AN5707

Application note

ST Bluetooth Mesh Sensor model

Introduction

The ubiquitous use of sensors in modern buildings augments day-by-day experience. The considerable variety of available sensors is classified on the basis of sensor application and usage context.

Sensor Server Model in mesh model Bluetooth specification addresses the challenge of smoothly integrating different sensors, and their associated behaviours, into a mesh network in an interoperable manner with client-server architecture.

Figure 1. Bluetooth Mesh multi-sensor node supported by Sensor Server Model

In Sensor Model, the sensor is described as a collection of standard Mesh device properties. A property is identified by an assigned property ID (a 2-byte value) and its state is called property value. Property ID describes the property data meaning and format.

Table 1. Representation of Bluetooth Mesh sensor node as a collection of mesh device properties

Mesh device properties	Property ID	Property values
Mesh device property 1	Property ID 1	
Mesh device property 2	Property ID 2	
...	...	
Mesh device property n	Property ID n	

ST Bluetooth Mesh Sensor Model example SDK gives flexibility to sensor definitions, initializes sensor contexts, enables all the required functions of sensor models, and provides ability to configure several parameters.

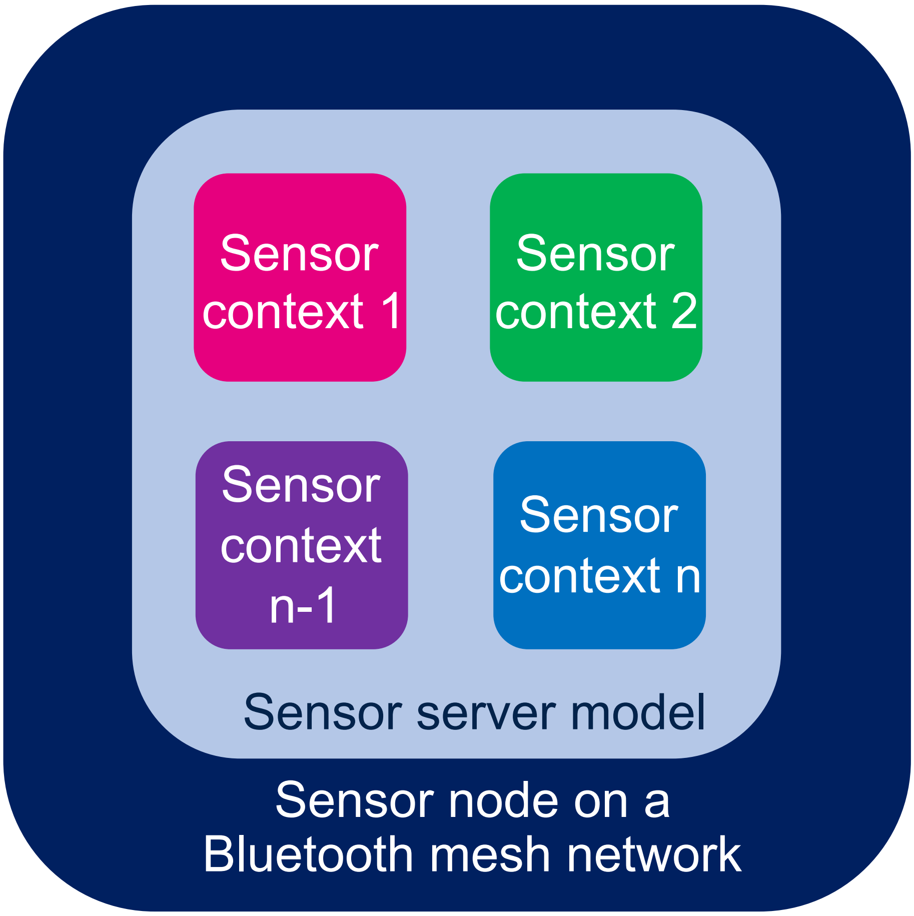
ST Sensor Model implementation, features, configuration, APIs and other peculiar functions are herein described.

The term "sensor" is used to reference both the sensor context exposed by the Sensor Server model and the physical sensor on the node.

1 Physical sensor vs sensor context

Sensor Server Model (as shown in the picture below) exposes different sensors to the sensor client. These software sensors are defined as sensor context which represents the sensor nature and behaviour understood by the sensor client with the help of the Sensor Descriptor (for further details, see Section 3.1).

Figure 2. Bluetooth Mesh sensor node software model highlighting Sensor Server Model and related sensor contexts to support physical sensors



A physical sensor can be mapped to multiple sensor contexts and vice-versa as described in the following examples.

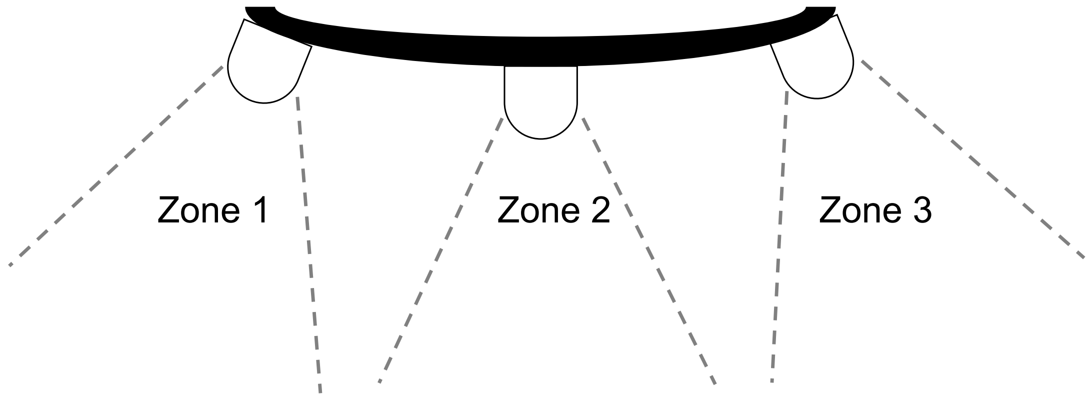
- Example 1: a Bluetooth Mesh sensor node should report both present ambient temperature and average ambient temperature in a determined moment of the day. This requires two sensor contexts to be exposed by the Sensor Server Model and both can be implemented using one physical temperature sensor:
  - the first sensor context exposes **Present Ambient Temperature** property which reports present ambient temperature.
  - the second sensor context exposes **Average Ambient Temperature In A Period of Day** property. The node keeps logs of the last 24-hour temperature readings and reports the average value.
- Example 2: a sensor node should report present ambient relative humidity. This requires one sensor context represented by **Present Ambient Relative Humidity** property but two physical sensors: humidity sensor and temperature sensor.

  $$relative\ humidity = \frac{actual\ \ vapor\ pressure}{saturated\ vapor\ pressure}$$

  The actual vapor pressure is provided by the humidity sensor while saturated vapor pressure is the temperature-dependent constant. Readings from both physical sensors are used to calculate the relative humidity reported by this sensor context.

- Example 3: a sensor node should report motion sensing information in its proximity. This requires only one sensor context, **Motion Sensed** property, to be exposed by the Sensor Server Model. Motion sensing measurement can be implemented using a combination of multiple motion detection sensors: PIR, microphone, ambient light sensor, etc. The measured value of motion sensing is used as the property value of this property.
- Example 4: a sensor node can have m sensor contexts and n physical sensors where m and n are independent from each other.
- Example 5: a sensor node can have multiple sensors supported on a single element (as shown in the figure below) or the same sensor on multiple elements. For example, a sensor node reports zone-wise motion detection according to 3 motion detection sensors covering different zones. Then, the node should have 3 motion sensor contexts (each of them with the same Property ID) on different elements.

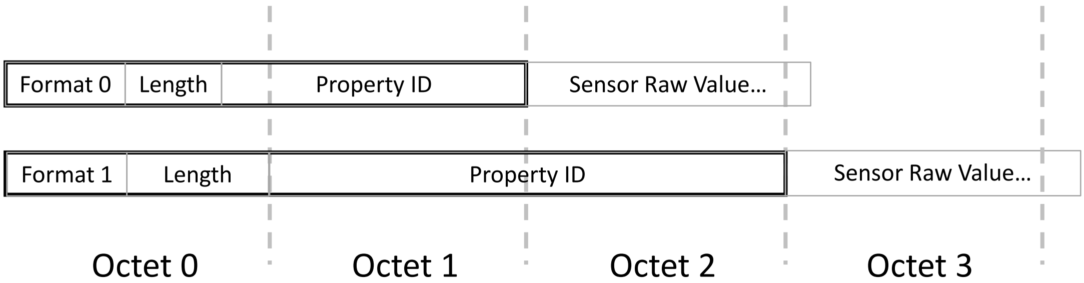
Figure 3. 3 motion detection sensors supported on 3 different elements



2 Sensor Server Model features

- Modular sensor contexts not directly dependent on physical sensors which enable the overall architecture modularity. The same set of sensor contexts running on entirely different products with physical sensors of different capability are treated similarly by the client. The client always access or understands a particular server node on the basis of the mesh device properties database independently from the node physical configuration, enhancing interoperability and compatibility.
- Periodic publishing of sensors status (that is sending sensor readings) to update potential client nodes about the present status. The publication can be related to a normal publishing period or a reduced publishing period known as fast publication (see Section 3.3). The latter is supported in two scenarios:
  – actual sensor property value: if values are in a particular range, the publishing period is reduced (fast cadence). For example, if a temperature sensor reading is too high, indicating a fire incident, it can send its reading at a rate faster than usual;
  – rate of change of property value: if the rate of change of sensor property value is more than a particular value, the publishing period is reduced.
- Support for sensors with single data point and with a property value represented as a column of a series of data points, such as a histogram (see Section 3.5).
- Run time configuration of sensor parameters: many parameters can be configured by the client to provide post-installation flexibility and sensor behavior optimization as well as on-the-fly usage.
- Marshalled sensor data to optimize message payload: sensor payloads of different sensors are efficiently packed. If Property ID value is less than 2048, it is represented by 11 bits (instead of 16 bits) and the corresponding property value length is represented by 4 bits (instead of 7 bits). This optimization saves one byte per sensor.

Figure 4. Marshalled Property ID with sensor raw values for different scenarios



Note: The Sensor Server Model does not support a Property ID value more than 2-byte long and a data length greater than 127.

3 Sensor state

Each sensor has a state which is a composite state consisting of four states:

- Sensor Descriptor
- Sensor Setting
- Sensor Cadence
- Sensor Data State or Sensor Series Column

There can be multiple instances of sensor state on the same element. For example, Sensor Model on a single element can support temperature sensor, pressure sensor and humidity sensor. These are called multi-sensors.

For further details on sensor state, refer to Mesh Model Specification v1.0.1

3.1 Sensor Descriptor state

Sensor descriptor constitutes attributes describing the sensor data. Sensor descriptor is constant and does not change throughout the lifetime of an element. The table below describes various fields of Sensor Descriptor state.

Table 2. **Sensor Descriptor state**

Field	Description
Property ID	The Sensor Property ID field is a 2-octet value referencing a device property that describes the meaning and the format of data reported by a sensor. The values for the field are: • 0x0000 – Prohibited • 0x0001- 0xFFFF – Identifier of a device property
Positive Sensor Tolerance	The Sensor Positive Tolerance field is a 12-bit value representing the magnitude of a possible positive error associated with the measurements that the sensor is reporting. For cases in which the tolerance information is not available,a special number is assigned indicating the value is Unspecified. The values for this state are: • 0x000 – Unspecified • 0x001 – 0xFFF – The positive tolerance of the sensor The magnitude of a possible positive error associated with the reported data (expressed as a percentage) is calculated by the following formula: $possible\,positive\ error[\,\%\,] = 100\left[\,\%\,\right] \cdot \frac{positive\ tolerance}{4095}$
Negative Tolerance	The Sensor Negative Tolerance field is a 12-bit value representing the magnitude of a possible negative error associated with the measurements that the sensor is reporting. When the tolerance information is not available, a special number is assigned indicating the value is Unspecified. The values for the state are: • 0x000 – Unspecified • 0x001 – 0xFFF – The negative tolerance of the sensor The magnitude of a possible negative error associated with the reported data (expressed as a percentage) is calculated by the following formula: $possible\,positive\ error[\,\%\,] = 100\left[\,\%\,\right] \cdot \frac{negative\ \ tolerance}{4095}$
Sampling Function	This Sensor Sampling Function field specifies the averaging operation or type of sampling function applied to the measured value. For example, this field can identify whether the measurement is an arithmetic mean value or an instantaneous value.

Field	Description
	For cases in which the sampling function is not available, a special number has been assigned to indicate the value is Unspecified. The values for this state are: • 0x00 – Unspecified • 0x01 – Instantaneous • 0x02 – Arithmetic Mean • 0x03 – RMS • 0x04 – Maximum • 0x05 – Minimum • 0x06 – Accumulated (cumulative moving range) • 0x07 – Count
Measurement Period	This Sensor Measurement Period field specifies a uint8 value n that represents the averaging time span, accumulation time, or measurement period in seconds over which the measurement is taken, using the formula: $represented\ value = 1.1^{n-64}$ For example, it can specify the length of the period used to obtain an average reading. For the cases where a value for the measurement period is not available or is not applicable, a special number has been assigned to indicate Not Applicable (0x00).
Update Interval	The measurement reported by a sensor is internally refreshed at the frequency indicated in the sensor Update Interval field (a temperature value that is internally updated every 15 minutes). This field specifies a uint8 value n that determines the interval (in seconds) among updates, using the formula: $represented\ value = 1.1^{n-64}$ For the cases where a value for the update interval is not available or is not applicable, a special number has been assigned to indicate Not Applicable (0x00).

3.2 Sensor Setting state

The Sensor Setting state controls parameter of a sensor. Single sensor can have multiple settings. For example, occupancy sensor may have a “sensitivity” setting that controls the sensitivity of the sensor. Sensitivity may be adjusted to prevent small animals from triggering the sensor. The table below describes various fields of Sensor Setting state.

Table 3. Sensor Setting state

Field	Description
Property ID	The Sensor Property ID field identifies the device property of a sensor. It matches the Sensor Property ID field of the Sensor Descriptor state.
Setting Property ID	The Sensor Setting Property ID field identifies the device property of a setting, including the size, format, and representation of the Sensor Setting Raw field. The values for this field are: • 0x0000 – Prohibited • 0x0001 – 0xFFFF – Identifier of a device property
Setting Access	The Sensor Setting Access field is an enumeration indicating whether the device property can be read or written. The values for the field are: • 0x0000 – Prohibited • 0x01 – The device property can be read • 0x02 – Prohibited • 0x03 – The device property can be read or written • 0x04 – 0xFF – Prohibited
Setting Raw	The Sensor Setting Raw field has a size and representation defined by the Sensor Setting Property ID and represents a setting of a sensor.

3.3 Sensor Cadence state

The Sensor Cadence state controls the cadence of sensor reports. It allows a sensor to be configured to send measured values using Sensor Status messages at a different cadence for a range of measured values. It also allows a sensor to be configured to send measured values when the value changes up or down by more than a configured delta value.

If the Fast Cadence High value is equal or higher than the Fast Cadence Low value, and the measured value is within the closed interval of [Fast Cadence Low, Fast Cadence High], the Sensor Status messages are published more frequently. The messages shall be published every Publish Period (configured for the model) divided by the Fast Cadence Period Divisor state.

If the Fast Cadence High value is lower than the Fast Cadence Low value, and the measured value is lower than the Fast Cadence High value or is higher than the Fast Cadence Low value, the Sensor Status messages are published more frequently. The messages shall be published every Publish Period (configured for the model) divided by the Fast Cadence Period Divisor state.

The table below describes various fields of Sensor Cadence state.

