Integration Guide
Bosch Software Environmental Cluster (BSEC)

Contents

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 can detect gases by measuring their unique electric fingerprint and therefore distinguish different gas compositions. This enables endless new applications. 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 a broad range of gases and support number of usecases

- Bad breath or spoiled food detection based on the measurement of volatile sulfur compounds, which are an indicator of bacteria growth
- Detection of unusual gases and smells, which might indicate for instance a leakage or fire
- Diaper state detection, e.g. for baby care
- Early detection of bad smells
- Wildfire detection

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 BME system can be understood as a combination of the below mentioned system architecture components

- BME688sensor (pressure, temperature, humidity and gas)
- Device with BME688 sensor 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 classifcation 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

A BSEC solution can be chosen a from 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
IAQ$*$	Index for Air Quality, sensor heating compensated temperature/humidity, raw signals
SELECTIVITY	Selectivity to gas,index for Air Quality, ambient temperature/humidity estimation, raw signals

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

- Compensate influence of supply voltage. Two predefined configuration are available
  - 1.8V
  - 3.3V
- The maximum allowed time between two bsec_sensor_control calls.This help operating system to optimise system resources
  - In case of selectivity, maximum allowed time is determined by the heater profile. This is equivalent to minimum heater profile step duration.
  - 3s supports the ULP plus feature for the ULP mode
  - 300s allows the system to sleep for 300s for the ULP mode in order minimize the power consumption of the system, but does not support the ULP plus feature
- The history BSEC considers for the automatic background calibration of the IAQ 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.

Available BSEC configurations are

Configuration	Supply voltage of BME688	Maximum time between bsec_sensor_control() calls	Time considered for background calibration
bsec_sel_iaq_18v_4d	1.8V	As per heater profile	4 days
bsec_sel_iaq_33v_4d	3.3V	As per heater profile	4 days
generic_33v_300s_28d	3.3V	300s	28 days
generic_33v_300s_4d	3.3V	300s	4 days
generic_33v_3s_28d	3.3V	3s	28 days
generic_33v_3s_4d	3.3V	3s	4 days
generic_18v_300s_28d	1.8V	300s	28 days
generic_18v_300s_4d	1.8V	300s	4 days
generic_18v_3s_28d	1.8V	3s	28 days
generic_18v_3s_4d	1.8V	3s	4 days

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 indoor air quality (IAQ) level outside the device
- Provide probability of the detected target gas class(eg: H2S, non-H2S).Maximum of 4 classes

Selectivity with BME688

Selectivity to target gas classes will enable wide variety of use cases for customers.From a gas sensing perspective, selectivity is the ability to classify different gas compositions (with different combinations and concentrations of gases) that occur in different situations. Metal oxide layer of gas sensor operated in temperature profile creates unique fingerprint for gas compositions. Standard heater profile is as shown below figure is executed in sensor ASIC which consists of 10 heater temperature steps.Signal features interpreted by AI based software brings selectivity feature.

