in Arduino.

A brief description of above-mentioned example codes is given below.

Example code	Description
basic.ino	This is an example for illustrating the basic BSEC virtual outputs
basic_config_state.ino	This is an example for illustrating the BSEC feature using desired configuration setting
bme68x_demo_sample.ino	This demonstrator application running on an x8 board has the feature of sensor data logging and BSEC algorithm illustrated.

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 6.25% 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 106.25% of 3 s which is 3.1875 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

This chapter will give possible possible correction methods in order to fix the issues mentioned below. An overview of the error codes is given in `bsec_library_return_t`.

3.3.1 Errors Returned by `bsec_do_steps()`

3.3.1.1 BSEC_E_DOSTEPS_INVALIDINPUT

Possible reasons:

- ▶ General description: `BSEC_E_DOSTEPS_INVALIDINPUT`
- ▶ 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

Possible reasons:

- ▶ General description: [BSEC_E_DOSTEPS_VALUELIMITS](#)
- ▶ 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 (-40 to 85)
 - ▶ HUMIDITY (0 to 100)
 - ▶ PRESSURE (300 to 110000)
 - ▶ GASRESISTOR (170 to 103000000)
 - ▶ PROFILE_PART (0 to 9)
 - ▶ Other Sensors (-3.4028e+38 to +3.4028e+38)

3.3.1.3 BSEC_E_DOSTEPS_DUPLICATEINPUT

Possible reasons:

- ▶ General description: [BSEC_E_DOSTEPS_DUPLICATEINPUT](#)
- ▶ 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

Possible reasons:

- ▶ General description: [BSEC_I_DOSTEPS_NOOUTPUTSRETURNABLE](#)
- ▶ 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

Possible reasons:

- ▶ General description: [BSEC_W_DOSTEPS_EXCESSOUTPUTS](#)
- ▶ 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

Possible reasons:

- ▶ General description: [BSEC_W_DOSTEPS_TSINTRADIFFOUTOFRANGE](#)
- ▶ 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

Possible reasons:

- ▶ General description: [BSEC_E_SU_WRONGDATARATE](#)
- ▶ Virtual sensors are requested with a sampling rate which is not allowed, e.g. 100 Hz.

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](#).

3.3.2.2 BSEC_E_SU_SAMPLERATELIMITS

Possible reasons:

- ▶ General description: [BSEC_E_SU_SAMPLERATELIMITS](#)
- ▶ 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

Possible reasons:

- ▶ General description: [BSEC_E_SU_DUPLICATEGATE](#)
- ▶ 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

Possible reasons:

- ▶ General description: [BSEC_E_SU_INVALIDSAMPLERATE](#)
- ▶ 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

Possible reasons:

- ▶ General description: [BSEC_E_SU_GATECOUNTEXCEEDSARRAY](#)
- ▶ 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

Possible reasons:

- ▶ General description: [BSEC_E_SU_SAMPLINTVLINTEGERMULT](#)
- ▶ 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

Possible reasons:

- ▶ General description: [BSEC_E_SU_MULTGASSAMPLINTVL](#)
- ▶ 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

Possible reasons:

- ▶ General description: [BSEC_E_SU_HIGHHEATERONDURATION](#)
- ▶ 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

Possible reasons:

- ▶ General description: [BSEC_W_SU_UNKNOWNOUTPUTGATE](#)
- ▶ 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

Possible reasons:

- ▶ General description: [BSEC_I_SU_SUBSCRIBEDOUTPUTGATES](#)
- ▶ 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.