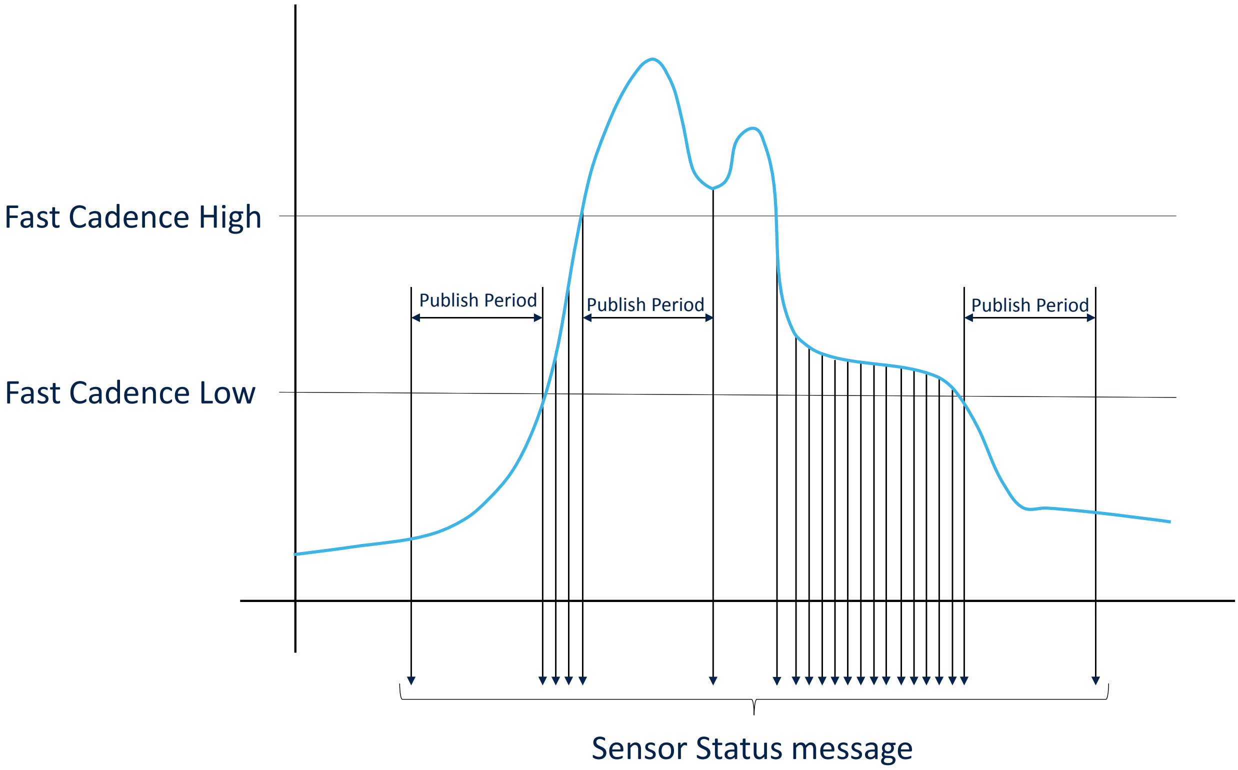
Table 4. Sensor Cadence state

Field	Description
Property ID	The Sensor Property ID field identifies the device property of a sensor. It matches the Sensor Property ID field of the Sensor Descriptor state.
Fast Cadence Period Divisor	The Fast Cadence Period Divisor field is a 7-bit value that controls the increased cadence of publishing Sensor Status messages. The value is represented as a divisor of the Publishing Period. For example, the value 0x04 would have a divisor of 16, and the value 0x00 would have a divisor of 1 (that is, the Publishing Period would not change). The valid range for the Fast Cadence Period Divisor state is 0–15; other values are Prohibited.
Status Trigger Type	The Status Trigger Type field defines the unit and format of the Status Trigger Delta Down and the Status Trigger Delta Up fields. • 0b0 means that the format is defined by the Format Type of the characteristic of the Sensor Property ID state references • 0b1 means that the unit is "unitless", the format type is 0x06 (uint16), and the value is represented as a percentage change with a resolution of 0.01 percent
Status Trigger Delta Down	The Status Trigger Delta Down field controls the negative change of a measured quantity that triggers publication of a Sensor Status message. The setting is calculated according to the value of the Status Trigger Type field: • if the value of the Status Trigger Type field is 0b0, the setting is calculated as defined by the Sensor Property ID state • if the value of the Status Trigger Type field is 0b1, the setting is calculated using the following formula: $represented\ \ value = Status\,TriggerDeltaDown/100$
Status Trigger Delta Up	The Status Trigger Delta Up field controls the positive change of a measured quantity that triggers publication of a Sensor Status message. The setting is calculated according to the value of the Status Trigger Type field: • if the value of the Status Trigger Type field is 0b0, the setting is calculated as defined by the Sensor Property ID state • if the value of the Status Trigger Type field is 0b1, the setting is calculated using the following formula: $represented\ \ value = Status\,TriggerDelta\,Up/100$
Status Min Interval	The Status Min Interval field is a 1-octet value that controls the minimum interval between publishing two consecutive Sensor Status messages. The value is represented in milliseconds. For example, the value 0x0A would represent an interval of 1024 ms. The valid range for the Status Min Interval is 0–26; other values are Prohibited
Fast Cadence Low	The Fast Cadence Low field defines the lower boundary of a range of measured quantities when the publishing cadence is increased as defined by the Fast Cadence Period Divisor field. The represented value is calculated as defined by the Sensor Property ID state. The Fast Cadence Low may be set to a value higher than the Fast Cadence High. In such cases, the increased cadence will occur outside the range (Fast Cadence High, Fast Cadence Low).
Fast Cadence High	The Fast Cadence High field defines the upper boundary of a range of measured quantities when the publishing cadence is increased as defined by the Fast Cadence Period Divisor field. The represented value is calculated as defined by the Sensor Property ID state.

Field	Description
	The Fast Cadence High can be set to a value lower than the Fast Cadence Low. In such cases, the increased cadence will occur outside the range (Fast Cadence High, Fast Cadence Low).

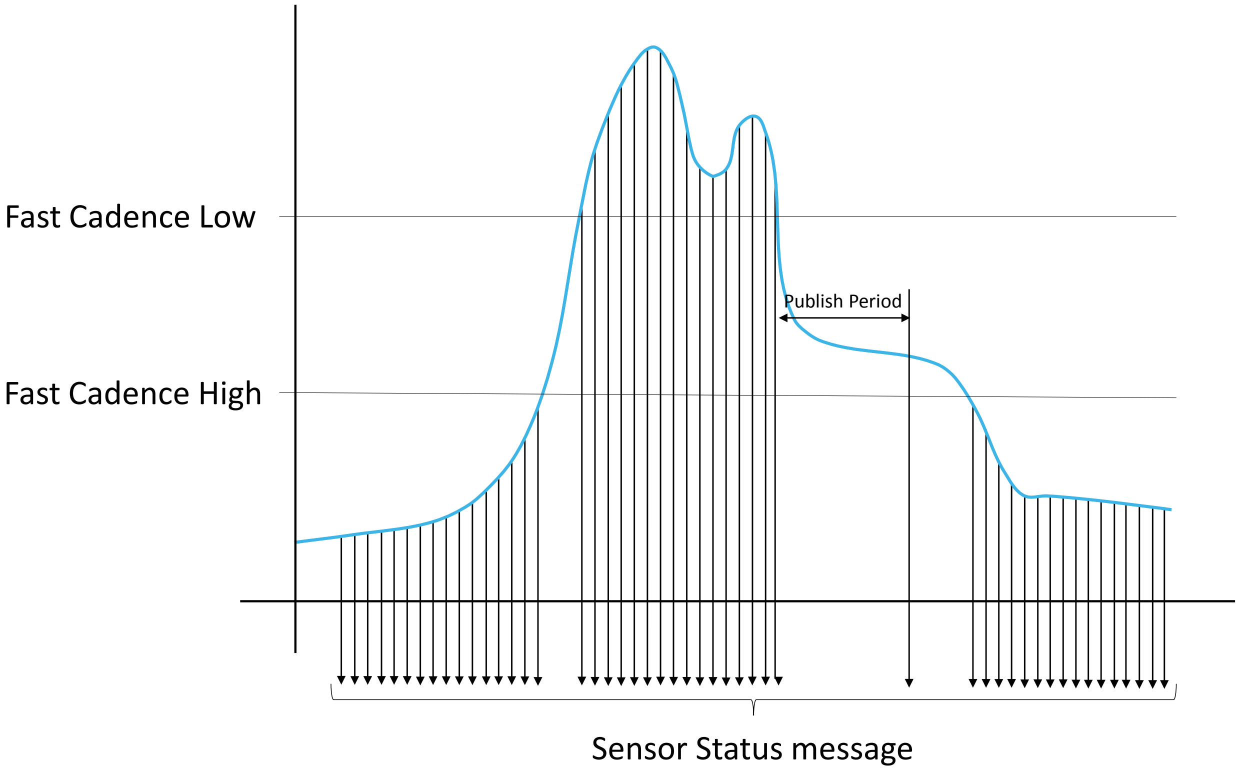
The figures below show how the cadence of sent messages varies on the basis of the measured quantity. If the measured value is within the range defined by the Fast Cadence High and the Fast Cadence Low values, messages are sent more frequently.

Figure 5. Publishing of Sensor Status messages at a fast cadence within a certain range of values



When measured values exceed the Fast Cadence High value or fall below the Fast Cadence Low value, messages are sent less frequently until the measured value is within the specified range again.

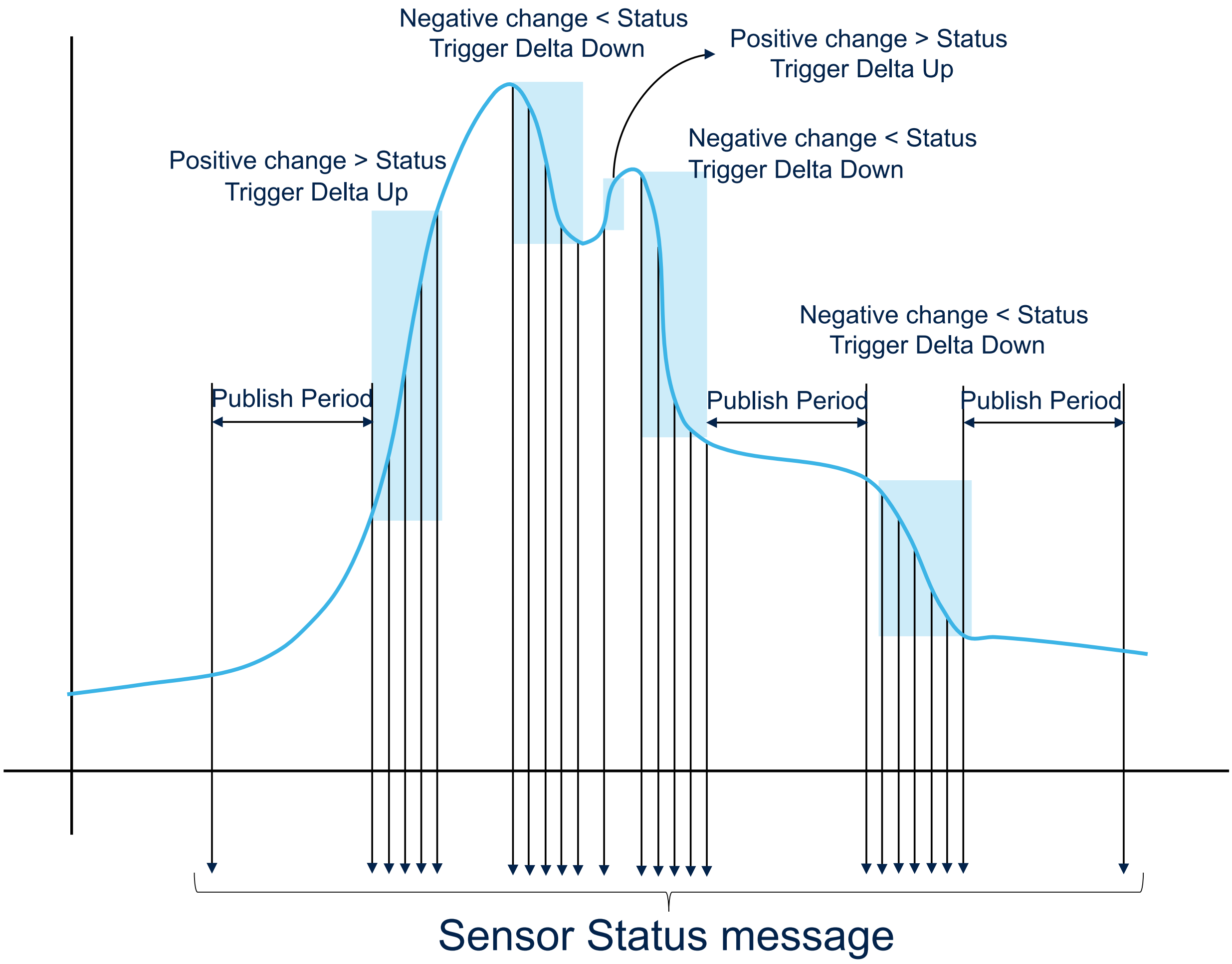
Figure 6. Publishing of Sensor Status messages at a slow cadence outside a certain range of values (Fast Cadence High < Fast Cadence Low)



If the change of the measured value is more rapid, the Sensor Status messages are published more frequently. A value represented by the Fast Cadence Period Divisor state is used as a divider for the Publish Period (configured for the model) if the change exceeds the conditions determined by the Status Trigger Type, Status Trigger Delta Down, and the Status Trigger Delta Up.

The figure below shows Sensor Status messages sending triggered by the measured quantity change exceeding the configured Status Trigger Delta Down or Status Trigger Delta Up value.

Figure 7. **Publishing of Sensor Status messages triggered by changes of measured quantity (absolute or in percentage as defined by status trigger type field)**



3.4 Sensor Data state

The table below describes the structure of Sensor Data state which is a sequence of pairs of Sensor Property ID and the corresponding Raw Value. Raw Value field size (in bytes) and its interpretation is defined by the characteristics referenced by the Sensor Property ID (for further information, refer to Mesh Device Properties v2.0). The combined size of the Sensor Data state cannot exceed the message payload size.

Multiple instances of Sensor states could be included in the same model, provided that each instance has a unique value of the Sensor Property ID. The pairs are organized in ascending order according to the value of the Property ID n field.

Table 5. **Sensor Data state structure**

Field	Description
Property ID 1	Property describing the data series of the sensor
Raw Value 1	Raw Value field with a size and representation defined by the 1st device property
Property ID 2	ID of the 2nd device property of a sensor
Raw Value 2	Raw Value field with a size and representation defined by the 2nd property
...	
Property ID n	ID of the nth device property of the sensor

Field	Description
Raw Value n	Raw Value field with a size and representation defined by the n^{th} device property

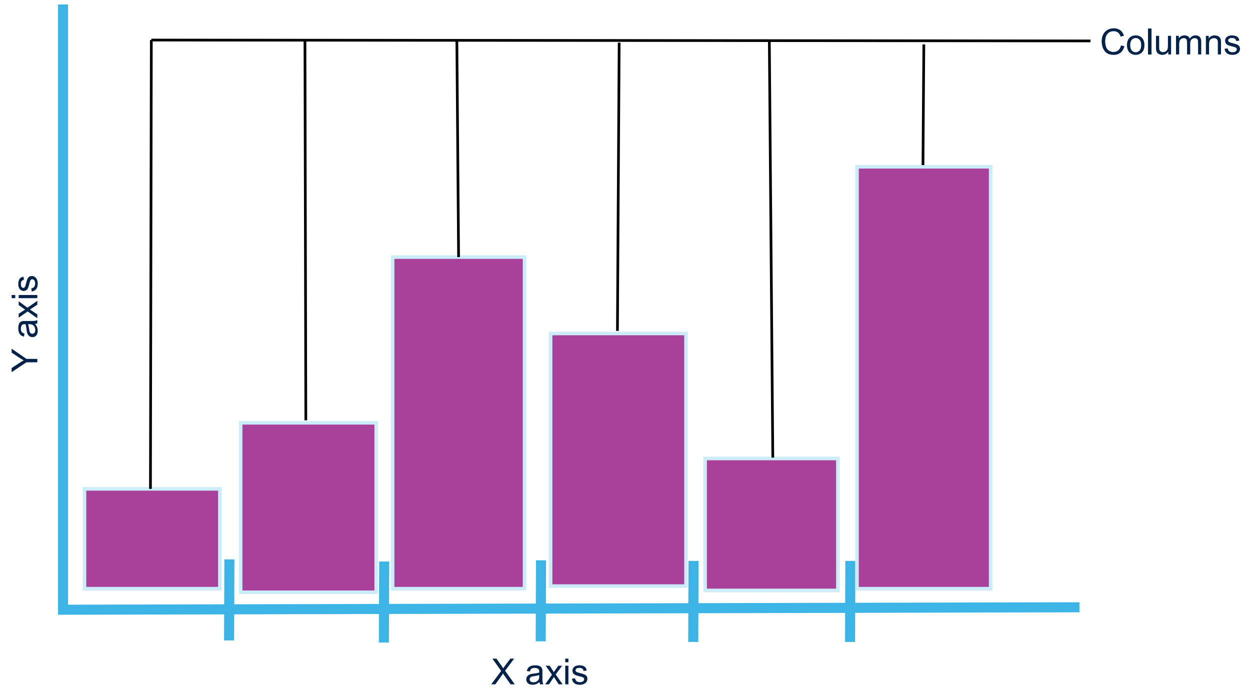
3.5 Sensor Series Column state

Values measured by sensors can be organized as arrays (and represented as series of columns or histograms). Each Sensor Series Column state represents a column of a series.

