AN1300: Understanding the Silicon Labs *Bluetooth*® Mesh Sensor Model Demonstration in SDK v6.x or Higher

The Bluetooth mesh SDK comes with two sample projects that create a wireless network of sensors and sensor clients using Bluetooth mesh technology. The examples assume use of Silicon Labs devices for sensors and sensor clients, and the Silicon Labs Bluetooth Mesh mobile app as provisioner. In this document we discuss the basics of sensor models and describe the related sample applications in the SDK.

KEY POINTS

- Short introduction to Bluetooth mesh sensor model
- Using the sensor example application
- Sensor example application code walkthrough

1. Introduction

This document focuses on explaining the Bluetooth mesh sensor demo, installed as part of the Bluetooth mesh SDK. For the most part, the document centers on the example application and its usage flow, along with an explanation of a key part of the source code. It also includes a brief discussion of some concepts of the sensor model specification that are important for understanding the example.

1.1 Sensor Model

The sensor model is Bluetooth mesh's method of interfacing with sensors. For a list of supported sensor types, refer to the Bluetooth Mesh Device Properties specifications. This model is made up of sensor states including descriptors, settings, cadence, data, and series columns. This model also defines the messages used for setting and reporting these states between client and server.

Sensor Descriptors: The sensor descriptors define the sensor property ID, to indicate the device's sensor type, the positive and negative tolerance of the sensor, the sampling function, the measurement period, and the update interval.

Sensor Settings: The sensor settings state controls the parameters of a sensor, such as sensitivity. The sensor setting property ID determines whether the sensor settings can be read and written as well as a raw setting's size and content.

Sensor Cadence: The sensor cadence state controls how often the sensor data is published. Data can be published either through a trigger or a fast cadence. The trigger can be defined either by the sensor property ID or as a percentage change in the measured value. A fast cadence can be used if the measured valued falls within a specified range.

Sensor Data: The sensor data state is constructed of a sensor property ID and a raw value. Multiple instances are permitted.

Sensor Series Column: Sensor measurements may be organized as arrays, conceptually as columns of data. The sensor series column state is made up of a raw Y value, raw X value, and column width. The sizes and contents of each of these is determined by the sensor property ID.

1.2 Sensor Messages

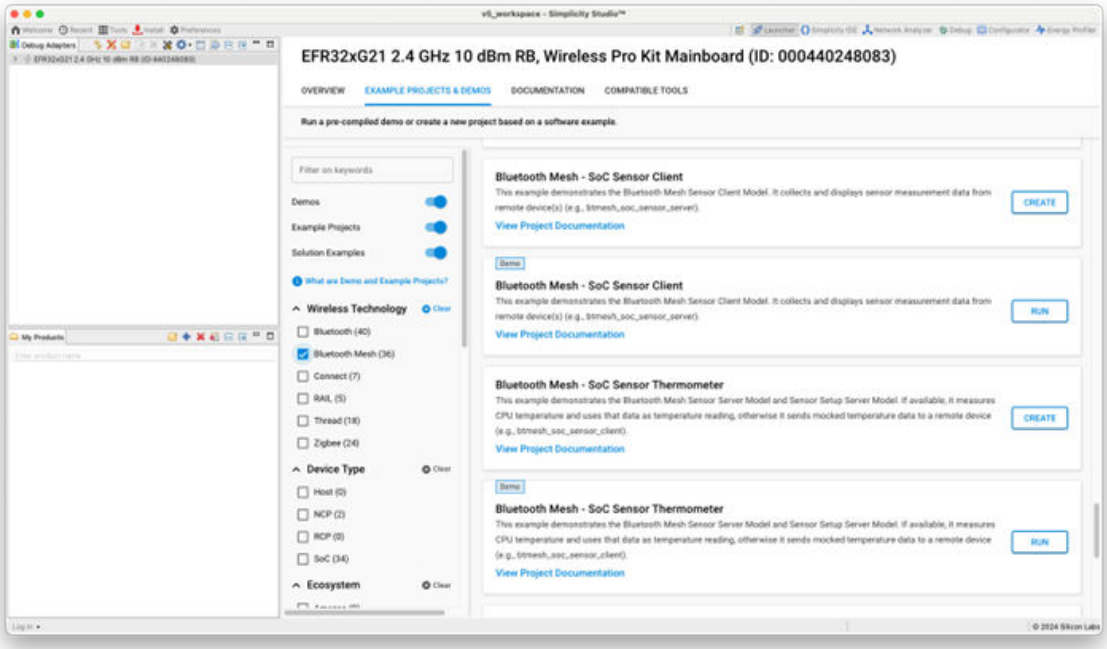
Each state in the sensor model has an acknowledged get message and an unacknowledged status message associated with it. A client requests the status message by sending the get message. In addition, writable sensor states such as the cadence and setting states also have both acknowledged and unacknowledged set messages.

1.3 Sensor Server and Client

The defined models are the Sensor Client, Sensor Server, and Sensor Setup Server. In any element where the Sensor Server is present, the Sensor Setup Server must also be present to allow configuration.

2. Getting Started with the Sensor Model Applications

Two applications are required in order to use the Sensor Model demonstration: **Bluetooth Mesh - SoC Sensor Client** and **Bluetooth Mesh - SoC Sensor Thermometer**. The applications are provided both as prebuilt demo binary images, ready to download and use, and corresponding example projects that you can modify and then build for the target part.