3.3.2.11 BSEC_W_SU_MODINNOULP

Possible reasons:

- ▶ General description: [BSEC_W_SU_MODINNOULP](#)
- ▶ Triggering measurement on demand (MOD) in non-ULP mode

Correction methods:

- ▶ Ensure that sensors are running in ULP mode or enable ULP mode using `bsec_update_subscription()` before triggering MOD

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

3.3.3.1 BSEC_E_CONFIG_VERSIONMISMATCH

Possible reasons:

- ▶ General description: [BSEC_E_CONFIG_VERSIONMISMATCH](#)
- ▶ 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

Possible reasons:

- ▶ General description: [BSEC_E_CONFIG_FEATUREMISMATCH](#)
- ▶ 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

Possible reasons:

- ▶ General description: [BSEC_E_CONFIG_CRCMISMATCH](#)
- ▶ 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

Possible reasons:

- ▶ General description: [BSEC_E_CONFIG_EMPTY](#)
- ▶ 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

Possible reasons:

- ▶ General description: [BSEC_E_CONFIG_INSUFFICIENTWORKBUFFER](#)
- ▶ 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

Possible reasons:

- ▶ General description: `BSEC_E_CONFIG_INVALIDSTRINGSIZE`
- ▶ 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

Possible reasons:

- ▶ General description: `BSEC_E_CONFIG_INSUFFICIENTBUFFER`
- ▶ 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

Possible reasons:

- ▶ General description: `BSEC_W_SC_CALL_TIMING_VIOLATION`
- ▶ 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 Enumerations and Definitions

4.1.1 Enumerations

4.1.1.1 bsec_library_return_t

enum `bsec_library_return_t`

Enumeration for function return codes.

Enumerator

<code>BSEC_OK</code>	Function execution successful
<code>BSEC_E_DOSTEPS_INVALIDINPUT</code>	Input (physical) sensor id passed to <code>bsec_do_steps()</code> is not in the valid range or not valid for requested virtual sensor
<code>BSEC_E_DOSTEPS_VALUELIMITS</code>	Value of input (physical) sensor signal passed to <code>bsec_do_steps()</code> is not in the valid range
<code>BSEC_W_DOSTEPS_TSINTRADIFFOUTOFRANGE</code>	Past timestamps passed to <code>bsec_do_steps()</code>
<code>BSEC_E_DOSTEPS_DUPLICATEINPUT</code>	Duplicate input (physical) sensor ids passed as input to <code>bsec_do_steps()</code>
<code>BSEC_I_DOSTEPS_NOOUTPUTSRETURNABLE</code>	No memory allocated to hold return values from <code>bsec_do_steps()</code> , i.e., <code>n_outputs == 0</code>
<code>BSEC_W_DOSTEPS_EXCESSOUTPUTS</code>	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>
<code>BSEC_W_DOSTEPS_GASINDEXMISS</code>	Gas index not provided to <code>bsec_do_steps()</code>
<code>BSEC_E_SU_WRONGDATARATE</code>	The <code>sample_rate</code> of the requested output (virtual) sensor passed to <code>bsec_update_subscription()</code> is zero
<code>BSEC_E_SU_SAMPLERATELIMITS</code>	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
<code>BSEC_E_SU_DUPLICATEGATE</code>	Duplicate output (virtual) sensor ids requested through <code>bsec_update_subscription()</code>
<code>BSEC_E_SU_INVALIDSAMPLERATE</code>	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

Enumerator

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</code> <code>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
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 <code>bsec_update_subscription()</code> is not in the valid range; e.g., <code>n_requested_virtual_sensors</code> > actual number of output (virtual) sensors requested
BSEC_W_SU_MODINNOULP	ULP plus can not be requested in non-ulp mode
BSEC_I_SU_SUBSCRIBEDOUTPUTGATES	No output (virtual) sensor data were requested via <code>bsec_update_subscription()</code>
BSEC_I_SU_GASESTIMATEPRECEDENCE	<code>GAS_ESTIMATE</code> is suscribed and take precedence over other requested outputs
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 <code>serialized_[settings/state]</code> passed to <code>bsec_set_[configuration/state]()</code> does not match with current version
BSEC_E_CONFIG_FEATUREMISMATCH	Enabled features encoded in <code>serialized_[settings/state]</code> passed to <code>bsec_set_[configuration/state]()</code> does not match with current library implementation
BSEC_E_CONFIG_CRCMISMATCH	<code>serialized_[settings/state]</code> 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 to short to be valid
BSEC_E_CONFIG_INSUFFICIENTWORKBUFFER	Provided <code>work_buffer</code> is not large enough to hold the desired string
BSEC_E_CONFIG_INVALIDSTRINGSIZE	String size encoded in <code>configuration/state</code> 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