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	Unit/Type	Acc.? [1]	Inc.? [2]	Sampling rate [3]
BSEC_OUTPUT_GAS_ESTIMATE_1	probability	yes	SELECTIVITY	SEL
BSEC_OUTPUT_GAS_ESTIMATE_2	probability	yes	SELECTIVITY	SEL
BSEC_OUTPUT_GAS_ESTIMATE_3	probability	yes	SELECTIVITY	SEL
BSEC_OUTPUT_GAS_ESTIMATE_4	probability	yes	SELECTIVITY	SEL
BSEC_OUTPUT_RAW_PRESSURE	Pa	no	IAQ,SELECTIVITY	ULP, LP, HP, HTR
BSEC_OUTPUT_RAW_TEMPERATURE	deg C	no	IAQ,SELECTIVITY	ULP, LP, HP, HTR
BSEC_OUTPUT_RAW_HUMIDITY	%	no	IAQ,SELECTIVITY	ULP, LP, HP, HTR
BSEC_OUTPUT_RAW_GAS	Ohm	no	IAQ,SELECTIVITY	ULP, LP, HP, HTR
BSEC_OUTPUT_RAW_GAS_INDEX		no	SELECTIVITY	HTR
BSEC_OUTPUT_GAS_PERCENTAGE	%	no	IAQ,SELECTIVITY	ULP, LP, HP
BSEC_OUTPUT_COMPENSATED_GAS	Ohm	no	IAQ,SELECTIVITY	ULP, LP, HP
BSEC_OUTPUT_IAQ		yes	IAQ,SELECTIVITY	ULP, LP, HP
BSEC_OUTPUT_STATIC_IAQ		yes	IAQ,SELECTIVITY	ULP, LP, HP
BSEC_OUTPUT_CO2_EQUIVALENT	ppm	yes	IAQ,SELECTIVITY	ULP, LP, HP
BSEC_OUTPUT_BREATH_VOC_EQUIVALENT	ppb	yes	IAQ,SELECTIVITY	ULP, LP, HP
BSEC_OUTPUT_SENSOR_HEAT_COMPENSATED_TEMPERATURE	deg C	no	IAQ,SELECTIVITY	ULP, LP, HP
BSEC_OUTPUT_SENSOR_HEAT_COMPENSATED_HUMIDITY	%	no	IAQ,SELECTIVITY	ULP, LP, HP
BSEC_OUTPUT_STABILIZATION_STATUS	y/n	no	IAQ,SELECTIVITY	ULP, LP, HP
BSEC_OUTPUT_RUN_IN_STATUS	y/n	no	IAQ,SELECTIVITY	ULP, LP, HP

[1] Accuracy status available (see bsec_output_t::accuracy). [2] Included in solution. [3] The sample rates of modes ULP, LP, HP,HTR and SEL are (1/300, 1/3, 1,1/heater step duration, 1/scan duration) Hz respectively.

1.3 Requirements for Integration

1.3.1 Hardware

BSEC was specifically designed to work together with Bosch environmental sensor of the BME68x 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 BME688 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.