Table 6. Sensor Series Column State

Field	Description
Property ID	Property describing the data series of the sensor
Raw Value X [n]	The Sensor Raw Value X field has a size and representation defined by the Sensor Property ID and represents the left corner of the column on the X axis
Column Width [n]	The Sensor Column Width field has a size and representation defined by the Sensor Property ID and represents the width of the column on the X axis
Raw Value Y [n]	The Sensor Raw Value Y field has a size and representation defined by the Sensor Property ID and represents the height of the column on the Y axis

Figure 8. Sensor Series Column example



Note: Sensors supporting Sensor Series Column state do not support periodic publishing (periodic publication of Sensor Status) and sensor cadence (fast publication of Sensor Status).

4 Sensor configuration

While designing sensor nodes with Sensor Server Model, configuration is done in two phases:

1. During product design and implementation, parameters are set and remain fixed for all the lifetime of a node. For example, Sensor Descriptor state is constant and does not change. Similarly, read-only Sensor Setting state values cannot be modified. All the required physical sensor(s) should also be initialized accordingly with the help of appropriate sensor drivers.
2. During product installation, run-time parameters and states are configured by the client. These parameters can also be changed later anytime. For example, Sensor Settings values which have write access can be optimized after installation. Similarly, for sensors which support Cadence, fields of Sensor Cadence State can be modified.

4.1 Sensor init structures

`sensor_server_init_params_t`, `sensor_init_params_t`, `sensor_settings_init_params_t` and `sensor_series_column_init_params_t`, have to be configured and initialized properly.

4.1.1 sensor_server_init_params_t

```
typedef struct
{
uint8_t sensorsCount;
sensor_init_params_t sensorInitParams [TOTAL_SENSORS_COUNT ];
} sensor_server_init_params_t;
```

Table 7. sensor_server_init_params_t members

Parameter	Description
`sensorsCount`	Number of sensors exposed by Sensor Server model
`sensorInitParams`	Initialization parameters of each sensor. All the sensors can be identified by sensor offset which indicates the sensor position in this array

4.1.2 sensor_init_params_t

```
typedef struct
{
uint8_t elementIDx;
uint16_t propertyId;
uint16_t positiveTolerance;
uint16_t negativeTolerance;
uint8_t samplingFunction;
uint8_t measurementPeriod;
uint8_t updateInterval;
uint8_t dataLength;
uint8_t cadenceState;
uint32_t valuesRange;
uint8_t settingsCount;
sensor_series_column_init_params_t seriesColumn [SENSOR_MAX_SERIES_COUNT];
} sensor_init_params_t;
```

Table 8. sensor_init_params_t members

Parameter	Description
`elementIdx`	Element index on which the sensor is present
`propertyId`	16-bit Assigned value given in Mesh Device Properties which represents current sensor
`positiveTolerance`	The Sensor Positive Tolerance field is a 12-bit value representing the magnitude of a possible positive error associated with the measurements that the sensor is reporting (Section 3.1)
`negativeTolerance`	The Sensor Negative Tolerance field is a 12-bit value representing the magnitude of a possible negative error associated with the measurements that the sensor is reporting (Section 3.1)
`samplingFunction`	This Sensor Sampling Function field specifies the averaging operation or type of sampling function applied to the measured value (Section 3.1)
`measurementPeriod`	This Sensor Measurement Period field specifies a uint8 value n that represents the averaging time span, accumulation time, or measurement period in seconds over which the measurement is taken, using the formula (Section 3.1): $representedvalue = 1.1^{n-64}$
`updateInterval`	The measurement reported by a sensor is internally refreshed at the frequency indicated in the Sensor Update Interval field (that is, a temperature value that is internally updated every 15 minutes). This field specifies a uint8 value n that determines the interval (in seconds) among updates, using the formula (Section 3.1): $representedvalue = 1.1^{n-64}$
`dataLength`	Size of property value (in bytes) required to represent sensor property value (Section 3.4)
`cadenceState`	This field specifies if cadence state is supported by the sensor
`valuesRange`	This field represent range of values supported by sensor property value. This is used to calculate change of sensor property value in percentage which is used for status trigger (see Mesh Device Properties) when status trigger type is 0b1 (Section 3.3)
`settingsCount`	Number of Sensor Setting state (Section 3.2) applicable to each sensor. This value can be any whole number
`settings`	Settings parameter of each setting (refer to Section 3.2 and Section 4.1.3)
`seriesCount`	Number of Sensor Series Column state (Section 3.5) associated with the sensor. It can be any whole number
`seriesColumn`	RawX and column width parameter of each series column (Section 3.5 and Section 4.1.4)

4.1.3 sensor_settings_init_params_t

```
typedef struct
{
uint16_t propertyId;
uint8_t settingAccess;
uint32_t settingRaw;
} sensor_settings_init_params_t;
```

Table 9. sensor_settings_init_params_t members

Parameter	Description
`propertyId`	16-bit value to identify the type of setting (see Section 3.2)
`settingAccess`	Represents read access or both read and write access of sensor setting (see Section 3.2)
`settingRaw`	Raw value of setting property (see Section 3.2)

4.1.4 sensor_series_column_init_params_t

```
typedef struct
{
uint32_t rawX;
uint32_t columnWidth;
} sensor_series_column_init_params_t;
```

Table 10. sensor_series_column_init_params_t members

Parameter	Description
`rawX`	RawX value of column (see Section 3.5)
`columnWidth`	Column Width of Series Column state (see Section 3.5)

An example implementation (described in Section Appendix A Initializing sensor structures) to initialize these structures using nested macros is available in the SDK sensor_cfg_usr.h file.

4.2 Constraints on configuration and initialization

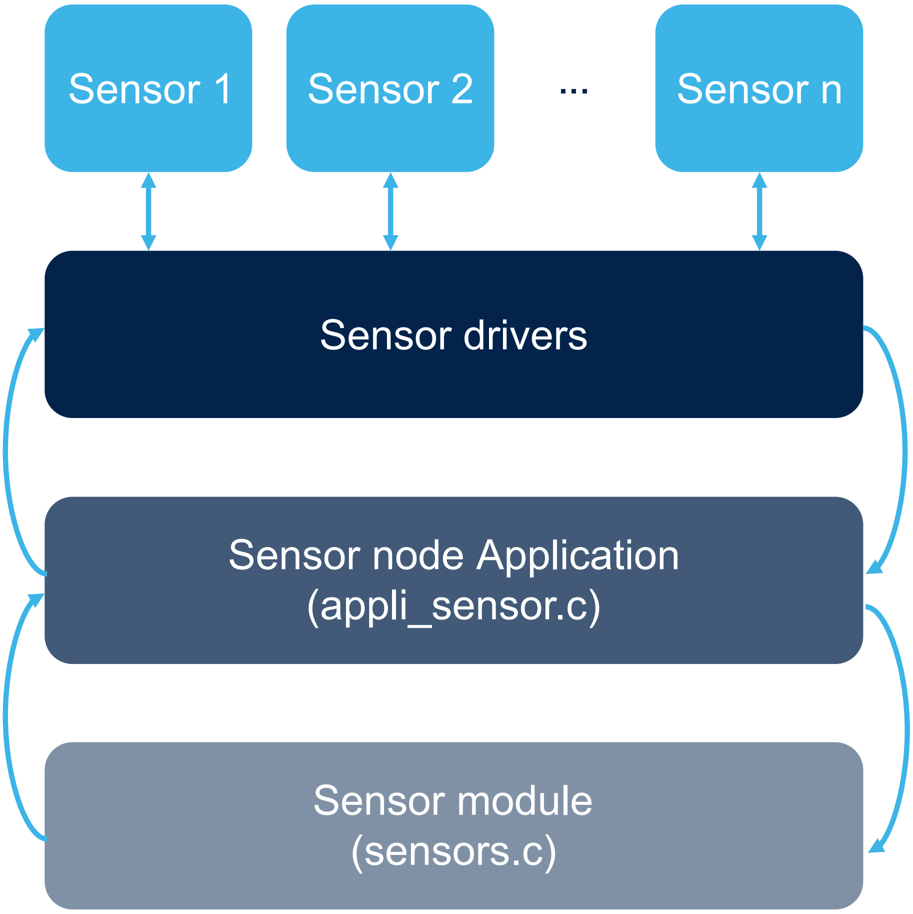
When initializing multiple sensors supported on multiple elements in a mesh enabled device, you need to follow a specific order, otherwise, the device is not able to successfully initialize the sensor module:

- Constraint 1: Sensor client model cannot be present on the same element with Light LC Server model
- Constraint 2: Sensor count has to be greater than or equal to 1
- Constraint 3: Element index value on which the corresponding sensor is supported should be lower than the total number of elements supported by the device (`APPLICATION_NUMBER_OF_ELEMENTS`). For example, for a device with 3 elements, the element index value should be lower than 3. Possible element index values (for all sensors) are 0, 1 and 2
- Constraint 4: Element index (for multi-sensors on multiple elements) defined for successive sensors has to be in ascending order
- Constraint 5: Sensors on a given element index should have different Property IDs and Property ID has to be different from zero. On element index n, each sensor Property ID should be unique. However, if element indexes are different, Property ID can be repeated
- Constraint 6: Property IDs of different sensors within an element has to be in ascending order. For example, consider that an element index n supports three sensors. In this case, Property IDs have to be arranged in ascending order in `sensor_server_init_params_t sensorInitParams`
- Constraint 7: Sensors with Sensor Series Column State cannot support cadence. Cadence is supported only by sensors supporting Sensor Data State
- Constraint 8: Allocated buffer size for sensor module has to be sufficient

4.3 Sensor application callbacks and APIs

User application and sensor module interacts with each other via APIs and callbacks as shown below.

Figure 9. Sensor node and physical sensors



4.3.1 Sensor module APIs

4.3.1.1 Sensor_Send

Table 11. Sensor_Send APIs

Function name	Prototype	Behavior description	Input parameter	Output parameter
`Sensor_Send`	`MOBLE_RESULT Sensor_Send (MOBLEUINT8 sensorOffset, MOBLEUINT8 elementIdx, MOBLEUINT16 propertyId)`	To publish status of a sensor identified by sensor offset parameter (see Section 4.4 for publication mechanisms)	`sensorOffset`: sensor offset in the sensor array `elementIdx`: element index on which the sensor is supported `propertyId`: sensor property ID	Fail if the sensor is not initialized

4.3.1.2 *Sensor_UpdateCadence*

Table 12. Sensor_UpdateCadence

Function name	Prototype	Behavior description	Input parameter	Output parameter
`Sensor_UpdateCadence`	`MOBLE_RESULT Sensor_UpdateCadence (MOBLEUINT8 sensorOffset, MOBLEUINT8 elementIdx, MOBLEUINT16 propertyId)`	Updates publishing period according to the cadence state. This function should be called after a change in the sensor value. For example, considering a temperature sensor node which supports Fast Cadence mechanism, whenever the temperature exceeds a particular threshold, it starts fast publication. Thus, a call to this function is required at every change in the sensor reading to keep updating the sensor module parameters. Call to this function results in additional callbacks from the sensor module (Section 4.4.4)	`sensorOffset`: sensor offset in the sensor array `elementIdx`: element index on which the sensor is supported `propertyId`: sensor property ID	Fail if sensors are not initialized or sensor offset does not exist Not implemented if cadence is not supported *False* if parameters are not consistent, otherwise *Success*

4.3.1.3 *Sensor_SleepDurationMs_Get*

Table 13. Sensor_SleepDurationMs_Get

Function name	Prototype	Behavior description	Input parameter	Output parameter
`Sensor_SleepDurationMs_Get`	`MOBLEUINT32 Sensor_SleepDurationMs_Get(void)`	Returns time after which call to process `Sensor_Process` is required to publish appropriate sensor status. During this time, the sensor module can remain inactive. This function can be used to implement the sensor node low power operation	None	Sleep time in milliseconds

4.3.1.4 *Sensor_Process*

Table 14. Sensor_Process

Function name	Prototype	Behavior description	Input parameter	Output parameter
`Sensor_Process`	`void Sensor_Process(void)`	Scheduler to send the sensor status according to the Model Publication State and Sensor Cadence State	None	None

Function name	Prototype	Behavior description	Input parameter	Output parameter
		The status messages of the sensors supported on the same element index, which have a status publication scheduled within 50 ms from each other, are combined in a single sensor status message. The single element publishes one sensor status message per time, even if the sensors are supported on multi-elements		

4.3.1.5 SensorServer_Init

Table 15. SensorServer_Init

Function name	Prototype	Behavior description	Input parameter	Output parameter
`SensorServer_Init`	`MOBLE_RESULT SensorServer_Init (void* sensorBuff, const sensor_server_cb_t* sensor_cb, MOBLEUINT16 sizeBuff, const void* initParams)`	Sensor server initialization	`sensorBuff`: buffer to be allocated to the sensor server model structure `sensor_cb`: reference to application callbacks used by sensor server `sizeBuff`: buffer size `initParams`: initialization parameters provided by the sensor application	*Success* if initialization successful, otherwise *Fail*

4.3.1.6 Sensor_ModelPublishSet

Table 16. Sensor_ModelPublishSet

Function name	Prototype	Behavior description	Input parameter	Output parameter
`Sensor_ModelPublishSet`	`MOBLE_RESULT Sensor_ModelPublishSet (model_publicationparams_t* pPublishParams)`	Updates the sensor publishing period of all the sensors on the element index identified by the incoming element index. Whenever the model publish period changes (by Config Client using Config Model Publish Set message), call to this function is required to	`pPublishParams`: publishes the model parameters	*Fail* if the sensor is not initialized, otherwise *Success*

Function name	Prototype	Behavior description	Input parameter	Output parameter
		update sensor module with the new publishing period		

4.3.2 Sensor module callbacks

Sensor callbacks are members of `sensor_server_cb_t`. Through these calls, the sensor module can request various parameters and data from the application.

Callbacks are call-to-function pointers where functions are implemented in the application area. In the final implementation, some of these functions can be skipped depending on their end-use.

Mapping of function pointers to functions is typically done at initialization (using `SensorServer_Init`).