Enumerator

BSEC_E_SET_INVALIDCHANNELIDENTIFIER	Internal error code, size of work buffer in setConfig must be set to BSEC_MAX_WORKBUFFER_SIZE
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
BSEC_W_SC_MODEXCEEDULPTIMELIMIT	ULP plus is not allowed because an ULP measurement just took or will take place
BSEC_W_SC_MODINSUFFICIENTWAITTIME	ULP plus is not allowed because not sufficient time passed since last ULP plus

4.1.1.2 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 BME68x [Pa].
BSEC_INPUT_HUMIDITY	Humidity sensor output of BME68x [%]. Note Relative humidity strongly depends on the temperature (it is measured at). It may require a conversion to the temperature outside of the device. See also bsec_virtual_sensor_t

Enumerator

BSEC_INPUT_TEMPERATURE	<p>Temperature sensor output of BME68x [degrees Celsius].</p> <p>Note</p> <p>The BME68x is factory trimmed, thus the temperature sensor of the BME68x 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	<p>Gas sensor resistance output of BME68x [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>
BSEC_INPUT_HEATSOURCE	<p>Additional input for device heat compensation. IAQ solution: The value is subtracted from BSEC_INPUT_TEMPERATURE to compute BSEC_OUTPUT_SENSOR_HEAT_COMPENSATED_TEMPERATURE. ALL solution: Generic heat source 1</p> <p>See also</p> <p>bsec_virtual_sensor_t</p>
BSEC_INPUT_DISABLE_BASELINE_TRACKER	<p>Additional input that disables baseline tracker. 0 - Normal, 1 - Event 1, 2 - Event 2</p>
BSEC_INPUT_PROFILE_PART	<p>Additional input that provides information about the state of the profile (1-9)</p>

4.1.1.3 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	<p>Indoor-air-quality estimate [0-500]. Indoor-air-quality (IAQ) gives an indication of the relative change in ambient TVOCs detected by BME68x.</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=50 corresponds to typical good air and IAQ=200 indicates typical polluted air.</p>
BSEC_OUTPUT_STATIC_IAQ	Unscaled indoor-air-quality estimate
BSEC_OUTPUT_CO2_EQUIVALENT	CO2 equivalent estimate [ppm]
BSEC_OUTPUT_BREATH_VOC_EQUIVALENT	Breath VOC concentration estimate [ppm]
BSEC_OUTPUT_RAW_TEMPERATURE	<p>Temperature sensor signal [degrees Celsius]. Temperature directly measured by BME68x in degree Celsius.</p> <p>Note</p> <p>This value is cross-influenced by the sensor heating and device specific heating.</p>
BSEC_OUTPUT_RAW_PRESSURE	Pressure sensor signal [Pa]. Pressure directly measured by the BME68x in Pa.
BSEC_OUTPUT_RAW_HUMIDITY	<p>Relative humidity sensor signal [%]. Relative humidity directly measured by the BME68x in %.</p> <p>Note</p> <p>This value is cross-influenced by the sensor heating and device specific heating.</p>
BSEC_OUTPUT_RAW_GAS	Gas sensor signal [Ohm]. Gas resistance measured directly by the BME68x 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).

Enumerator

<p>BSEC_OUTPUT_STABILIZATION_STATUS</p>	<p>Gas sensor stabilization status [boolean]. Indicates initial stabilization status of the gas sensor element: stabilization is ongoing (0) or stabilization is finished (1).</p>
<p>BSEC_OUTPUT_RUN_IN_STATUS</p>	<p>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).</p>
<p>BSEC_OUTPUT_SENSOR_HEAT_COMPENSATED_TEMPERATURE</p>	<p>Sensor heat compensated temperature [degrees Celsius]. Temperature measured by BME68x which is compensated for the influence of sensor (heater) in degree Celsius. 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 Celsius), 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: $BSEC_OUTPUT_SENSOR_HEAT_COMPENSATED_TEMPERATURE = BSEC_INPUT_TEMPERATURE - function(sensor\ operation\ mode, sensor\ supply\ voltage) - BSEC_INPUT_HEATSOURCE$ ▶ other solutions: $BSEC_OUTPUT_SENSOR_HEAT_COMPENSATED_TEMPERATURE = BSEC_INPUT_TEMPERATURE - function(sensor\ operation\ mode, sensor\ supply\ voltage)$ <p>The self-heating in operation mode <code>BSEC_SAMPLE_RATE_ULP</code> is negligible. The self-heating in operation mode <code>BSEC_SAMPLE_RATE_LP</code> is supported for 1.8V by default (no config file required). If the BME68x sensor supply voltage is 3.3V, the corresponding config file shall be used.</p>