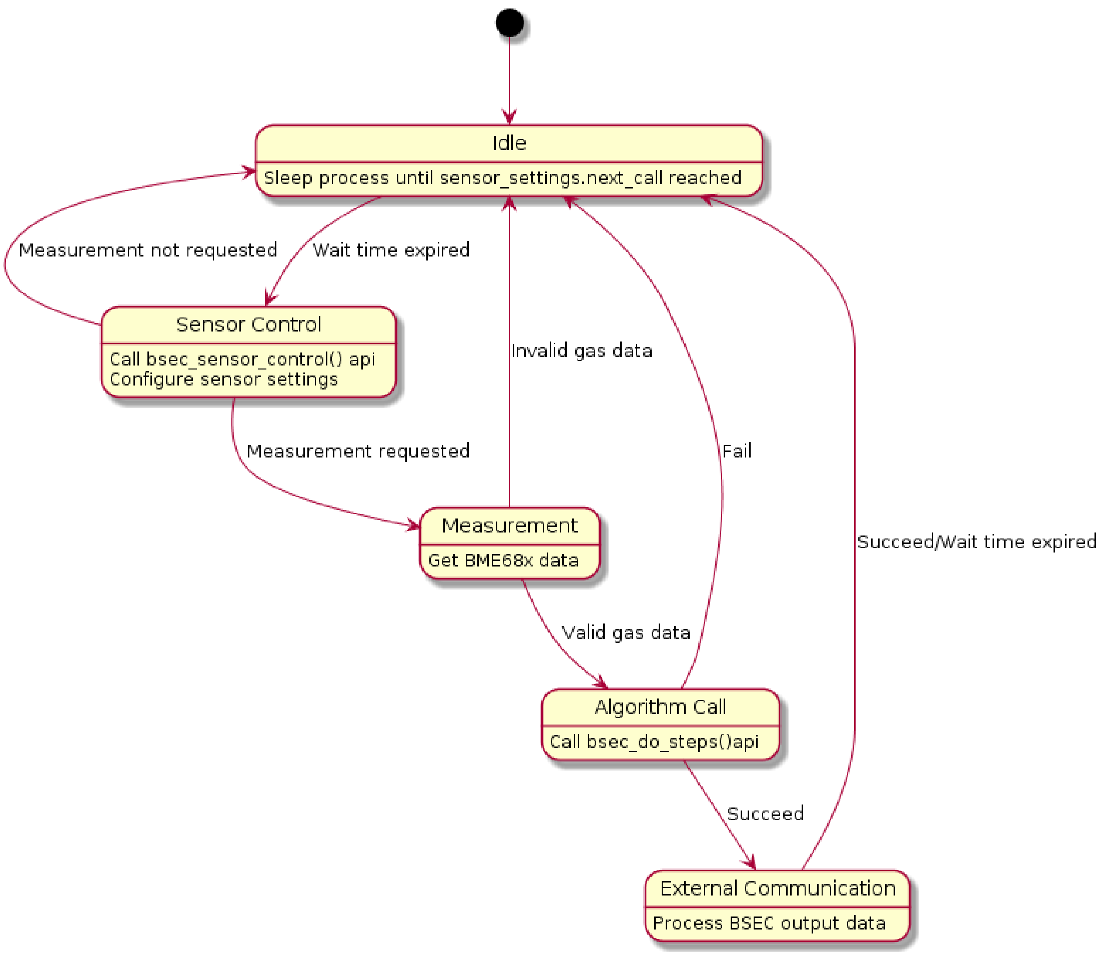

Figure 1.1: BSEC State Machine

Typical durations for the "Sleep until measurement is finished" are 0.190 seconds for LP mode and 2 seconds for

ULP mode. Typical durations for the "Sleep until next time bsec_sensor_control() asked to be called" are 2.8 seconds for LP mode and 298 seconds for ULP mode. Sleep duration of selectivity feature is as configured in config string.Sensor control provides interface for ON and OFF duration.

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, such as the BM↩ E280/680/688.

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	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 BME680/688 sensors together with the BSEC software package to estimate selectivity of certain target gases.

2.1 Prerequisites

First of all, you will need a BME688 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 in order to derive indoor-air-quality in the end. Of course, you will also need a development environment for the MCU of your choice. In this example, we will use the Arduino-based evaluation kit (BME688 shuttle & adapter (sold by BST)). Detailed instruction on how the environment shall be set up can be understood from (available on GitHub)

Once we are set with our hardware and development environment, we need two pieces of software to get the most out of our BME680/688 sensor:

1. BME68x 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 BME680/688 ourselves.
2. BSEC library - Instruction for downloading the library as a zip and importing it into the Arduino IDE is available under (Install the BSEC Library)

While the above might sound somewhat intimidating, we are lucky that BSEC comes with a 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 BME68x API, we need to add all included .h and .c files. In case of BSEC and Datalogger, we need to add the following (including BSEC interface headers as well the example that we want to extend):

- `examples\basic_config_state`
  - `basic_config_state.ino`
- `BSEC_Software_Library`
  - `inc`
    - `bsec_interface.h`
    - `bsec_datatypes.h`
  - `BSEC driver support`
    - `bsec.cpp`
    - `bsec.h`

  - `bsec_defs.h`
- `BSEC configuration`

Please note that example code here was tried on ESP8266 MCU. As BSEC is made available as a pre-compiled binary, we should also get the correct static library (.a) file. To use example code 'basic_config_state.ino' in an Arduino sketch, we should copy all the above mentioned folders. We can alter this reference code to adapt to the specific use case of interest.

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

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. Error codes with respect to selectivity is not illustrated here.

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 (-65 to 125)
  - HUMIDITY (0 to 100)
  - PRESSURE (0 to 2000000)
  - 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_E_DOSTEPS_TSINTRADIFFOUTOFRANGE

Possible reasons:

- General description: BSEC_E_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 a 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

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 in 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_PHYSIC↩ AL_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 arguement 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.