4.3.2.1 Sensor_CadenceGet_cb

Table 17. **Sensor_CadenceGet_cb**

Function name	Prototype	Behavior description	Input parameter	Output parameter
`Sensor_CadenceGet_cb`	`void (*Sensor_CadenceGet_cb) (sensor_CadenceCbParam_t* pCadenceParam, MOBLEUINT32 length, MOBLE_ADDRESS peerAddr, MOBLE_ADDRESS dstPeer, MOBLEUINT8 elementIndex)`	Callback corresponding to valid Sensor Cadence Get message received from client. It is used to inform sensor application about processing of Sensor Cadence Get message by the sensor module	`pCadenceParam`: message parameters `length`: data length `peerAddr`: client address `dstPeer`: destination server address. It can be a unicast address or one of the subscribed address `elementIndex`: element index on which the message is received	None

4.3.2.2 Sensor_CadenceSet_cb

Table 18. **Sensor_CadenceSet_cb**

Function name	Prototype	Behavior description	Input parameter	Output parameter
`Sensor_CadenceSet_cb`	`void (*Sensor_CadenceSet_cb) (sensor_CadenceCbParam_t* pCadenceParam, MOBLEUINT32 length, MOBLE_ADDRESS peerAddr, MOBLE_ADDRESS dstPeer, MOBLEUINT8 elementIndex);`	Callback corresponding to valid Sensor Cadence Set message received from client. It is used to inform sensor application about processing of Sensor Cadence Set message by the sensor module	`pCadenceParam`: message parameters `length`: data length `peerAddr`: client address `dstPeer`: destination server address. It can be a unicast address or one of the subscribed address	None

Function name	Prototype	Behavior description	Input parameter	Output parameter
			`elementIndex`: element index on which the message is received	

4.3.2.3 *Sensor_CadenceSetUnack_cb*

Table 19. Sensor_CadenceSetUnack_cb

Function name	Prototype	Behavior description	Input parameter	Output parameter
`Sensor_CadenceSetUnack_cb`	`void(*Sensor_ CadenceSetUnack_cb) (sensor_CadenceCbParam_t* pCadenceParam, MOBLEUINT32 length, MOBLE_ADDRESS peerAddr, MOBLE_ADDRESS dstPeer, MOBLEUINT8 elementIndex)`	Callback corresponding to valid Sensor Cadence Set Unacknowledged message received from client. It is used to inform sensor application about processing of Sensor Cadence Set Unacnowledged message by the sensor module	`pCadenceParam`: message parameters `length`: data length `peerAddr`: client address `dstPeer`: destination server address. It can be a unicast address or one of the subscribed address `elementIndex`: element index on which the message is received	None

4.3.2.4 *Sensor_SettingsGet_cb*

Table 20. Sensor_SettingsGet_cb

Function name	Prototype	Behavior description	Input parameter	Output parameter
`Sensor_SettingsGet_cb`	`void (*Sensor_SettingsGet_cb) (sensor_SettingsCbParams_t* pSettingsParam, MOBLEUINT32 length, MOBLE_ADDRESS peerAddr, MOBLE_ADDRESS dstPeer, MOBLEUINT8 elementIndex)`	Callback corresponding to valid Sensor Settings Get message received from client. It is used to inform sensor application about processing of Sensor Settings Get message by the sensor module	`pSettingsParam`: message parameters `length`: data length `peerAddr`: client address `dstPeer`: destination server address. It can be a unicast address or one of the subscribed address `elementIndex`: element index on which the message is received	None

4.3.2.5 Sensor_SettingGet_cb

Table 21. Sensor_SettingGet_cb

Function name	Prototype	Behavior description	Input parameter	Output parameter
Sensor_SettingGet_cb	`void (*Sensor_SettingGet_cb) (sensor_SettingCbParams_t* pSettingParam, MOBLEUINT32 length, MOBLE_ADDRESS peerAddr, MOBLE_ADDRESS dstPeer, MOBLEUINT8 elementIndex)`	Callback corresponding to valid Sensor Setting Get message received from client. It is used to inform sensor application about processing of Sensor Setting Get message by the sensor module	`pSettingParam`: message parameters `length`: data length `peerAddr`: client address `dstPeer`: destination server address. It can be a unicast address or one of the subscribed address `elementIndex`: element index on which the message is received	None

4.3.2.6 Sensor_SettingSet_cb

Table 22. Sensor_SettingSet_cb

Function name	Prototype	Behavior description	Input parameter	Output parameter
Sensor_SettingSet_cb	`void (*Sensor_SettingSet_cb) (sensor_SettingCbParams_t* pSettingParam, MOBLEUINT32 length, MOBLE_ADDRESS peerAddr, MOBLE_ADDRESS dstPeer, MOBLEUINT8 elementIndex)`	Callback corresponding to valid Sensor Setting Set message received from client. It is used to inform sensor application about processing of Sensor Setting Set message by the sensor module	`pSettingParam`: message parameters `length`: data length `peerAddr`: client address `dstPeer`: destination server address. It can be a unicast address or one of the subscribed address `elementIndex`: element index on which the message is received	None

4.3.2.7 Sensor_SettingSetUnack_cb

Table 23. Sensor_SettingSetUnack_cb

Function name	Prototype	Behavior description	Input parameter	Output parameter
Sensor_SettingSetUnack_cb	`void(*Sensor_ SettingSetUnack_cb) (sensor_SettingCbParams_t* pSettingParam, MOBLEUINT32 length, MOBLE_ADDRESS peerAddr, MOBLE_ADDRESS dstPeer, MOBLEUINT8 elementIndex)`	Callback corresponding to valid Sensor Setting Set Unacknowledged message received from client. It is used to inform sensor application about processing of Sensor Setting Set message by the sensor module	`pSettingParam`: message parameters `length`: data length `peerAddr`: client address `dstPeer`: destination server address. It can be a unicast address or one of the subscribed address `elementIndex`: element index on which the message is received	None

4.3.2.8 Sensor_DescriptorGet_cb

Table 24. Sensor_DescriptorGet_cb

Function name	Prototype	Behavior description	Input parameter	Output parameter
Sensor_DescriptorGet_cb	`void (*Sensor_DescriptorGet_cb) (MOBLEUINT8 propID, MOBLEUINT32 length, MOBLE_ADDRESS peerAddr, MOBLE_ADDRESS dstPeer, MOBLEUINT8 elementIndex)`	Callback corresponding to valid Sensor Descriptor Get received from client. It is used to inform sensor application about processing of Sensor Descriptor Get message by the sensor module	`propID`: sensor property ID `length`: data length `peerAddr`: client address `dstPeer`: destination server address. It can be a unicast address or one of the subscribed address `elementIndex`: element index on which the message is received	None

4.3.2.9 Sensor_Get_cb

Table 25. Sensor_Get_cb

Function name	Prototype	Behavior description	Input parameter	Output parameter
Sensor_Get_cb	`void (*Sensor_Get_cb) (MOBLEUINT8 propID, MOBLEUINT32 length, MOBLE_ADDRESS peerAddr, MOBLE_ADDRESS dstPeer, MOBLEUINT8 elementIndex)`	Callback corresponding to valid Sensor Get message received from client. It is used to inform sensor application about processing of Sensor Get message by the sensor module	`propID`: sensor property ID `length`: data length `peerAddr`: client address `dstPeer`: destination server address. It can be a unicast address or one of the subscribed address `elementIndex`: element index on which the message is received	None

4.3.2.10 Sensor_ColumnGet_cb

Table 26. Sensor_ColumnGet_cb

Function name	Prototype	Behavior description	Input parameter	Output parameter
Sensor_ColumnGet_cb	`void (*Sensor_ColumnGet_cb) (sensor_ColumnCbParams_t* pColumnParam, MOBLEUINT32 length, MOBLE_ADDRESS peerAddr, MOBLE_ADDRESS dstPeer, MOBLEUINT8 elementIndex)`	Callback corresponding to valid Sensor Column Get message received from client. It is used to inform sensor application about processing of Sensor Column Get message by the sensor module	`pColumnParam`: message parameters `length`: data length `peerAddr`: client address `dstPeer`: destination server address. It can be a unicast address or one of the subscribed address `elementIndex`: element index on which the message is received	None

4.3.2.11 Sensor_SeriesGet_cb

Table 27. Sensor_SeriesGet_cb

Function name	Prototype	Behavior description	Input parameter	Output parameter
Sensor_SeriesGet_cb	`void (*Sensor_SeriesGet_cb) (sensor_SeriesCbParams_t* pSeriesParam, MOBLEUINT32 length, MOBLE_ADDRESS peerAddr, MOBLE_ADDRESS dstPeer, MOBLEUINT8 elementIndex)`	Callback corresponding to valid Sensor Series Get message received from client. It is used to inform sensor application about processing of Sensor Series Get message by	`pSeriesParam`: message parameters `length`: data length `peerAddr`: client address `dstPeer`: destination server address. It can be a unicast address or one of the subscribed address	None

Function name	Prototype	Behavior description	Input parameter	Output parameter
		the sensor module	`elementIndex`: element index on which the message is received	

4.3.2.12 Sensor_ReadDescriptor_cb

Table 28. Sensor_ReadDescriptor_cb

Function name	Prototype	Behavior description	Input parameter	Output parameter
`Sensor_ReadDescriptor_cb`	`MOBLE_RESULT (*Sensor_ReadDescriptor_cb) (MOBLEUINT8 sensorOffset, sensor_DescriptorCbParams_t* pDescriptorParams)`	Callback to update sensor_DescriptorCbParams_t parameters. The sensor module requests to read the sensor descriptor (identified by sensor offset). The following parameters have to be updated for the sensor: • Positive tolerance • Negative tolerance • Sampling function • Measurement Period • Update interval	`sensorOffset`: sensor offset in the sensor array `pDescriptorParams`: parameters set by the application	Result as set by the application

4.3.2.13 Sensor_ReadValue_cb

Table 29. Sensor_ReadValue_cb

Function name	Prototype	Behavior description	Input parameter	Output parameter
`Sensor_ReadValue_cb`	`MOBLE_RESULT (*Sensor_ReadValue_cb) (MOBLEUINT8 sensorOffset, sensor_ValueCbParams_t* pValueParams)`	Callback requests sensor property value. Sensor data should be formatted according to Property ID characteristic defined in Mesh Device Properties v2.0	`sensorOffset`: sensor offset in the sensor array `pValueParams`: parameters set by the application	Result as set by the application

4.3.2.14 Sensor_ReadColumn_cb

Table 30. Sensor_ReadColumn_cb

Function name	Prototype	Behavior description	Input parameter	Output parameter
`Sensor_ReadColumn_cb`	`MOBLE_RESULT (*Sensor_ReadColumn_cb) (MOBLEUINT8 sensorOffset, MOBLEUINT8 columnOffset, sensor_ColumnCbParams_t* pColumnParams)`	This is a mandatory callback if at least one sensor supporting Sensor Series Column State is present. It requests the sensor column data from the application (for details, refer to Section 3.5)	`sensorOffset`: sensor offset in the sensor array `columnOffset`: column offset in the sensor array `pColumnParams`: parameters set by the application	Result as set by the application

4.3.2.15 Sensor_ReadSeries_cb

Table 31. Sensor_ReadSeries_cb

Function name	Prototype	Behavior description	Input parameter	Output parameter
`Sensor_ReadSeries_cb`	`MOBLE_RESULT (*Sensor_ReadSeries_cb) (MOBLEUINT8 sensorOffset, sensor_SeriesCbParams_t* pSeriesParams)`	This is a mandatory callback if at least one sensor supporting Sensor Series Column State is present. It requests sensor series of all the columns (for details refer to Section 3.5)	`sensorOffset`: sensor offset in the sensor array `pSeriesParams`: parameters set by the application	Result as set by the application

4.3.2.16 Sensor_IsFastCadence_cb

Table 32. Sensor_IsFastCadence_cb

Function name	Prototype	Behavior description	Input parameter	Output parameter
`Sensor_IsFastCadence_cb`	`MOBLEUINT8 (*Sensor_IsFastCadence_cb) (MOBLEUINT8 sensorOffset, void* pFastCadenceLow, void* pFastCadenceHigh)`	This is a mandatory callback if at least one sensor supporting Sensor Data State and Sensor Cadence is present. This callback determines if fast	`sensorOffset`: sensor offset in the sensor array `pFastCadenceLow`: Fast Cadence Low value `pFastCadenceHigh`: Fast Cadence High value	Trigger status

Function name	Prototype	Behavior description	Input parameter	Output parameter
		cadence has to be used or not for a sensor.		

4.3.2.17 *Sensor_IsStatusTrigger_cb*

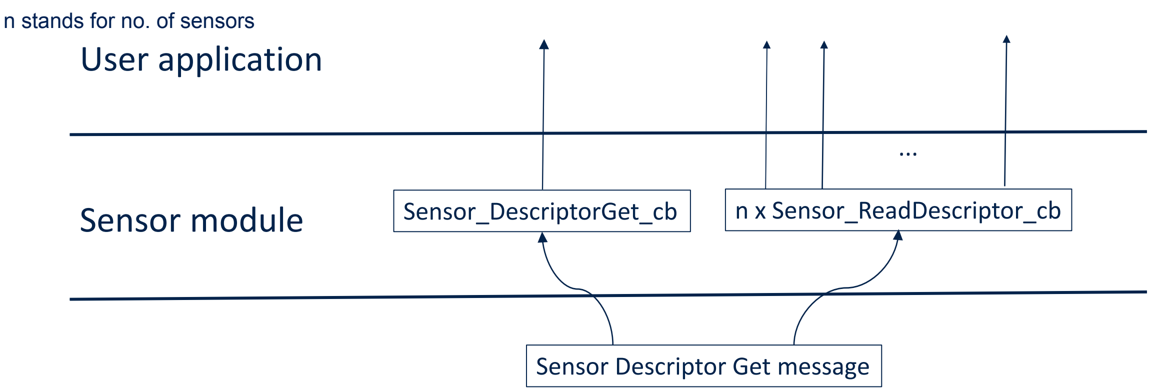
Table 33. **Sensor_IsStatusTrigger_cb**

Function name	Prototype	Behavior description	Input parameter	Output parameter
`Sensor_IsStatusTrigger_cb`	`MOBLEUINT8 (*Sensor_IsStatusTrigger_cb) (MOBLEUINT8 sensorOffset, status_trigger_type_e triggerType, void* pDeltaDown, void* pDeltaUp)`	This is a mandatory callback if at least one sensor supporting Sensor Data State along with cadence is present. This callback determines if status should be triggered or not for a sensor.	`sensorOffset`: sensor offset in the sensor array `triggerType`: trigger type status `pDeltaDown`: delta down value `pDeltaUp`: delta up value	Trigger status

4.4 Examples to demonstrate different callback relations with Sensor Server Model messages and APIs

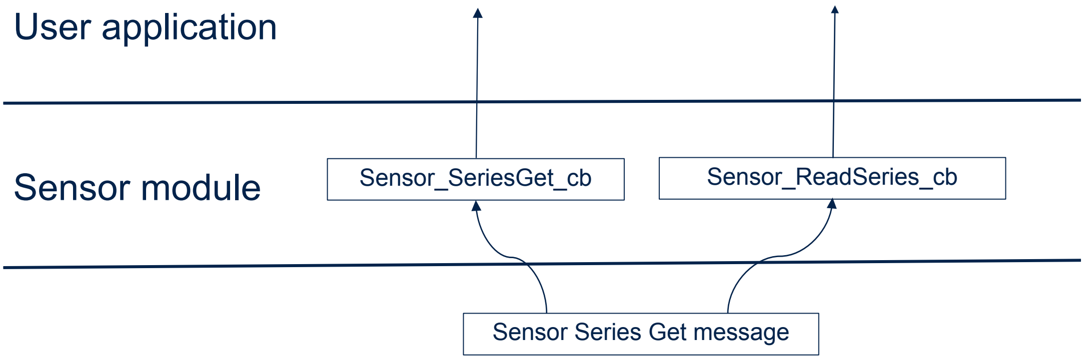
4.4.1 Callbacks corresponding to Sensor Descriptor Get message

Figure 10. **Application callbacks corresponding to Sensor Descriptor Get message**