Enumerator

<p>BSEC_OUTPUT_SENSOR_HEAT_COMPENSATED_HUMIDITY</p>	<p>Sensor heat compensated humidity [%]. Relative measured by BME68x 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>
<p>BSEC_OUTPUT_GAS_PERCENTAGE</p>	<p>Percentage of min and max filtered gas value [%]</p>
<p>BSEC_OUTPUT_GAS_ESTIMATE_1</p>	<p>Gas estimate output channel 1 [0-1]. The gas scan result is given in probability for each of the used gases. In standard scan mode, the probability of H2S and non-H2S is provided by the variables BSEC_OUTPUT_GAS_ESTIMATE_1 & BSEC_OUTPUT_GAS_ESTIMATE_2 respectively. A maximum of 4 classes can be used by configuring using BME AI-studio.</p>
<p>BSEC_OUTPUT_GAS_ESTIMATE_2</p>	<p>Gas estimate output channel 2 [0-1]</p>
<p>BSEC_OUTPUT_GAS_ESTIMATE_3</p>	<p>Gas estimate output channel 3 [0-1]</p>
<p>BSEC_OUTPUT_GAS_ESTIMATE_4</p>	<p>Gas estimate output channel 4 [0-1]</p>
<p>BSEC_OUTPUT_RAW_GAS_INDEX</p>	<p>Gas index cyclically running from 0 to heater_profile_length-1, range of heater profile length is from 1 to 10, default being 10</p>

4.1.2 Defines

4.1.2.1 BSEC_MAX_PHYSICAL_SENSOR

```
#define BSEC_MAX_PHYSICAL_SENSOR (8)
```

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

4.1.2.2 BSEC_MAX_PROPERTY_BLOB_SIZE

```
#define BSEC_MAX_PROPERTY_BLOB_SIZE (1974)
```

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

4.1.2.3 BSEC_MAX_STATE_BLOB_SIZE

```
#define BSEC_MAX_STATE_BLOB_SIZE (221)
```

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

4.1.2.4 BSEC_MAX_WORKBUFFER_SIZE

```
#define BSEC_MAX_WORKBUFFER_SIZE (4096)
```

Maximum size (in bytes) of the work buffer

4.1.2.5 BSEC_NUMBER_OUTPUTS

```
#define BSEC_NUMBER_OUTPUTS (19)
```

Number of outputs, depending on solution

4.1.2.6 BSEC_OUTPUT_INCLUDED

```
#define BSEC_OUTPUT_INCLUDED (66222575)
```

bitfield that indicates which outputs are included in the solution

4.1.2.7 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](#)

4.1.2.8 BSEC_PROCESS_HUMIDITY

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

process_data bitfield constant for humidity

See also

[bsec_bme_settings_t](#)

4.1.2.9 BSEC_PROCESS_PRESSURE

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

process_data bitfield constant for pressure

See also

[bsec_bme_settings_t](#)

4.1.2.10 BSEC_PROCESS_TEMPERATURE

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

process_data bitfield constant for temperature

See also

[bsec_bme_settings_t](#)