3.3.4.2 BSEC_W_SC_MODINSUFFICIENTWAITTIME

possible reasons:

- General description: BSEC_W_SC_MODINSUFFICIENTWAITTIME
- Insufficient wait time between two MODs

Correction methods:

- Make sure that there is sufficient wait time between two MODs

3.3.4.3 BSEC_W_SC_MODEXCEEDULPTIMELIMIT

Possible reasons:

- General description: BSEC_W_SC_MODEXCEEDULPTIMELIMIT
- Insufficient time between MOD and the last regular ULP measurement.
- Insufficient time between MOD and the next regular ULP measurement.

Correction methods:

- Make sure to trigger MOD within the defined time range between two regular ULP measurements.

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 BME68x API and example codes
Read results from sensor	See BME68x 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 initialized 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 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	*inputs*	Array of input data samples. Each array element represents a sample of a different physical sensor.
in	*n_inputs*	Number of passed input data structs.
out	*outputs*	Array of output data samples. Each array element represents a sample of a different virtual sensor.

Parameters

in,out	*n_outputs*	[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[5];
uint8_t n_input = 5;
bsec_output_t output[4];
uint8_t  n_output= 4;

bsec_library_return_t status;

// Populate the input structs, assuming the we have timestamp (ts),
// gas sensor resistance (R), temperature (T), pressure(P) ,humidity (rH) and profilePart(pP) available as
       input
// profilePart(pP) indicates the state of the profile (0-9)
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_PRESSURE;
input[2].signal = P;
input[2].time_stamp= ts;
input[3].sensor_id = BSEC_INPUT_HUMIDITY;
input[3].signal = rH;
input[3].time_stamp= ts;
input[4].sensor_id = BSEC_INPUT_PROFILE_PART;
input[4].signal = pP;
input[4].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)
        {
         // we have library support for 4 target classes
         // Maximum probability value out of Gas_Estimate_1/2/3/4 denotes the predicted target class here
            case BSEC_OUTPUT_GAS_ESTIMATE_1:
                // Retrieve the Gas_Estimate result of target class1 from output[i].signal
                // and do something with the data
                break;
            case BSEC_OUTPUT_GAS_ESTIMATE_2:
                // Retrieve the Gas_Estimate result of target class2 from output[i].signal
                // and do something with the data
                break;
            case BSEC_OUTPUT_GAS_ESTIMATE_3:
                // Retrieve the Gas_Estimate result of target class3 from output[i].signal
                // and do something with the data
                break;
         case BSEC_OUTPUT_GAS_ESTIMATE_4:
                // Retrieve the Gas_Estimate result of target class4 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	*config_id*	Identifier for a specific set of configuration settings to be returned; shall be zero to retrieve all configuration settings.
out	*serialized_settings*	Buffer to hold the serialized config blob
in	*n_serialized_settings_max*	Maximum available size for the serialized settings
in,out	*work_buffer*	Work buffer used to parse the binary blob
in	*n_work_buffer*	Length of the work buffer available for parsing the blob
out	*n_serialized_settings*	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_STATE_BLOB_SIZE for allotting the required size.

Parameters

in	*state_set_id*	Identifier for a specific set of states to be returned; shall be zero to retrieve all states.
out	*serialized_state*	Buffer to hold the serialized config blob
in	*n_serialized_state_max*	Maximum available size for the serialized states
in,out	*work_buffer*	Work buffer used to parse the blob
in	*n_work_buffer*	Length of the work buffer available for parsing the blob
out	*n_serialized_state*	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_STATE_BLOB_SIZE];
uint32_t  n_work_buffer_size = BSEC_MAX_STATE_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	*bsec_version_p*	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);
```

4.1.2.5 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 IAQ output in case of BSEC IAQ Solution set, reset means zeroing the output to the current ambient conditions.

Parameters

in	*sensor_id*	Virtual sensor to be reset

Returns

Zero when successful, otherwise an error code