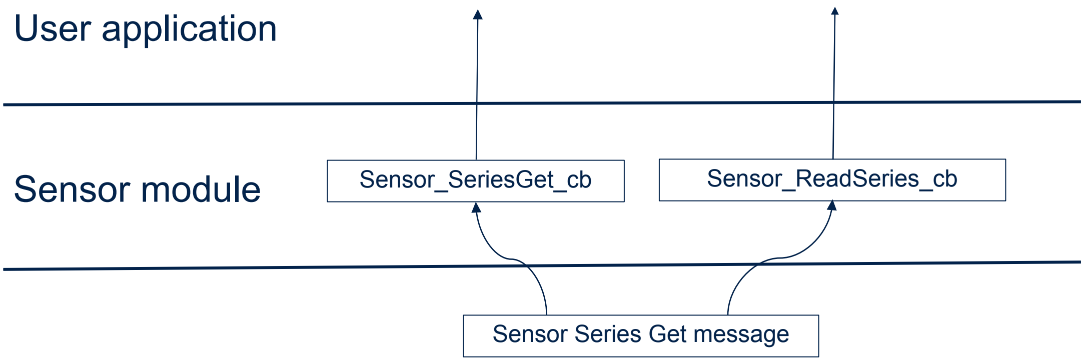
4.4.2 Callbacks corresponding to Sensor Series Get message

Figure 11. Application callbacks corresponding to Sensor Series Get message



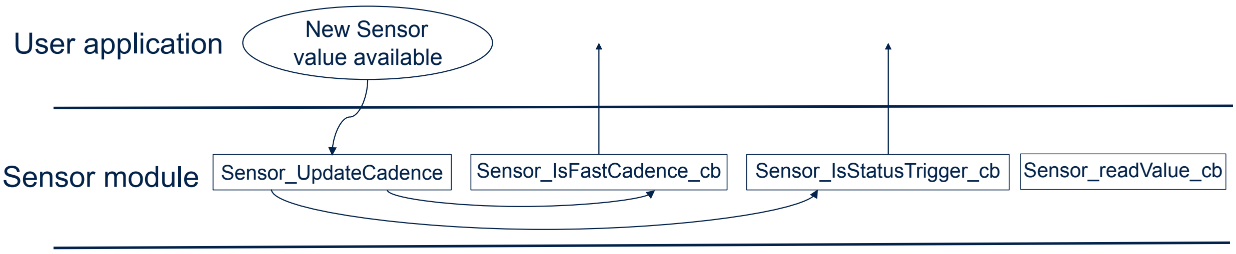
4.4.3 Callbacks corresponding to Sensor Cadence Set Unack message of Sensor Cadence Set message

Figure 12. Application callbacks corresponding to Sensor Set Unack or Sensor Set message



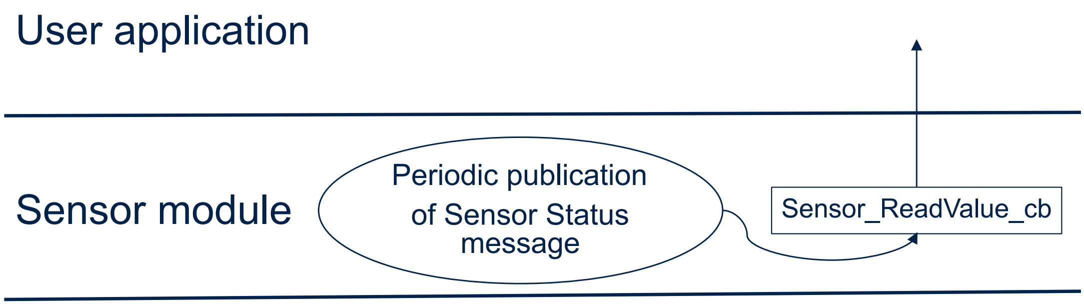
4.4.4 Callbacks corresponding to Sensor_UpdateCadence API

Figure 13. Application callbacks corresponding to Sensor_UpdateCadence API



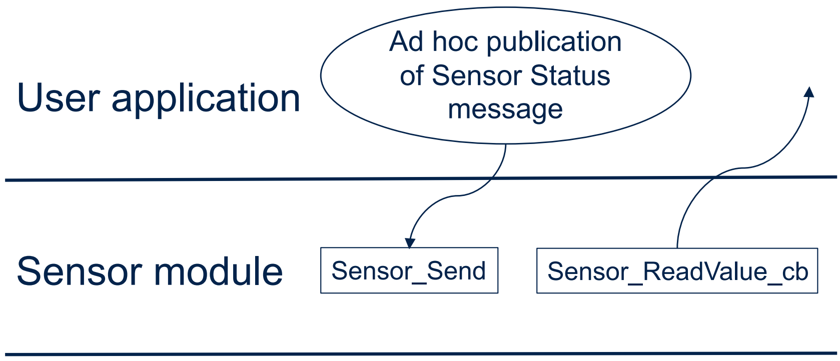
4.4.5 Callbacks corresponding to periodic publication of Sensor Status message

Figure 14. Application callbacks corresponding to periodic publication of Sensor Status message



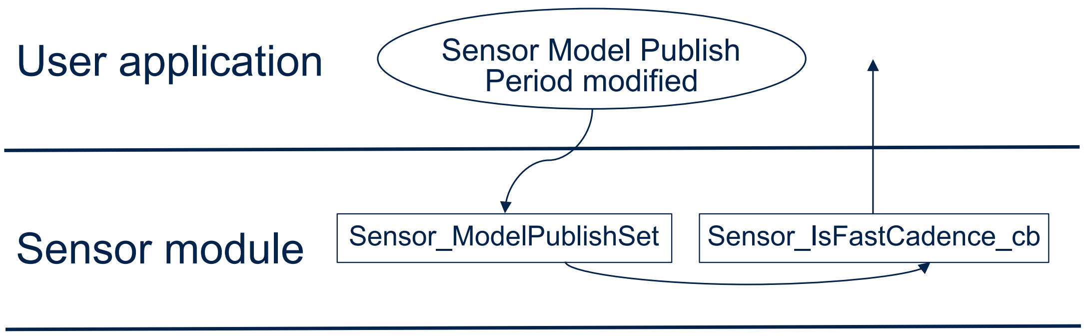
4.4.6 Callbacks corresponding to Sensor_Send API

Figure 15. Application callbacks corresponding to ad hoc publication of Sensor Status message



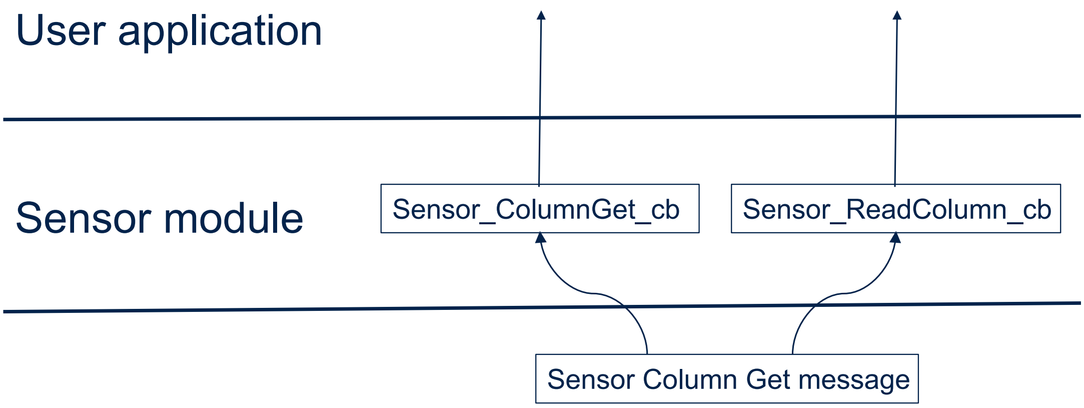
4.4.7 Callbacks corresponding to Sensor_ModelPublishSet API

Figure 16. Application callbacks corresponding to change in Sensor Model Publish period due to change in model publish state



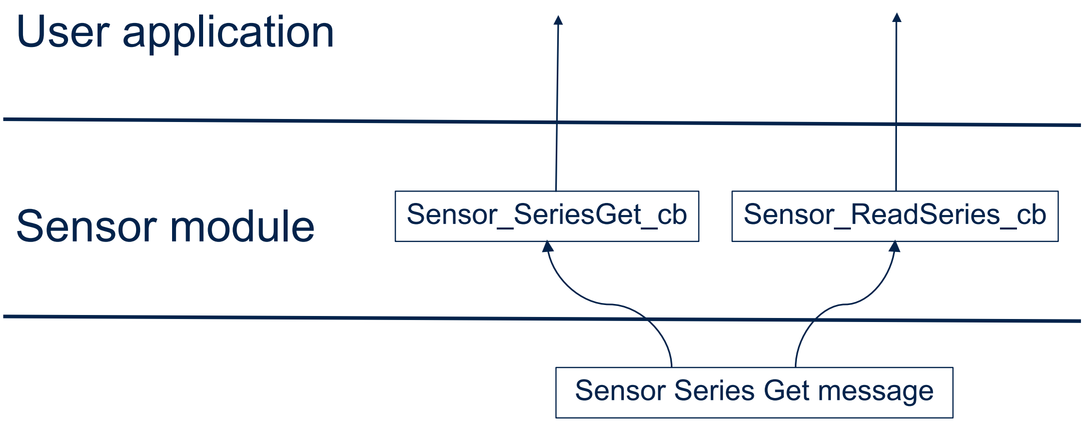
4.4.8 Callbacks corresponding to Sensor Column Get message

Figure 17. Application callbacks corresponding to Sensor Column Get message



4.4.9 Callbacks corresponding to Sensor Series Get message

Figure 18. Application callbacks corresponding to Sensor Series Get message



4.5 Sensor status publishing mechanisms

Sensor nodes publish Sensor Data State using Sensor Status message. There are four different scenarios in which Sensor Status message is published:

- in response to Sensor Get message - Sensor Get is an acknowledged message and Sensor Status is sent as an acknowledgement
- periodically with publish parameters defined by the corresponding Model Publication State - Publishing period in this case can be equal to Publish Period State or reduced (fast cadence) by a factor as defined by Sensor Cadence State
- immediately (or at fast rate) on Status Trigger Delta Up or Status Trigger Delta Down as defined by Sensor Cadence State
- using sensor module API Sensor_Send - The application, if required, can publish Sensor Status message itself
  - This is provided as an additional option available to the application to send sensor status message to subscribed nodes. In some cases (which are not covered in the specification) the application can publish additional status messages (not to be used too often to avoid network congestion).

4.6 Low power support

A sensor node can be a low power node. To achieve low power consumption, the sensor node has to remain active for a short duration and can go back to sleep for longer durations. The duration time for which the sensor node should remain in sleep mode is computed by the sensor module API (`Sensor_SleepDurationMs_Get`).

In addition to other low power constraints, a sensor module constraint can also be added while going to sleep to ensure proper operation of the low power sensor node.

4.7 Sensor module memory requirements

The volatile memory requirements of Sensor Server model depend on sensor count, sensor setting count and sensor series column count parameters as listed below.

Table 34. RAM footprints for different scenarios

Parameter	Case 1	Case 2	Case 3	Case 4
TOTAL_SENSOR_COUNT	2	3	5	5
TOTAL_SENSOR_SETTINGS_COUNT	2	10	10	10
TOTAL_SENSOR_SERIES_COLUMN_COUNT	0	0	0	20
RAM	160	280	392	472

Moreover, the sensor application can also consume additional memory resources.

5 Apps

Android and iOS ST BLE Mesh apps are available on Google Play and Apple Store, respectively.

Figure 19. **Android and iOS stacks**

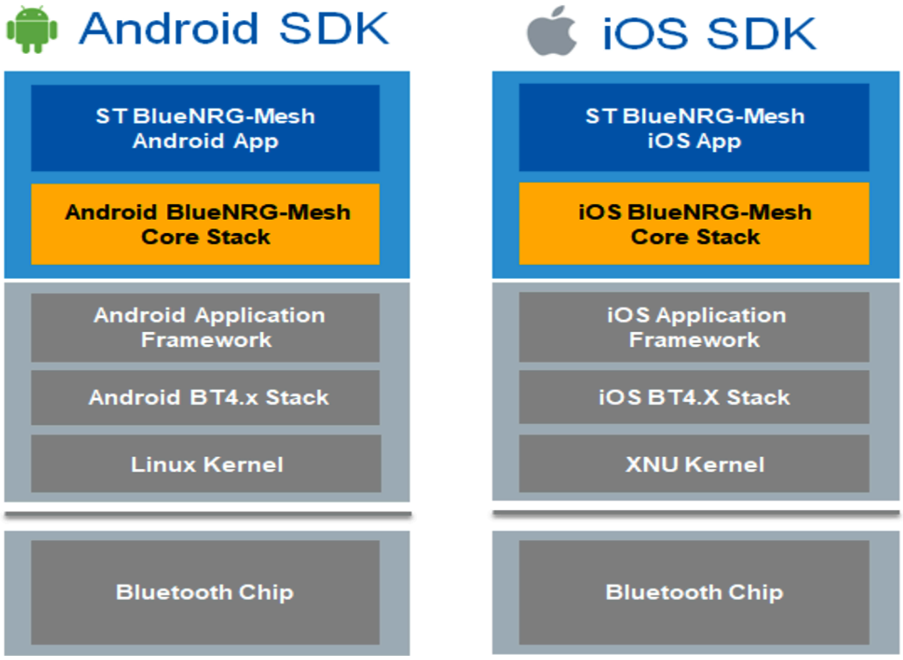


Figure 20. **Android and iOS QR codes**

To use the Sensor Client Model in the app(s), follow the procedure described by the figures below.

Figure 21. Sensor Client Model app screenshot (1 of 4)

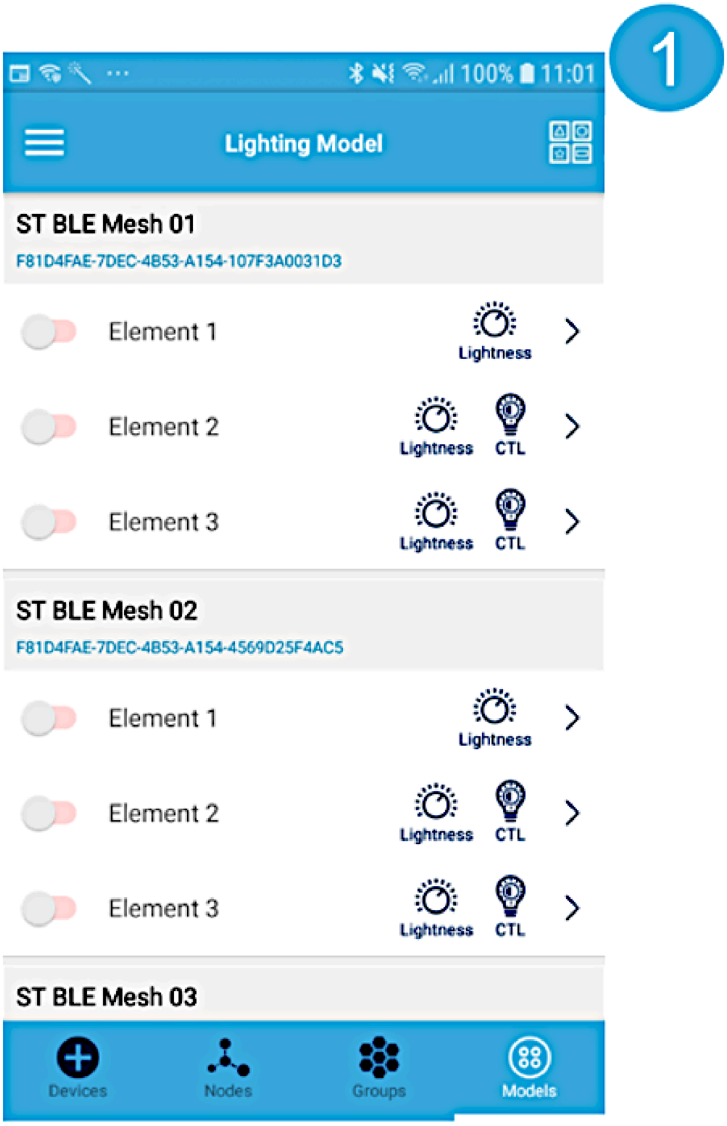


Figure 22. Sensor Client Model app screenshot (2 of 4)