4.1.2.11 BSEC_SAMPLE_RATE_CONT

```
#define BSEC_SAMPLE_RATE_CONT (1.0f)
```

Sample rate in case of Continuous Mode

4.1.2.12 BSEC_SAMPLE_RATE_DISABLED

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

Sample rate of a disabled sensor

4.1.2.13 BSEC_SAMPLE_RATE_LP

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

Sample rate in case of Low Power Mode

4.1.2.14 BSEC_SAMPLE_RATE_SCAN

```
#define BSEC_SAMPLE_RATE_SCAN (0.055556f)
```

Sample rate in case of scan mode

4.1.2.15 BSEC_SAMPLE_RATE_ULP

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

Sample rate in case of Ultra Low Power Mode

4.1.2.16 BSEC_SAMPLE_RATE_ULP_MEASUREMENT_ON_DEMAND

```
#define BSEC_SAMPLE_RATE_ULP_MEASUREMENT_ON_DEMAND (0.0f)
```

Input value used to trigger an extra measurement (ULP plus)

4.1.2.17 BSEC_STRUCT_NAME

```
#define BSEC_STRUCT_NAME Bsec
```

Internal struct name

4.2 BSEC Standard Interfaces

Standard interfaces of BSEC signal processing library. These interfaces supports in interfacing single BME68x sensor with the BSEC library.

4.2.1 Interface Functions

4.2.1.1 `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 behavior.
- ▶ 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.
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:
                // Retrieve the IAQ results from output[i].signal
                // and do something with the data
                break;
            case BSEC_OUTPUT_STATIC_IAQ:
                // Retrieve the static IAQ results from output[i].signal
                // and do something with the data
                break;
        }
    }
}
```

4.2.1.2 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_WORKBUFFER_SIZE];
uint32_t n_work_buffer = BSEC_MAX_WORKBUFFER_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);
```

4.2.1.3 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_STATE_BLOB_SIZE](#) for allotting the required size.

Parameters

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

Returns

Zero when successful, otherwise an error code

```
// Example //

// Allocate variables
uint8_t serialized_state[BSEC_MAX_STATE_BLOB_SIZE];
uint32_t n_serialized_state_max = BSEC_MAX_STATE_BLOB_SIZE;
uint32_t n_serialized_state = BSEC_MAX_STATE_BLOB_SIZE;
uint8_t work_buffer_state[BSEC_MAX_WORKBUFFER_SIZE];
uint32_t n_work_buffer_size = BSEC_MAX_WORKBUFFER_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);
```

4.2.1.4 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	<i>bsec_version</i> _↔ <i>_p</i>	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.%d",version.major, version.minor, version.
    major_bugfix, version.minor_bugfix);
```

4.2.1.5 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();
```

4.2.1.6 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);
```

4.2.1.7 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

4.2.1.8 `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
in,out	<code>work_buffer</code>	Work buffer used to parse the blob
in	<code>n_work_buffer_size</code>	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_WORKBUFFER_SIZE];
uint32_t n_work_buffer = BSEC_MAX_WORKBUFFER_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);
```

4.2.1.9 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_STATE_BLOB_SIZE` for allotting the required size.

Parameters

in	<code>serialized_state</code>	States serialized to a binary blob
in	<code>n_serialized_state</code>	Size of the state blob
in,out	<code>work_buffer</code>	Work buffer used to parse the blob
in	<code>n_work_buffer_size</code>	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_WORKBUFFER_SIZE];
```

```
uint32_t n_work_buffer_size = BSEC_MAX_WORKBUFFER_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);
```

4.2.1.10 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;
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);
```

4.3 BSEC Multi-instance Interfaces

The multi-instance interface of BSEC signal processing library is used for interfacing multiple sensors with BSEC library.

4.3.1 Interface Functions

4.3.1.1 bsec_do_steps_m()

```
bsec_library_return_t bsec_do_steps_m (
    void * inst,
    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 library instance.

Processing of the input signals and returning of output samples for each instances of BSEC library is performed by `bsec_do_steps_m()`. `bsec_do_steps_m()` processes multiple instances of BSEC library similar to how `bsec_do_steps()` handles single instance.

Parameters

in,out	<i>inst</i>	Reference to the pointer containing the instance
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.
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

4.3.1.2 bsec_get_configuration_m()

```
bsec_library_return_t bsec_get_configuration_m (
    void * inst,
    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 instance configuration.