```
// Example //
bsec_reset_output(BSEC_OUTPUT_IAQ);
```

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 or parallel mode measurement is triggered if requested by bsec_sensor_control().
- Once this sensor 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	*time_stamp*	Current timestamp in [ns]
out	*sensor_settings*	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	*serialized_settings*	Settings serialized to a binary blob
in	*n_serialized_settings*	Size of the settings blob
in,out	*work_buffer*	Work buffer used to parse the blob
in	*n_work_buffer_size*	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	*serialized_state*	States serialized to a binary blob
in	*n_serialized_state*	Size of the state blob
in,out	*work_buffer*	Work buffer used to parse the blob
in	*n_work_buffer_size*	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 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	*requested_virtual_sensors*	Pointer to array of requested virtual sensor (output) configurations for the library
in	*n_requested_virtual_sensors*	Number of virtual sensor structs pointed by requested_virtual_sensors
out	*required_sensor_settings*	Pointer to array of required physical sensor configurations for the library
in,out	*n_required_sensor_settings*	[in] Size of allocated required_sensor_settings 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 //

// Subscribe for 4 virtual sensors
bsec_sensor_configuration_t requested_virtual_sensors[4];
uint8_t n_requested_virtual_sensors = 4;

requested_virtual_sensors[0].sensor_id = BSEC_OUTPUT_GAS_ESTIMATE_1;
```

```
requested_virtual_sensors[0].sample_rate =
      BSEC_SAMPLE_RATE_HIGH_PERFORMANCE;
requested_virtual_sensors[1].sensor_id = BSEC_OUTPUT_GAS_ESTIMATE_2;
requested_virtual_sensors[1].sample_rate =
      BSEC_SAMPLE_RATE_HIGH_PERFORMANCE;
requested_virtual_sensors[2].sensor_id = BSEC_OUTPUT_GAS_ESTIMATE_3;
requested_virtual_sensors[2].sample_rate =
      BSEC_SAMPLE_RATE_HIGH_PERFORMANCE;
requested_virtual_sensors[3].sensor_id = BSEC_OUTPUT_GAS_ESTIMATE_4;
requested_virtual_sensors[3].sample_rate =
      BSEC_SAMPLE_RATE_HIGH_PERFORMANCE;

// 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 bsec_do_steps() 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 bsec_do_steps() is not in the valid range
BSEC_E_DOSTEPS_TSINTRADIFFOUTOFRANGE	Duplicate timestamps passed to bsec_do_steps()
BSEC_E_DOSTEPS_DUPLICATEINPUT	Duplicate input (physical) sensor ids passed as input to bsec_do_steps()
BSEC_I_DOSTEPS_NOOUTPUTSRETURNABLE	No memory allocated to hold return values from bsec_do_steps(), i.e., n_outputs == 0
BSEC_W_DOSTEPS_EXCESSOUTPUTS	Not enough memory allocated to hold return values from bsec_do_steps(), i.e., n_outputs < maximum number of requested output (virtual) sensors
BSEC_W_DOSTEPS_GASINDEXMISS	Gas Profile Indices Miss indication
BSEC_E_SU_WRONGDATARATE	The sample_rate of the requested output (virtual) sensor passed to bsec_update_subscription() is zero
BSEC_E_SU_SAMPLERATELIMITS	The sample_rate of the requested output (virtual) sensor passed to bsec_update_subscription() does not match with the sampling rate allowed for that sensor
BSEC_E_SU_DUPLICATEGATE	Duplicate output (virtual) sensor ids requested through bsec_update_subscription()

Enumerator

BSEC_E_SU_INVALIDSAMPLERATE	The sample_rate of the requested output (virtual) sensor passed to bsec_update_subscription() 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 bsec_update_subscription(), i.e., n_required_sensor_settings < BSEC_MAX_PHYSICAL_SENSOR
BSEC_E_SU_SAMPLINTVLINTEGERMULT	The sample_rate of the requested output (virtual) sensor passed to bsec_update_subscription() is not correct
BSEC_E_SU_MULTGASSAMPLINTVL	The sample_rate of the requested output (virtual), which requires the gas sensor, is not equal to the sample_rate 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 bsec_update_subscription() is not in the valid range; e.g., n_requested_virtual_sensors > 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 bsec_update_subscription()
BSEC_E_PARSE_SECTIONEXCEEDSWORKBU↩ FFER	n_work_buffer_size passed to bsec_set_[configuration/state]() not sufficient
BSEC_E_CONFIG_FAIL	Configuration failed
BSEC_E_CONFIG_VERSIONMISMATCH	Version encoded in serialized_[settings/state] passed to bsec_set_[configuration/state]() does not match with current version
BSEC_E_CONFIG_FEATUREMISMATCH	Enabled features encoded in serialized_[settings/state] passed to bsec_set_[configuration/state]() does not match with current library implementation