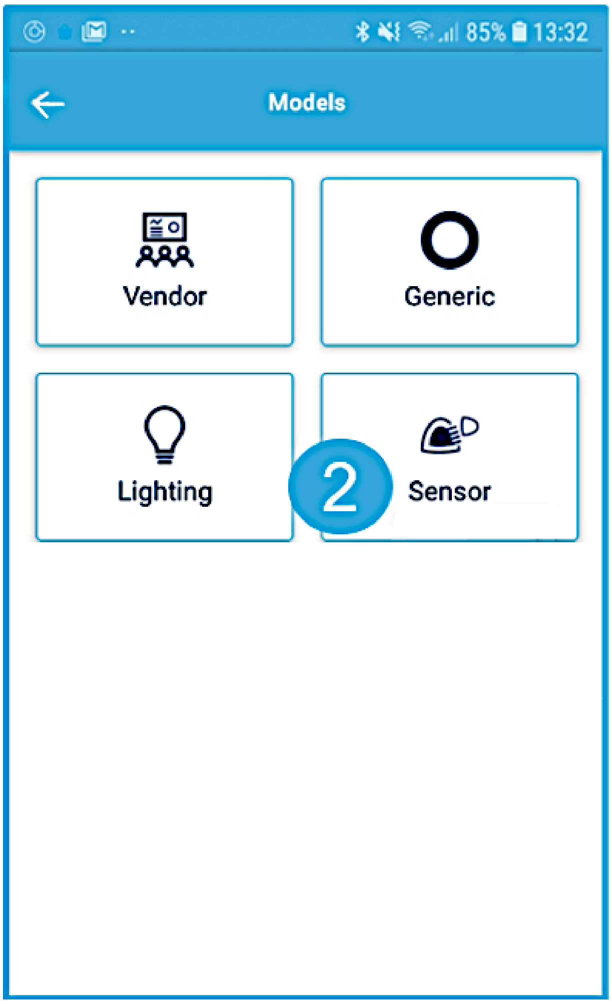


Figure 23. Sensor Client Model app screenshot (3 of 4)

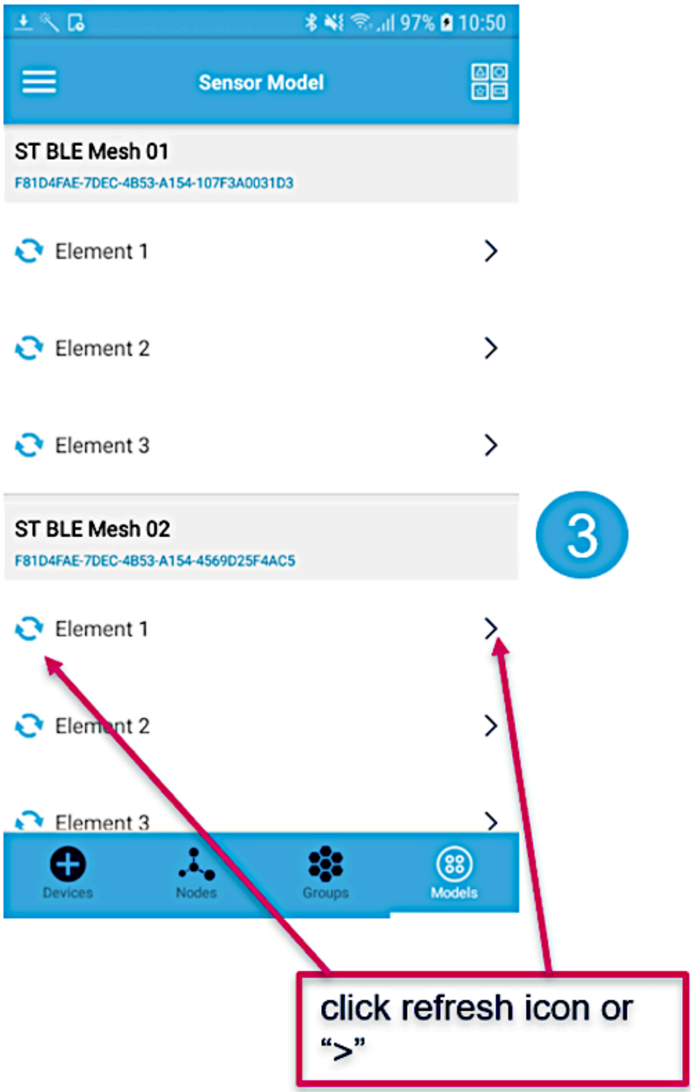
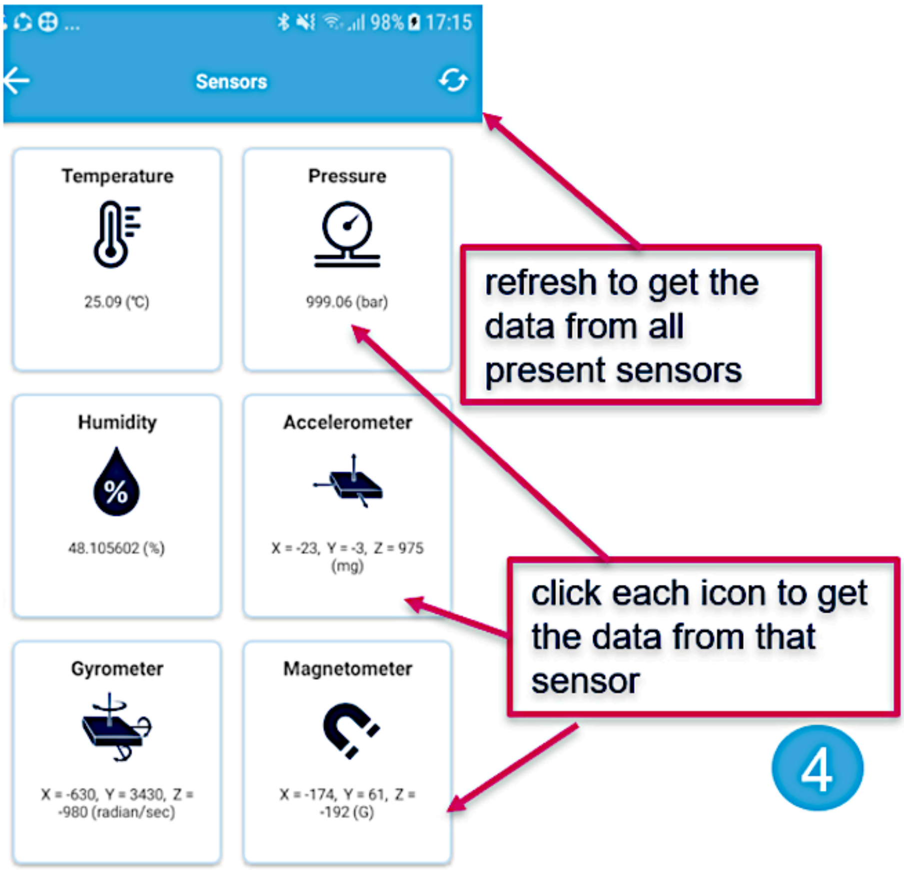


Figure 24. Sensor Client Model app screenshot (4 of 4)



Appendix A Initializing sensor structures

This appendix provides a way of initializing sensor structures corresponding to the examples available in the SDK. It includes two examples: one for single sensor nodes and one for multi-sensor nodes. The SDK already contains similar examples supported on standard evaluation boards and no change is required in the sensor_cfg_usr.h file.

A.1 Example 1

A.1.1 Initializing temperature sensor node

This example takes into consideration a sensor node which reports the present ambient temperature and has two sensor settings. The following sub-sections provide example values of sensor initialization structures.

A.1.1.1 sensor_server_init_params_t (represented by SENSOR_SERVER_INIT_PARAMS)

Table 35. Example values of SensorServerInitParams

Field	Represented by	Value
sensorsCount	TOTAL_SENSORS_COUNT	1
sensorInitParams[0]	SENSOR1_INIT_PARAMS	Refer to Table 36 for the values of its members

A.1.1.1.1 sensorInitParams[0] (represented by SENSOR1_INIT_PARAMS)

Table 36. Example values of sensorInitParams[0]

Field	Represented by	Value
elementIdx	SENSOR1_ELEMENT_IDX	0
propertyId	SENSOR1_PROPERTY_ID	PRESENT_AMBIENT_TEMPERATURE_PID
positiveTolerance	SENSOR1_POSITIVE_TOLERANCE	SENSOR_POSITIVE_TOLERANCE_UNSPECIFIED
negativeTolerance	SENSOR1_NEGATIVE_TOLERANCE	SENSOR_NEGATIVE_TOLERANCE_UNSPECIFIED
samplingFunction	SENSOR1_SAMPLING_FUNCTION	SENSOR_SAMPLING_FUNC_UNSPECIFIED
measurementPeriod	SENSOR1_MEASUREMENT_PERIOD	SENSOR_MEASUREMENT_PERIOD_NA
updateInterval	SENSOR1_UPDATE_INTERVAL	SENSOR_UPDATE_INTERVAL_NA
dataLength	SENSOR1_DATA_LENGTH	1
cadenceState	SENSOR1_CADENCE_STATE	SENSOR_CADENCE_SUPPORTED
valuesRange	SENSOR1_VALUES_RANGE	254
settingsCount	SENSOR1_SETTINGS_COUNT	2
settings[0]	SENSOR1_SETTINGS1_INIT_PARAMS	Refer to Table 37 for values of its members
settings[1]	SENSOR1_SETTINGS2_INIT_PARAMS	Refer to Table 38 for values of its members
seriesCount	SENSOR1_SERIES_COUNT	0
SeriesColumn[0]	{0}	Refer to Table 39 for values of its members

A.1.1.1.1.1 settings[0] (represented by SENSOR1_SETTINGS1_INIT_PARAMS)

Table 37. Example values of settings[0]

Field	Represented by	Value
`settingPropertyId`	`SENSOR1_SETTING1_PROPERTY_ID`	0x00AD
`settingAccess`	`SENSOR1_SETTING1_ACCESS`	`SENSOR_SETTING_ACCESS_READ`
`settingRaw`	`SENSOR1_SETTING1_RAW`	0

A.1.1.1.1.2 settings[1] (represented by SENSOR1_SETTINGS2_INIT_PARAMS)

Table 38. Example values of settings[1]

Field	Represented by	Value
`settingPropertyId`	`SENSOR1_SETTING2_PROPERTY_ID`	0x00BB
`settingAccess`	`SENSOR1_SETTING2_ACCESS`	`SENSOR_SETTING_ACCESS_READ_WRITE`
`settingRaw`	`SENSOR1_SETTING2_RAW`	0

A.1.1.1.1.3 seriesColumn[0] (represented by {0})

Table 39. Example values of seriesColumn[0]

Field	Represented by	Value
`rawX`	{0}	NA
`columnWidth`		NA

Each field value is represented by an appropriate macro for easy understanding. The `sensor_server_init_params_t` is initialized in appli_sensor.c file:

```
const sensor server init params t SensorServerInitParams = SENSOR SERVER INIT PARAMS;
```

where `SENSOR_SERVER_INIT_PARAMS` is:

```
#define SENSOR_SERVER_INIT_PARAMS \
{ \
   TOTAL_SENSORS_COUNT, \
    { \
       SENSOR1_INIT_PARAMS, \
    } \
}
```

where SENSOR1_INIT_PARAMS is:

```
#define SENSOR1_INIT_PARAMS \
{ \
   SENSOR1_ELEMENT_IDx, \
   SENSOR1_PROPERTY_ID, \
   SENSOR1_POSITIVE_TOLERANCE, \
   SENSOR1_NEGATIVE_TOLERANCE, \
   SENSOR1_SAMPLING_FUNCTION, \
   SENSOR1_MEASUREMENT_PERIOD, \
   SENSOR1_UPDATE_INTERVAL, \
   SENSOR1_DATA_LENGTH, \
   SENSOR1_CADENCE_STATE, \
   SENSOR1_VALUES_RANGE, \
   SENSOR1_SETTINGS_COUNT, \
   { \
      SENSOR1_SETTINGS1_INIT_PARAMS, \
      SENSOR1_SETTINGS2_INIT_PARAMS \
      }, \
      SENSOR1_SERIES_COUNT, \
      {\
          {0}\
      }\
}
```

where SENSOR1_SETTINGS1_INIT_PARAMS is:

```
#define SENSOR1_SETTINGS1_INIT_PARAMS \
{\
   SENSOR1_SETTING1_PROPERTY_ID, \
   SENSOR1_SETTING1_ACCESS, \
   SENSOR1_SETTING1_RAW \
}
```

and SENSOR1_SETTINGS2_INIT_PARAMS is:

```
#define SENSOR1_SETTINGS2_INIT_PARAMS \
{\
   SENSOR1_SETTING2_PROPERTY_ID, \
   SENSOR1_SETTING2_ACCESS, \
   SENSOR1_SETTING2_RAW \
}
```

A.1.2 Initializing additional parameters

Table 40. Initialization of additional parameters

Field	Value	Description
SENSOR_MAX_SETTINGS_COUNT	2	Maximum of all the values of settings count per sensor. Minimum value is 1
SENSOR_MAX_SERIES_COUNT	1	Maximum of all the value of series count per sensor. Minimum value is 1
TOTAL_SENSORS_COUNT	1	Total sensors count
TOTAL_SENSOR_SETTINGS_COUNT	2	Total sensor settings count of all the sensors
TOTAL_SENSOR_SERIES_COLUMN_COUNT	0	Total sensor series column count of all the sensors

A.2 Example 2

A.2.1 Initializing a multi-sensor node

In a node with multiple sensors (present ambient temperature, average ambient temperature, motion sensed, total device energy use, total device runtime), each sensor may support additional sensor settings or have no settings. Similarly, some of them may support sensor cadence while others may not.

Example values of sensor initialization structures are listed in Table 41 to Table 57. Pseudo example code is available in Section A.2.2 .