BSEC allows to retrieve the current configuration of the library instance using `bsec_get_configuration_m()`. In the same way as `bsec_get_configuration()`, this API returns a binary blob encoding the current configuration parameters of the library in a format compatible with `bsec_set_configuration_m()`.

Please use `BSEC_MAX_PROPERTY_BLOB_SIZE` for allotting the required size.

Parameters

in,out	<i>inst</i>	Reference to the pointer containing the instance
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

4.3.1.3 bsec_get_instance_size_m()

```
size_t bsec_get_instance_size_m (
    void )
```

Function that provides the size of the internal instance in bytes. To be used for allocating memory for struct `BSEC_STRUCT_NAME`.

Returns

Size of the internal instance in bytes

4.3.1.4 bsec_get_state_m()

```
bsec_library_return_t bsec_get_state_m (
    void * inst,
    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 instance state.

BSEC allows to retrieve the current states of all signal processing modules and the BSEC module of the library instance using `bsec_get_state_m()`. As done by `bsec_get_state()`, this allows a restart of the processing after a reboot of the system by calling `bsec_set_state_m()`.

Please use `BSEC_MAX_STATE_BLOB_SIZE` for allotting the required size.

Parameters

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

Returns

Zero when successful, otherwise an error code

4.3.1.5 bsec_get_version_m()

```
bsec_library_return_t bsec_get_version_m (
    void * inst,
    bsec_version_t * bsec_version_p )
```

Return the version information of BSEC library instance.

Parameters

in,out	<i>inst</i>	Reference to the pointer containing the instance
out	<i>bsec_version</i> _↔ <i>_p</i>	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](#)

4.3.1.6 bsec_init_m()

```
bsec_library_return_t bsec_init_m (
    void * inst )
```

Initialize the library instance.

Initialization and reset of BSEC library instance is performed by calling [bsec_init_m\(\)](#) as done with [bsec_init\(\)](#) for standard interface.

Parameters

in,out	<i>inst</i>	Reference to the pointer containing the instance
--------	-------------	--

Returns

Zero if successful, otherwise an error code

4.3.1.7 bsec_reset_output_m()

```
bsec_library_return_t bsec_reset_output_m (
    void * inst,
    uint8_t sensor_id )
```

Reset a particular virtual sensor output of the library instance.

This function allows specific virtual sensor outputs of each library instance to be reset. It processes in same way as [bsec_reset_output\(\)](#).

Parameters

in,out	<i>inst</i>	Reference to the pointer containing the instance
in	<i>sensor_id</i>	Virtual sensor to be reset

Returns

Zero when successful, otherwise an error code

4.3.1.8 bsec_sensor_control_m()

```
bsec_library_return_t bsec_sensor_control_m (
    void * inst,
    const int64_t time_stamp,
    bsec_bme_settings_t * sensor_settings )
```

Retrieve BMExxx sensor instructions for the library instance.

The [bsec_sensor_control_m\(\)](#) allows an easy way for the signal processing library to control the operation of the BME sensor which uses the corresponding BSEC library instance. Operation of [bsec_sensor_control_m\(\)](#) is similar to [bsec_sensor_control\(\)](#) except that former API supports multiples library instances.

Parameters

in,out	<i>inst</i>	Reference to the pointer containing the instance
in	<i>time_stamp</i>	Current timestamp in [ns]
out	<i>sensor_settings</i>	Settings to be passed to API to operate sensor at this time instance

Returns

Zero when successful, otherwise an error code

4.3.1.9 bsec_set_configuration_m()

```
bsec_library_return_t bsec_set_configuration_m (
    void * inst,
    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 of the library instance.

As done with [bsec_set_configuration\(\)](#), the initial configuration of BSEC library instance can be customized by [bsec_set_configuration_m\(\)](#). This is an optional step.

Please use [BSEC_MAX_PROPERTY_BLOB_SIZE](#) for allotting the required size.