BSEC_E_CONFIG_CRCMISMATCH	serialized_[settings/state] passed to bsec_set_[configuration/state]() is corrupted
BSEC_E_CONFIG_EMPTY	n_serialized_[settings/state] passed to bsec_set_[configuration/state]() is to 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 bsec_set_[configuration/state]() does not match with the actual string size n_serialized_[settings/state] passed to these functions

Enumerator

BSEC_E_CONFIG_INSUFFICIENTBUFFER	String buffer insufficient to hold serialized data from BSEC library
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

4.1.3.2 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 [%]. 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	Temperature sensor output of BMExxx [degrees Celsius]. Note The BME688 is factory trimmed, thus the temperature sensor of the BME688 is very accurate. The temperature value is a very local measurement value and can be influenced by external heat sources. See also bsec_virtual_sensor_t
BSEC_INPUT_GASRESISTOR	Gas sensor resistance output of BMExxx [Ohm]. The resistance 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 subtracted from BSEC_INPUT_TEMPERATURE to compute BSEC_OUTPUT_SENSOR_HEAT_COMPENSATED_TEMPERATURE. ALL solution: Generic heat source 1 See also bsec_virtual_sensor_t
BSEC_INPUT_DISABLE_BASELINE_TRACKER	Additional input for device heat compensation 8. Generic heat source 8 Additional input that disables baseline tracker 0 - Normal 1 - Event 1 2 - Event 2
BSEC_INPUT_PROFILE_PART	Additional input that provides information about the state of the profile (0-9)

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	Indoor-air-quality estimate [0-500]. Indoor-air-quality (IAQ) gives an indication of the relative change in ambient TVOCs detected by BME688 sensor. Note 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.
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	Temperature sensor signal [degrees Celsius]. Temperature directly measured by BME688 in degree Celsius. Note This value is cross-influenced by the sensor heating and device specific heating.
BSEC_OUTPUT_RAW_PRESSURE	Pressure sensor signal [Pa]. Pressure directly measured by the BME688 in Pa.
BSEC_OUTPUT_RAW_HUMIDITY	Relative humidity sensor signal [%]. Relative humidity directly measured by the BME688 in %. Note This value is cross-influenced by the sensor heating and device specific heating.
BSEC_OUTPUT_RAW_GAS	Gas sensor signal [Ohm]. Gas resistance measured directly by the BME688 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

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	Sensor heat compensated temperature [degrees Celsius]. Temperature measured by BME688 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. Note 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. Thus, the value is calculated as follows: ▶ 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)` 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 BMExxx sensor supply voltage is 3.3V, the IoT_LP_3_3V.config shall be used.

Enumerator

BSEC_OUTPUT_SENSOR_HEAT_COMPENSATED_HUMIDITY	Sensor heat compensated humidity [%]. Relative humidity measured by BME688 which is compensated for the influence of sensor (heater) in %. It converts the BSEC_INPUT_HUMIDITY from temperature BSEC_INPUT_TEMPERATURE to temperature BSEC_OUTPUT_SENSOR_HEAT_COMPENSATED_TEMPERATURE. Note 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.
BSEC_OUTPUT_COMPENSATED_GAS	Gas Resistance after undergoing temp and humidity compensation
BSEC_OUTPUT_GAS_PERCENTAGE	percentage of min and max filtered gas value [%]
BSEC_OUTPUT_GAS_ESTIMATE_1	Gas_Estimate [probability 0.00-1]. Outputs corresponding to different target groups(Classes) with the probability of occurrence It takes inputs as gas resitance BSEC_INPUT_GASRESISTOR, temperature BSEC_INPUT_TEMPERATURE, pressure BSEC_INPUT_PRESSURE, humidity BSEC_INPUT_HUMIDITY and profile part BSEC_INPUT_PROFILE_PART. Note Selectivity solution is when we have these Gas_Estimate outputs subscribed toGas estimate output corresponding to Class1
BSEC_OUTPUT_GAS_ESTIMATE_2	Gas estimate output corresponding to Class2
BSEC_OUTPUT_GAS_ESTIMATE_3	Gas estimate output corresponding to Class3
BSEC_OUTPUT_GAS_ESTIMATE_4	Gas estimate output corresponding to Class4
BSEC_OUTPUT_RAW_GAS_INDEX	Gas index cyclically running from 0 to heater_profile_length-1

4.1.4 Defines

4.1.4.1 BSEC_MAX_PHYSICAL_SENSOR

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

Number of physical sensors that need allocated space before calling bsec_update_subscription()

4.1.4.2 BSEC_MAX_PROPERTY_BLOB_SIZE