A.2.1.1 sensor_server_init_params_t (represented by SENSOR_SERVER_INIT_PARAMS)

Table 41. Example values of SensorServerInitParams

Field	Represented by	Value
sensorsCount	TOTAL_SENSORS_COUNT	5
sensorInitParams[0]	SENSOR1_INIT_PARAMS	Refer to Table 42 for the values of its members
sensorInitParams[1]	SENSOR2_INIT_PARAMS	Refer to Table 46 for the values of its members
sensorInitParams[2]	SENSOR3_INIT_PARAMS	Refer to Table 49 for the values of its members
sensorInitParams[3]	SENSOR4_INIT_PARAMS	Refer to Table 52 for the values of its members
sensorInitParams[4]	SENSOR5_INIT_PARAMS	Refer to Table 55 for the values of its members

A.2.1.1.1 sensorInitParams[0] (represented by SENSOR1_INIT_PARAMS)

Table 42. Example values of sensorInitParams[0]

Field	Represented by	Value
elementIdx	SENSOR1_ELEMENT_IDX	0
propertyId	SENSOR1_PROPERTY_ID	PRESENT_AMBIENT_TEMPERATURE_PID
positiveTolerance	SENSOR1_POSITIVE_TOLERANCE	SENSOR_POSITIVE_TOLERANCE_UNSPECIFIED
negativeTolerance	SENSOR1_NEGATIVE_TOLERANCE	SENSOR_NEGATIVE_TOLERANCE_UNSPECIFIED
samplingFunction	SENSOR1_SAMPLING_FUNCTION	SENSOR_SAMPLING_FUNC_UNSPECIFIED
measurementPeriod	SENSOR1_MEASUREMENT_PERIOD	SENSOR_MEASUREMENT_PERIOD_NA
updateInterval	SENSOR1_UPDATE_INTERVAL	SENSOR_UPDATE_INTERVAL_NA
dataLength	SENSOR1_DATA_LENGTH	1
cadenceState	SENSOR1_CADENCE_STATE	SENSOR_CADENCE_SUPPORTED
valuesRange	SENSOR1_VALUES_RANGE	254
settingsCount	SENSOR1_SETTINGS_COUNT	2
settings[0]	SENSOR1_SETTINGS1_INIT_PARAMS	Refer to Table 43 for the values of its members
settings[1]	SENSOR1_SETTINGS2_INIT_PARAMS	Refer to Table 44 for the values of its members
seriesCount	SENSOR1_SERIES_COUNT	0
SeriesColumn[0]	{0}	Refer to Table 45 for the values of its members

A.2.1.1.1.1 settings[0] (represented by SENSOR1_SETTINGS1_INIT_PARAMS)

Table 43. Example values of settings[0]

Field	Represented by	Value
settingPropertyId	SENSOR1_SETTING1_PROPERTY_ID	0x00AD
settingAccess	SENSOR1_SETTING1_ACCESS	SENSOR_SETTING_ACCESS_READ
settingRaw	SENSOR1_SETTING1_RAW	0

A.2.1.1.1.2 settings[1] (represented by SENSOR1_SETTINGS2_INIT_PARAMS)

Table 44. Example values of settings[1]

Field	Represented by	Value
settingPropertyId	SENSOR1_SETTING2_PROPERTY_ID	0x00BB
settingAccess	SENSOR1_SETTING2_ACCESS	SENSOR_SETTING_ACCESS_READ_WRITE
settingRaw	SENSOR1_SETTING2_RAW	0

A.2.1.1.1.3 seriesColumn[0] (represented by {0})

Table 45. Example values of seriesColumn[0]

Field	Represented by	Value
rawX	{0}	NA
columnWidth		NA

A.2.1.1.2 sensorInitParams[1] (represented by SENSOR2_INIT_PARAMS)

Table 46. Example values of sensorInitParams[1]

Field	Represented by	Value
elementIdx	SENSOR2_ELEMENT_IDX	0
propertyId	SENSOR2_PROPERTY_ID	AVERAGE_AMBIENT_TEMPERATURE_IN_A_PERIOD_OF_DAY_PID
positiveTolerance	SENSOR2_POSITIVE_TOLERANCE	SENSOR_POSITIVE_TOLERANCE_UNSPECIFIED
negativeTolerance	SENSOR2_NEGATIVE_TOLERANCE	SENSOR_NEGATIVE_TOLERANCE_UNSPECIFIED
samplingFunction	SENSOR2_SAMPLING_FUNCTION	SENSOR_SAMPLING_FUNC_UNSPECIFIED
measurementPeriod	SENSOR2_MEASUREMENT_PERIOD	SENSOR_MEASUREMENT_PERIOD_NA
updateInterval	SENSOR2_UPDATE_INTERVAL	SENSOR_UPDATE_INTERVAL_NA
dataLength	SENSOR2_DATA_LENGTH	1
cadenceState	SENSOR2_CADENCE_STATE	SENSOR_CADENCE_NOT_SUPPORTED
valuesRange	SENSOR2_VALUES_RANGE	0 - NA
settingsCount	SENSOR2_SETTINGS_COUNT	0

Field	Represented by	Value
settings[0]	{0}	Refer to Table 47 for the values of its members
seriesCount	SENSOR2_SERIES_COUNT	0
SeriesColumn[0]	{0}	Refer to Table 48 for the values of its members

A.2.1.1.2.1 settings[0] (represented by {0})

Table 47. Example values of settings[0]

Field	Represented by	Value
settingPropertyId	{0}	NA
settingAccess		NA
settingRaw		NA

A.2.1.1.2.2 seriesColumn[0] (represented by {0})

Table 48. Example values of seriesColumn[0]

Field	Represented by	Value
rawX	{0}	NA
columnWidth		NA

A.2.1.1.3 sensorInitParams[2] (represented by SENSOR3_INIT_PARAMS)

Table 49. Example values of sensorInitParams[2]

Field	Represented by	Value
elementIdx	SENSOR3_ELEMENT_IDX	0
propertyId	SENSOR3_PROPERTY_ID	MOTION_SENSED_PID
positiveTolerance	SENSOR3_POSITIVE_TOLERANCE	SENSOR_POSITIVE_TOLERANCE_UNSPECIFIED
negativeTolerance	SENSOR3_NEGATIVE_TOLERANCE	SENSOR_NEGATIVE_TOLERANCE_UNSPECIFIED
samplingFunction	SENSOR3_SAMPLING_FUNCTION	SENSOR_SAMPLING_FUNC_UNSPECIFIED
measurementPeriod	SENSOR3_MEASUREMENT_PERIOD	SENSOR_MEASUREMENT_PERIOD_NA
updateInterval	SENSOR3_UPDATE_INTERVAL	SENSOR_UPDATE_INTERVAL_NA
dataLength	SENSOR3_DATA_LENGTH	1
cadenceState	SENSOR3_CADENCE_STATE	SENSOR_CADENCE_NOT_SUPPORTED
valuesRange	SENSOR3_VALUES_RANGE	0 - NA
settingsCount	SENSOR3_SETTINGS_COUNT	1
settings[0]	SENSOR3_SETTINGS1_INIT_PARAMS	Refer to Table 50 for the values of its members
seriesCount	SENSOR3_SERIES_COUNT	0
SeriesColumn[0]	{0}	Refer to Table 51 for the values of its members

A.2.1.1.3.1 settings[0] (represented by SENSOR3_SETTINGS1_INIT_PARAMS)

Table 50. Example values of settings[0]

Field	Represented by	Value
settingPropertyId	SENSOR1_SETTING1_PROPERTY_ID	MOTION_THRESHOLD_PID
settingAccess	SENSOR1_SETTING1_ACCESS	SENSOR_SETTING_ACCESS_READ_WRITE
settingRaw	SENSOR1_SETTING1_RAW	0

A.2.1.1.3.2 seriesColumn[0] (represented by {0})

Table 51. Example values of seriesColumn[0]

Field	Represented by	Value
rawX	{0}	NA
columnWidth		NA

A.2.1.1.4 sensorInitParams[3] (represented by SENSOR4_INIT_PARAMS)

Table 52. Example values of sensorInitParams[3]

Field	Represented by	Value
elementIdx	SENSOR4_ELEMENT_IDX	0
propertyId	SENSOR4_PROPERTY_ID	TOTAL_DEVICE_ENERGY_USE_PID
positiveTolerance	SENSOR4_POSITIVE_TOLERANCE	SENSOR_POSITIVE_TOLERANCE_UNSPECIFIED
negativeTolerance	SENSOR4_NEGATIVE_TOLERANCE	SENSOR_NEGATIVE_TOLERANCE_UNSPECIFIED
samplingFunction	SENSOR4_SAMPLING_FUNCTION	SENSOR_SAMPLING_FUNC_UNSPECIFIED
measurementPeriod	SENSOR4_MEASUREMENT_PERIOD	SENSOR_MEASUREMENT_PERIOD_NA
updateInterval	SENSOR4_UPDATE_INTERVAL	SENSOR_UPDATE_INTERVAL_NA
dataLength	SENSOR4_DATA_LENGTH	4
cadenceState	SENSOR4_CADENCE_STATE	SENSOR_CADENCE_NOT_SUPPORTED
valuesRange	SENSOR4_VALUES_RANGE	0 – NA
settingsCount	SENSOR4_SETTINGS_COUNT	0
Settings[0]	{0}	Refer to Table 53 for the values of its members
seriesCount	SENSOR4_SERIES_COUNT	0
SeriesColumn[0]	{0}	Refer to Table 54 for the values of its members

A.2.1.1.4.1 settings[0] (represented by {0})

Table 53. Example values of settings[0]

Field	Represented by	Value
settingPropertyId	{0}	NA
settingAccess		NA
settingRaw		NA

A.2.1.1.4.2 seriesColumn[0] (represented by {0})

Table 54. Example values of seriesColumn[0]

Field	Represented by	Value
rawX	{0}	NA
columnWidth		NA

A.2.1.1.5 sensorInitParams[4] (represented by SENSOR5_INIT_PARAMS)

Table 55. Example values of sensorInitParams[4]

Field	Represented by	Value
elementIdx	SENSOR5_ELEMENT_IDX	0
propertyId	SENSOR5_PROPERTY_ID	TOTAL_DEVICE_RUNTIME_PID
positiveTolerance	SENSOR5_POSITIVE_TOLERANCE	SENSOR_POSITIVE_TOLERANCE_UNSPECIFIED
negativeTolerance	SENSOR5_NEGATIVE_TOLERANCE	SENSOR_NEGATIVE_TOLERANCE_UNSPECIFIED
samplingFunction	SENSOR5_SAMPLING_FUNCTION	SENSOR_SAMPLING_FUNC_UNSPECIFIED
measurementPeriod	SENSOR5_MEASUREMENT_PERIOD	SENSOR_MEASUREMENT_PERIOD_NA
updateInterval	SENSOR5_UPDATE_INTERVAL	SENSOR_UPDATE_INTERVAL_NA
dataLength	SENSOR5_DATA_LENGTH	3
cadenceState	SENSOR5_CADENCE_STATE	SENSOR_CADENCE_SUPPORTED
valuesRange	SENSOR5_VALUES_RANGE	254
settingsCount	SENSOR5_SETTINGS_COUNT	0
settings[0]	SENSOR5_SETTINGS1_INIT_PARAMS	Refer to Table 56 for the values of its members
seriesCount	SENSOR5_SERIES_COUNT	0
SeriesColumn[0]	{0}	Refer to Table 57 for the values of its members

A.2.1.1.5.1 settings[0] (represented by {0})

Table 56. Example values of settings[0]

Field	Represented by	Value
settingPropertyId	{0}	NA
settingAccess		NA
settingRaw		NA

A.2.1.1.5.2 seriesColumn[0] (represented by {0})

Table 57. Example values of seriesColumn[0]

Field	Represented by	Value
rawX	{0}	NA
columnWidth		NA

Each field value is represented by an appropriate macro for easy understanding. The `sensor_server_init_params_t` is initialized in appli_sensor.c file:

```
const sensor server init params t SensorServerInitParams = SENSOR SERVER INIT PARAMS;
```

where `SENSOR_SERVER_INIT_PARAMS` is:

```
#define SENSOR_SERVER_INIT_PARAMS \
{ \
   TOTAL_SENSORS_COUNT, \
    { \
       SENSOR1_INIT_PARAMS, \
       SENSOR2_INIT_PARAMS, \
       SENSOR3_INIT_PARAMS, \
       SENSOR4_INIT_PARAMS, \
       SENSOR5_INIT_PARAMS, \
    } \
}
```

where `SENSOR1_INIT_PARAMS` is:

```
#define SENSOR1_INIT_PARAMS \
{ \
   SENSOR1_ELEMENT_IDx, \
   SENSOR1_PROPERTY_ID, \
   SENSOR1_POSITIVE_TOLERANCE, \
   SENSOR1_NEGATIVE_TOLERANCE, \
   SENSOR1_SAMPLING_FUNCTION, \
   SENSOR1_MEASUREMENT_PERIOD, \
   SENSOR1_UPDATE_INTERVAL, \
   SENSOR1_DATA_LENGTH, \
   SENSOR1_CADENCE_STATE, \
   SENSOR1_VALUES_RANGE, \
   SENSOR1_SETTINGS_COUNT, \
   { \
      SENSOR1_SETTINGS1_INIT_PARAMS, \
      SENSOR1_SETTINGS2_INIT_PARAMS \
      }, \
      SENSOR1_SERIES_COUNT, \
      {\
          {0}\
      }\
}
```

where `SENSOR2_INIT_PARAMS` is:

```
#define SENSOR2_INIT_PARAMS \
{ \
   SENSOR2_ELEMENT_IDx, \
   SENSOR2_PROPERTY_ID, \
   SENSOR2_POSITIVE_TOLERANCE, \
   SENSOR2_NEGATIVE_TOLERANCE, \
   SENSOR2_SAMPLING_FUNCTION, \
   SENSOR2_MEASUREMENT_PERIOD, \
   SENSOR2_UPDATE_INTERVAL, \
   SENSOR2_DATA_LENGTH, \
   SENSOR2_CADENCE_STATE, \
   SENSOR2_VALUES_RANGE, \
   SENSOR2_SETTINGS_COUNT, \
   { \
      SENSOR2_SETTINGS1_INIT_PARAMS, \
      SENSOR2_SETTINGS2_INIT_PARAMS \
      }, \
      SENSOR2_SERIES_COUNT, \
      {\
          {0}\
      }\
}
```

where `SENSOR3_INIT_PARAMS` is:

```
#define SENSOR3_INIT_PARAMS \
{ \
   SENSOR3_ELEMENT_IDx, \
   SENSOR3_PROPERTY_ID, \
   SENSOR3_POSITIVE_TOLERANCE, \
   SENSOR3_NEGATIVE_TOLERANCE, \
   SENSOR3_SAMPLING_FUNCTION, \
   SENSOR3_MEASUREMENT_PERIOD, \
   SENSOR3_UPDATE_INTERVAL, \
   SENSOR3_DATA_LENGTH, \
   SENSOR3_CADENCE_STATE, \
   SENSOR3_VALUES_RANGE, \
   SENSOR3_SETTINGS_COUNT, \
   { \
      SENSOR3_SETTINGS1_INIT_PARAMS, \
      SENSOR3_SETTINGS2_INIT_PARAMS \
      }, \
      SENSOR3_SERIES_COUNT, \
      {\
          {0}\
      }\
}
```

where SENSOR4_INIT_PARAMS is:

```
#define SENSOR4_INIT_PARAMS \
{ \
   SENSOR4_ELEMENT_IDx, \
   SENSOR4_PROPERTY_ID, \
   SENSOR4_POSITIVE_TOLERANCE, \
   SENSOR4_NEGATIVE_TOLERANCE, \
   SENSOR4_SAMPLING_FUNCTION, \
   SENSOR4_MEASUREMENT_PERIOD, \
   SENSOR4_UPDATE_INTERVAL, \
   SENSOR4_DATA_LENGTH, \
   SENSOR4_CADENCE_STATE, \
   SENSOR4_VALUES_RANGE, \
   SENSOR4_SETTINGS_COUNT, \
   { \
      SENSOR4_SETTINGS1_INIT_PARAMS, \
      SENSOR4_SETTINGS2_INIT_PARAMS \
      }, \
      SENSOR4_SERIES_COUNT, \
      {\
          {0}\
      }\
}
```

where SENSOR5_INIT_PARAMS is:

```
#define SENSOR5_INIT_PARAMS \
{ \
   SENSOR5_ELEMENT_IDx, \
   SENSOR5_PROPERTY_ID, \
   SENSOR5_POSITIVE_TOLERANCE, \
   SENSOR5_NEGATIVE_TOLERANCE, \
   SENSOR5_SAMPLING_FUNCTION, \
   SENSOR5_MEASUREMENT_PERIOD, \
   SENSOR5_UPDATE_INTERVAL, \
   SENSOR5_DATA_LENGTH, \
   SENSOR5_CADENCE_STATE, \
   SENSOR5_VALUES_RANGE, \
   SENSOR5_SETTINGS_COUNT, \
   { \
      SENSOR5_SETTINGS1_INIT_PARAMS, \
      SENSOR5_SETTINGS2_INIT_PARAMS \
      }, \
      SENSOR5_SERIES_COUNT, \
      {\
          {0}\
      }\
}
```