Parameters

in,out	<i>inst</i>	Reference to the pointer containing the instance
in	<i>serialized_settings</i>	Settings serialized to a binary blob
in	<i>n_serialized_settings</i>	Size of the settings 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

4.3.1.10 bsec_set_state_m()

```
bsec_library_return_t bsec_set_state_m (
    void * inst,
    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 instance.

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

Please use [BSEC_MAX_STATE_BLOB_SIZE](#) for allotting the required size.

Parameters

in,out	<i>inst</i>	Reference to the pointer containing the instance
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

4.3.1.11 bsec_update_subscription_m()

```
bsec_library_return_t bsec_update_subscription_m (
    void * inst,
    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.

Like [bsec_update_subscription\(\)](#), [bsec_update_subscription_m\(\)](#) is used to instruct BSEC which of the processed output signals of the library instance are requested at which sample rates.

Parameters

in,out	<i>inst</i>	Reference to the pointer containing the instance
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](#)

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`
Heater temperature [degrees Celsius].
- ▶ `uint16_t heater_duration`
Heater duration [ms].
- ▶ `uint16_t heater_temperature_profile` [10]
Heater temperature profile [degrees Celsius].
- ▶ `uint16_t heater_duration_profile` [10]
Heater duration profile [ms].
- ▶ `uint8_t heater_profile_len`
Heater profile length [0-10].
- ▶ `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].
- ▶ `uint8_t op_mode`
Sensor operation mode [0/1].

5.1.1 Detailed Description

Structure returned by `bsec_sensor_control()` to configure BME68x sensor.

This structure contains settings that must be used to configure the BME68x 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_duration

```
uint16_t bsec_bme_settings_t::heater_duration
```

Heater duration [ms].

5.1.2.2 heater_duration_profile

```
uint16_t bsec_bme_settings_t::heater_duration_profile[10]
```

Heater duration profile [ms].

5.1.2.3 heater_profile_len

```
uint8_t bsec_bme_settings_t::heater_profile_len
```

Heater profile length [0-10].

5.1.2.4 heater_temperature

```
uint16_t bsec_bme_settings_t::heater_temperature
```

Heater temperature [degrees Celsius].

5.1.2.5 heater_temperature_profile

```
uint16_t bsec_bme_settings_t::heater_temperature_profile[10]
```

Heater temperature profile [degrees Celsius].

5.1.2.6 humidity_oversampling

```
uint8_t bsec_bme_settings_t::humidity_oversampling
```

Humidity oversampling settings [0-5].

5.1.2.7 next_call

```
int64_t bsec_bme_settings_t::next_call
```

Time stamp of the next call of the `sensor_control`.

5.1.2.8 op_mode

uint8_t bsec_bme_settings_t::op_mode

Sensor operation mode [0/1].

5.1.2.9 pressure_oversampling

uint8_t bsec_bme_settings_t::pressure_oversampling

Pressure oversampling settings [0-5].

5.1.2.10 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_GAS](#), [BSEC_PROCESS_TEMPERATURE](#), [BSEC_PROCESS_HUMIDITY](#), [BSEC_PROCESS_PRESSURE](#)

5.1.2.11 run_gas

uint8_t bsec_bme_settings_t::run_gas

Enable gas measurements [0/1].

5.1.2.12 temperature_oversampling

uint8_t bsec_bme_settings_t::temperature_oversampling

Temperature oversampling settings [0-5].

5.1.2.13 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 0-3.

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 0-3.

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

Name	Value	Accuracy description
MEDIUM_ACCURACY	2	Medium accuracy, sensor calibration may improve performance
HIGH_ACCURACY	3	High accuracy

For example:

- ▶ 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	Stabilization / run-in ongoing
	1	Low accuracy, to reach high accuracy(3), please expose sensor once to good air (e.g. outdoor air) and bad air (e.g. box with exhaled breath) for auto-trimming
	2	Medium accuracy: auto-trimming ongoing
	3	High accuracy

- ▶ Gas estimator accuracy indicator will notify the user when she/he gets valid gas prediction output.

Virtual sensor	Value	Accuracy description
GAS_ESTIMATE_x*	0	No valid gas estimate found - BSEC collecting gas features
	3	Valid gas estimate found - BSEC gas estimation prediction available

* GAS_ESTIMATE_1, GAS_ESTIMATE_2, GAS_ESTIMATE_3, GAS_ESTIMATE_4

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.