```
#define BSEC_MAX_PROPERTY_BLOB_SIZE (2277)
```

Maximum size (in bytes) of the data blobs returned by bsec_get_configuration()

4.1.4.3 BSEC_MAX_STATE_BLOB_SIZE

```
#define BSEC_MAX_STATE_BLOB_SIZE (197)
```

Maximum size (in bytes) of the data blobs returned by bsec_get_state()

4.1.4.4 BSEC_MAX_WORKBUFFER_SIZE

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

Maximum size (in bytes) of the work buffer

4.1.4.5 BSEC_NUMBER_OUTPUTS

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

Number of outputs, depending on solution

4.1.4.6 BSEC_OUTPUT_INCLUDED

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

bitfield that indicates which outputs are included in the solution

4.1.4.7 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

4.1.4.8 BSEC_PROCESS_HUMIDITY

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

process_data bitfield constant for humidity

See also

bsec_bme_settings_t

4.1.4.9 BSEC_PROCESS_PRESSURE

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

process_data bitfield constant for pressure

See also

bsec_bme_settings_t

4.1.4.10 BSEC_PROCESS_TEMPERATURE

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

process_data bitfield constant for temperature

See also

bsec_bme_settings_t

4.1.4.11 BSEC_SAMPLE_RATE_DISABLED

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

Sample rate of a disabled sensor

4.1.4.12 BSEC_SAMPLE_RATE_HIGH_PERFORMANCE

```
#define BSEC_SAMPLE_RATE_HIGH_PERFORMANCE (0.055556f)
```

Sample rate in case of high performance

4.1.4.13 BSEC_SAMPLE_RATE_LP

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

Sample rate in case of Low Power Mode

4.1.4.14 BSEC_SAMPLE_RATE_ULP

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

Sample rate in case of Ultra Low Power Mode

4.1.4.15 BSEC_SAMPLE_RATE_ULP_MEASUREMENT_ON_DEMAND

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

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

4.1.4.16 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
  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 sensor 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 BMExxx sensor.

This structure contains settings that must be used to configure the BMExxx to perform a parallel 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_profile and bsec_bme_settings_t::heater_duration_profile 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_*

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 sensor 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 results 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 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:

- Gas Estimate accuracy would be either 0 or 3.

Virtual sensor	Value	Accuracy description
Gas Estimate	0	Gas Estimate output is not yet ready/burn-in phase
	3	On a successful scan cycle as per the heater profile cycle duration

- 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	Calibration on-going
	3	Calibration is done, now IAQ estimate achieves best performance

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
requested_virtual_sensors	bsec_virtual_sensor_t
required_sensor_settings	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.