A.2.2 Pseudo example code

```
/**
* Maximum count of settings that can be supported by a sensor
* E.g., 5 sensors
*  1st sensor has 2 settings
*  2nd sensor has 3 settings
*  3rd sensor has 2 settings
*  4th sensor has 1 setting
*  5th sensor has 2 settings
*  This value is max(2, 3, 2, 1, 2) = 3
*  value is >=1
*/
#define SENSOR_MAX_SETTINGS_COUNT              2

/**
* Maximum count of series column that is supported by a sensor
* E.g., 2 sensors supporting series column
*  One sensor supports 2 columns while other sensor supports 20 columns
*  This value is max(2, 20) = 20
*  value is >=1
*/
#define SENSOR_MAX_SERIES_COUNT                1

/**
* Total sensors count on all elements
* It is sum of sensors count on all elements
* Sensor init fails in case of mismatch with sensor server initialization parameters
*/
#define TOTAL_SENSORS_COUNT                    5

/**
* Sum of sensor settings on all sensors on all elements
* Sensor init fails in case of mismatch with sensor server initialization parameters
*/
#define TOTAL_SENSOR_SETTINGS_COUNT            3

/**
* Sum of sensor series columns on all sensors on all elements
* Sensor init fails in case of mismatch with sensor server initialization parameters
*/
#define TOTAL_SENSOR_SERIES_COLUMN_COUNT       0


/**
* This structure contains sensor setting initialization parameters
*/
typedef struct
{
    uint16_t settingPropertyId;
    uint8_t settingAccess;
    uint32_t settingRaw;
}sensor_settings_init_params_t;


/**
* This structure contains sensor series column initialization parameters
*/
typedef struct
{
    uint32_t rawX;
    uint32_t columnWidth;
}sensor_series_column_init_params_t;


/**
* This structure contains sensor initialization parameters
*/
typedef struct
```

```
{
    uint8_t elementIdx;
    uint16_t propertyId;
    uint16_t positiveTolerance;
    uint16_t negativeTolerance;
    uint8_t samplingFunction;
    uint8_t measurementPeriod;
    uint8_t updateInterval;
    uint8_t dataLength;
    uint8_t cadenceState;
    uint32_t valuesRange;
    uint8_t settingsCount;
    sensor_settings_init_params_t settings[SENSOR_MAX_SETTINGS_COUNT];
    uint16_t seriesCount;
    sensor_series_column_init_params_t seriesColumn[SENSOR_MAX_SERIES_COUNT];
}sensor_init_params_t;


/**
* This structure contains sensor server initialization parameters
*/
typedef struct
{
    uint8_t sensorsCount;
    sensor_init_params_t sensorInitParams[TOTAL_SENSORS_COUNT];
} sensor_server_init_params_t;


/**
* Below section represents initialization parameters of sensors supported
* Define sensors in ascending order of element index followed by ascending
* order of Property IDs else initialization of sensor structure would fail
* Single element can support one instance of sensor PID, there can't be multiple
* instances of same PID on same element
* For e.g. 10 sensors with PID (PID1 < PIDn ... < PID7) supported on 3 elements
* with element index (0, 1 and 2) in below fashion
* Element index 0 supports sensors corresponding to PID3, PID4, PID6 and PID7
* Element index 1 supports sensors corresponding to PID2, PID4, PID5 and PID6
* Element index 2 supports sensors corresponding to PID1, PID5
* Corrector order of naming sensors (SENSORX) is
* Element index 0 -> SENSOR1(PID3), SENSOR2(PID4), SENSOR3(PID6) and SENSOR4(PID7)
* Element index 1 -> SENSOR5(PID2), SENSOR6(PID4), SENSOR7(PID5), and SENSOR8(PID6)
* Element index 2 -> SENSOR9(PID1), and SENSOR10(PID5)
*/

/* Sensor 1 initialization */

/**
* Element index for SENSOR1
* varies from 0 to n-1 (n = number of elements)
*/
#define SENSOR1_ELEMENT_IDX                                                 0

/**
* Property ID of sensor, identifies device characteristics and other features
* Defined by Mesh Device Properties or a custom value
* 16 bit value
* 0x0000 - Prohibited
*/
#define SENSOR1_PROPERTY_ID                     PRESENT_AMBIENT_TEMPERATURE_PID
#define SENSOR1_POSITIVE_TOLERANCE              SENSOR_POSITIVE_TOLERANCE_UNSPECIFIED
#define SENSOR1_NEGATIVE_TOLERANCE              SENSOR_NEGATIVE_TOLERANCE_UNSPECIFIED
#define SENSOR1_SAMPLING_FUNCTION               SENSOR_SAMPLING_FUNC_UNSPECIFIED
#define SENSOR1_MEASUREMENT_PERIOD              SENSOR_MEASUREMENT_PERIOD_NA
#define SENSOR1_UPDATE_INTERVAL                 SENSOR_UPDATE_INTERVAL_NA
#define SENSOR1_DATA_LENGTH                     1
#define SENSOR1_CADENCE_STATE                   SENSOR_CADENCE_SUPPORTED
#define SENSOR1_VALUES_RANGE                    254
#define SENSOR1_SETTINGS_COUNT                  2
```

```
#define SENSOR1_SETTING1_PROPERTY_ID            0x00BB
#define SENSOR1_SETTING1_ACCESS                 SENSOR_SETTING_ACCESS_READ
#define SENSOR1_SETTING1_RAW                    0
#define SENSOR1_SETTING2_PROPERTY_ID            0x00AD
#define SENSOR1_SETTING2_ACCESS                 SENSOR_SETTING_ACCESS_READ_WRITE
#define SENSOR1_SETTING2_RAW                    0
#define SENSOR1_SERIES_COUNT                    0

#define SENSOR1_SETTINGS1_INIT_PARAMS \
{\
  SENSOR1_SETTING1_PROPERTY_ID,\
  SENSOR1_SETTING1_ACCESS,\
  SENSOR1_SETTING1_RAW \
}

#define SENSOR1_SETTINGS2_INIT_PARAMS \
{\
  SENSOR1_SETTING2_PROPERTY_ID,\
  SENSOR1_SETTING2_ACCESS,\
  SENSOR1_SETTING2_RAW \
}

#define SENSOR1_INIT_PARAMS \
{\
  SENSOR1_ELEMENT_IDX,\
  SENSOR1_PROPERTY_ID,\
  SENSOR1_POSITIVE_TOLERANCE,\
  SENSOR1_NEGATIVE_TOLERANCE,\
  SENSOR1_SAMPLING_FUNCTION,\
  SENSOR1_MEASUREMENT_PERIOD,\
  SENSOR1_UPDATE_INTERVAL,\
  SENSOR1_DATA_LENGTH,\
  SENSOR1_CADENCE_STATE,\
  SENSOR1_VALUES_RANGE,\
  SENSOR1_SETTINGS_COUNT,\
  {\
    SENSOR1_SETTINGS1_INIT_PARAMS,\
    SENSOR1_SETTINGS2_INIT_PARAMS\
  },\
  SENSOR1_SERIES_COUNT,\
  {\
    {0}\
  }\
}


/* Sensor 2 initialization */

#define SENSOR2_ELEMENT_IDX                     0
#define SENSOR2_PROPERTY_ID
AVERAGE_AMBIENT_TEMPERATURE_IN_A_PERIOD_OF_DAY_PID
#define SENSOR2_POSITIVE_TOLERANCE              SENSOR_POSITIVE_TOLERANCE_UNSPECIFIED
#define SENSOR2_NEGATIVE_TOLERANCE              SENSOR_NEGATIVE_TOLERANCE_UNSPECIFIED
#define SENSOR2_SAMPLING_FUNCTION               SENSOR_SAMPLING_FUNC_UNSPECIFIED
#define SENSOR2_MEASUREMENT_PERIOD              SENSOR_MEASUREMENT_PERIOD_NA
#define SENSOR2_UPDATE_INTERVAL                 SENSOR_UPDATE_INTERVAL_NA
#define SENSOR2_DATA_LENGTH                     1
#define SENSOR2_CADENCE_STATE                   SENSOR_CADENCE_NOT_SUPPORTED
#define SENSOR2_VALUES_RANGE                    0
#define SENSOR2_SETTINGS_COUNT                  0
#define SENSOR2_SERIES_COUNT                    0

#define SENSOR2_INIT_PARAMS \
{\
  SENSOR2_ELEMENT_IDX,\
  SENSOR2_PROPERTY_ID,\
  SENSOR2_POSITIVE_TOLERANCE,\
  SENSOR2_NEGATIVE_TOLERANCE,\
  SENSOR2_SAMPLING_FUNCTION,\
```

```
  SENSOR2_MEASUREMENT_PERIOD,\
  SENSOR2_UPDATE_INTERVAL,\
  SENSOR2_DATA_LENGTH,\
  SENSOR2_CADENCE_STATE,\
  SENSOR2_VALUES_RANGE,\
  SENSOR2_SETTINGS_COUNT,\
  {\
    {0}\
  },\
  SENSOR2_SERIES_COUNT,\
  {\
    {0}\
  }\
}


/* Sensor 3 initialization */

#define SENSOR3_ELEMENT_IDX                        0
#define SENSOR3_PROPERTY_ID                        MOTION_SENSED_PID
#define SENSOR3_POSITIVE_TOLERANCE                 SENSOR_POSITIVE_TOLERANCE_UNSPECIFIED
#define SENSOR3_NEGATIVE_TOLERANCE                 SENSOR_NEGATIVE_TOLERANCE_UNSPECIFIED
#define SENSOR3_SAMPLING_FUNCTION                  SENSOR_SAMPLING_FUNC_UNSPECIFIED
#define SENSOR3_MEASUREMENT_PERIOD                 SENSOR_MEASUREMENT_PERIOD_NA
#define SENSOR3_UPDATE_INTERVAL                    SENSOR_UPDATE_INTERVAL_NA
#define SENSOR3_DATA_LENGTH                        1
#define SENSOR3_CADENCE_STATE                      SENSOR_CADENCE_NOT_SUPPORTED
#define SENSOR3_VALUES_RANGE                       0
#define SENSOR3_SETTINGS_COUNT                     1
#define SENSOR3_SETTING1_PROPERTY_ID               MOTION_THRESHOLD_PID
#define SENSOR3_SETTING1_ACCESS                    SENSOR_SETTING_ACCESS_READ_WRITE
#define SENSOR3_SETTING1_RAW                       0
#define SENSOR3_SERIES_COUNT                       0

#define SENSOR3_SETTINGS1_INIT_PARAMS \
{\
  SENSOR3_SETTING1_PROPERTY_ID,\
  SENSOR3_SETTING1_ACCESS,\
  SENSOR3_SETTING1_RAW\
}

#define SENSOR3_INIT_PARAMS \
{\
  SENSOR3_ELEMENT_IDX,\
  SENSOR3_PROPERTY_ID,\
  SENSOR3_POSITIVE_TOLERANCE,\
  SENSOR3_NEGATIVE_TOLERANCE,\
  SENSOR3_SAMPLING_FUNCTION,\
  SENSOR3_MEASUREMENT_PERIOD,\
  SENSOR3_UPDATE_INTERVAL,\
  SENSOR3_DATA_LENGTH,\
  SENSOR3_CADENCE_STATE,\
  SENSOR3_VALUES_RANGE,\
  SENSOR3_SETTINGS_COUNT,\
  {\
    SENSOR3_SETTINGS1_INIT_PARAMS\
  },\
  SENSOR3_SERIES_COUNT,\
  {\
    {0}\
  }\
}


/* Sensor 4 initialization */

#define SENSOR4_ELEMENT_IDX                        0
#define SENSOR4_PROPERTY_ID                        TOTAL_DEVICE_ENERGY_USE_PID
#define SENSOR4_POSITIVE_TOLERANCE                 SENSOR_POSITIVE_TOLERANCE_UNSPECIFIED
```

```
#define SENSOR4_NEGATIVE_TOLERANCE              SENSOR_NEGATIVE_TOLERANCE_UNSPECIFIED
#define SENSOR4_SAMPLING_FUNCTION               SENSOR_SAMPLING_FUNC_UNSPECIFIED
#define SENSOR4_MEASUREMENT_PERIOD              SENSOR_MEASUREMENT_PERIOD_NA
#define SENSOR4_UPDATE_INTERVAL                 SENSOR_UPDATE_INTERVAL_NA
#define SENSOR4_DATA_LENGTH                     4
#define SENSOR4_CADENCE_STATE                   SENSOR_CADENCE_NOT_SUPPORTED
#define SENSOR4_VALUES_RANGE                    0
#define SENSOR4_SETTINGS_COUNT                  0
#define SENSOR4_SERIES_COUNT                    0

#define SENSOR4_INIT_PARAMS \
{\
  SENSOR4_ELEMENT_IDX,\
  SENSOR4_PROPERTY_ID,\
  SENSOR4_POSITIVE_TOLERANCE,\
  SENSOR4_NEGATIVE_TOLERANCE,\
  SENSOR4_SAMPLING_FUNCTION,\
  SENSOR4_MEASUREMENT_PERIOD,\
  SENSOR4_UPDATE_INTERVAL,\
  SENSOR4_DATA_LENGTH,\
  SENSOR4_CADENCE_STATE,\
  SENSOR4_VALUES_RANGE,\
  SENSOR4_SETTINGS_COUNT,\
  {\
    {0}\
  },\
  SENSOR4_SERIES_COUNT,\
  {\
    {0}\
  }\
}


/* Sensor 5 initialization */

#define SENSOR5_ELEMENT_IDX                     0
#define SENSOR5_PROPERTY_ID                     TOTAL_DEVICE_RUNTIME_PID
#define SENSOR5_POSITIVE_TOLERANCE              SENSOR_POSITIVE_TOLERANCE_UNSPECIFIED
#define SENSOR5_NEGATIVE_TOLERANCE              SENSOR_NEGATIVE_TOLERANCE_UNSPECIFIED
#define SENSOR5_SAMPLING_FUNCTION               SENSOR_SAMPLING_FUNC_UNSPECIFIED
#define SENSOR5_MEASUREMENT_PERIOD              SENSOR_MEASUREMENT_PERIOD_NA
#define SENSOR5_UPDATE_INTERVAL                 SENSOR_UPDATE_INTERVAL_NA
#define SENSOR5_DATA_LENGTH                     3
#define SENSOR5_CADENCE_STATE                   SENSOR_CADENCE_NOT_SUPPORTED
#define SENSOR5_VALUES_RANGE                    0
#define SENSOR5_SETTINGS_COUNT                  0
#define SENSOR5_SERIES_COUNT                    0

#define SENSOR5_INIT_PARAMS \
{\
  SENSOR5_ELEMENT_IDX,\
  SENSOR5_PROPERTY_ID,\
  SENSOR5_POSITIVE_TOLERANCE,\
  SENSOR5_NEGATIVE_TOLERANCE,\
  SENSOR5_SAMPLING_FUNCTION,\
  SENSOR5_MEASUREMENT_PERIOD,\
  SENSOR5_UPDATE_INTERVAL,\
  SENSOR5_DATA_LENGTH,\
  SENSOR5_CADENCE_STATE,\
  SENSOR5_VALUES_RANGE,\
  SENSOR5_SETTINGS_COUNT,\
  {\
    {0}\
  },\
  SENSOR5_SERIES_COUNT,\
  {\
    {0}\
  }\
}
```

```
/**
* Combined defined of all sensors intialization parameters
*/
#define SENSOR_SERVER_INIT_PARAMS \
{\
  TOTAL_SENSORS_COUNT,\
  {\
    SENSOR1_INIT_PARAMS,\
    SENSOR2_INIT_PARAMS,\
    SENSOR3_INIT_PARAMS,\
    SENSOR4_INIT_PARAMS,\
    SENSOR5_INIT_PARAMS\
  }\
}
```

Appendix B References

- Mesh Model Specification v1.0.1
- Mesh Device Properties v2.0
- STBLEMesh - BLE Mesh application for Android and iOS
- STSW-BNRG-Mesh - Mesh over Bluetooth Low Energy
- STM32CubeWB - STM32Cube MCU Package for STM32WB series
- X-CUBE-BLEMESH1 - Mesh over Bluetooth low energy software expansion for STM32Cube
- FP-SNS-BLEMESH1 - STM32Cube function pack for IoT node with BLE Mesh connectivity and sensor model

Revision history

Table 58. **Document revision history**

Date	Revision	Changes
06-Sep-2021	1	Initial release.

Contents

List of tables

List of figures