The precompiled demos are only available for a limited set of parts, including selected EFR32xG13, xG21, xG24 and xG27 parts, and BGM13 and BGM220 modules. The examples can be built for any part supported by the Bluetooth Mesh SDK.

Note: EFR32xG22 parts have limited support for Bluetooth Mesh (only LPN is supported).

This chapter describes how to start the mesh sensor model demonstration from either the precompiled demos, or from binaries that you build from the example projects.

2.1 Requirements

The following is required to run the demo.

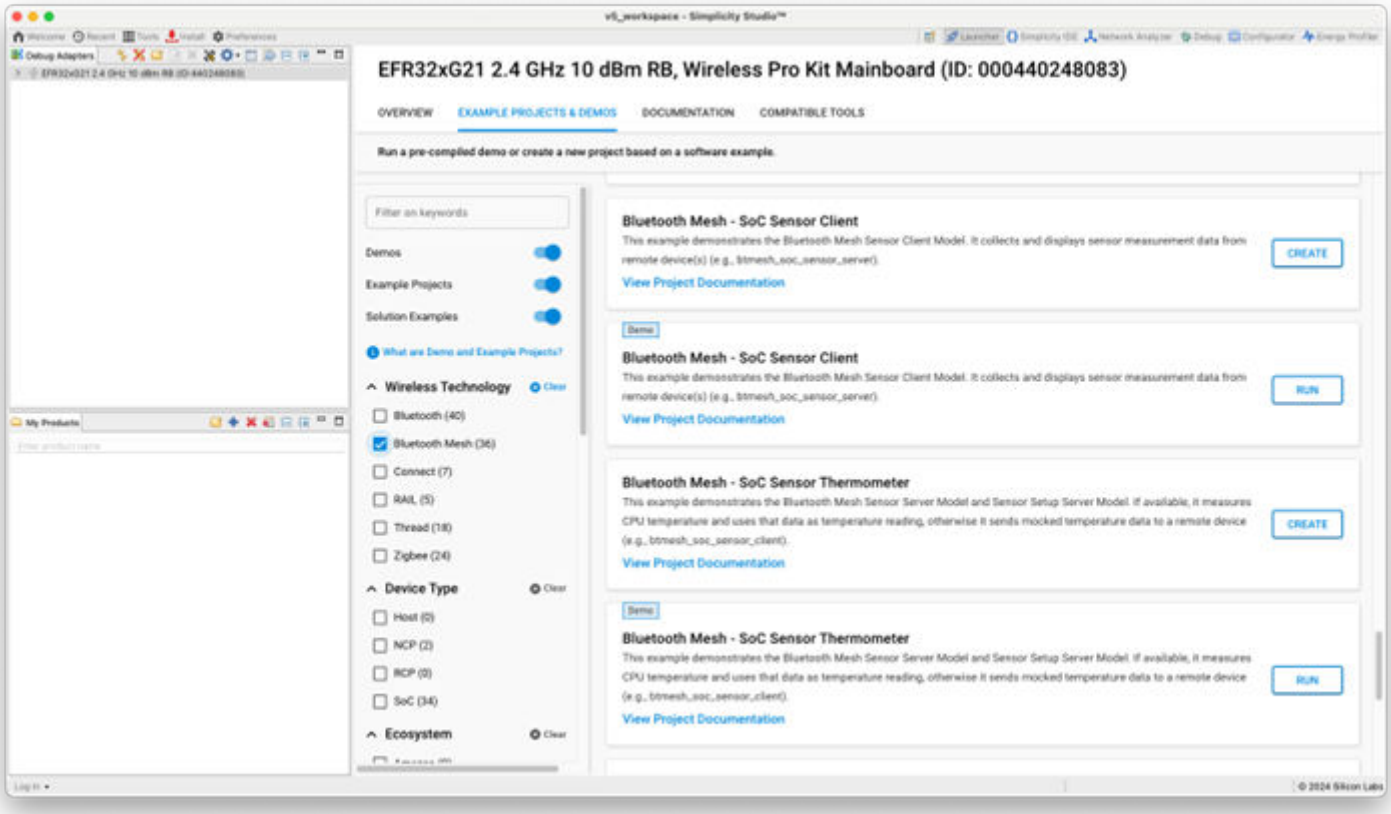
- To see the complete demo in action, two mainboards with a supported board installed, one used for the Client and one for the Server
- Simplicity Studio 5
- Bluetooth Mesh SDK 6.0.0 or later, distributed through Simplicity Studio 5. The prebuilt demos and examples are included in the SDK.
- Silicon Labs Bluetooth Mesh Mobile Application
  - Used for discovering and provisioning devices.
  - Includes network, group, and publish-subscribe setup.
  - Allows device configuration for the sensor models.

Example projects and additional code development can be done with GCC (supplied with Simplicity Studio 5), IAR EWARM, or command line tools.

Simplicity Studio has a network analyzer capable of capturing and decoding Bluetooth mesh packets. See *AN1317: Using Network Analyzer with Bluetooth® Low Energy and Mesh*.

2.2 Starting with the Precompiled Demos

Open Simplicity Studio 5 with a compatible SoC wireless kit connected to the computer. Select the part in Debug Adapters view to open the Launcher perspective. Click the **Example Projects & Demos** tab. To see only the Bluetooth Mesh demos, turn off the Example Projects and Solution Examples, and set the Bluetooth Mesh checkbox. Next to either **Bluetooth Mesh - SoC Sensor Client** or to **Bluetooth Mesh - SoC Sensor Thermometer**, click **RUN**.

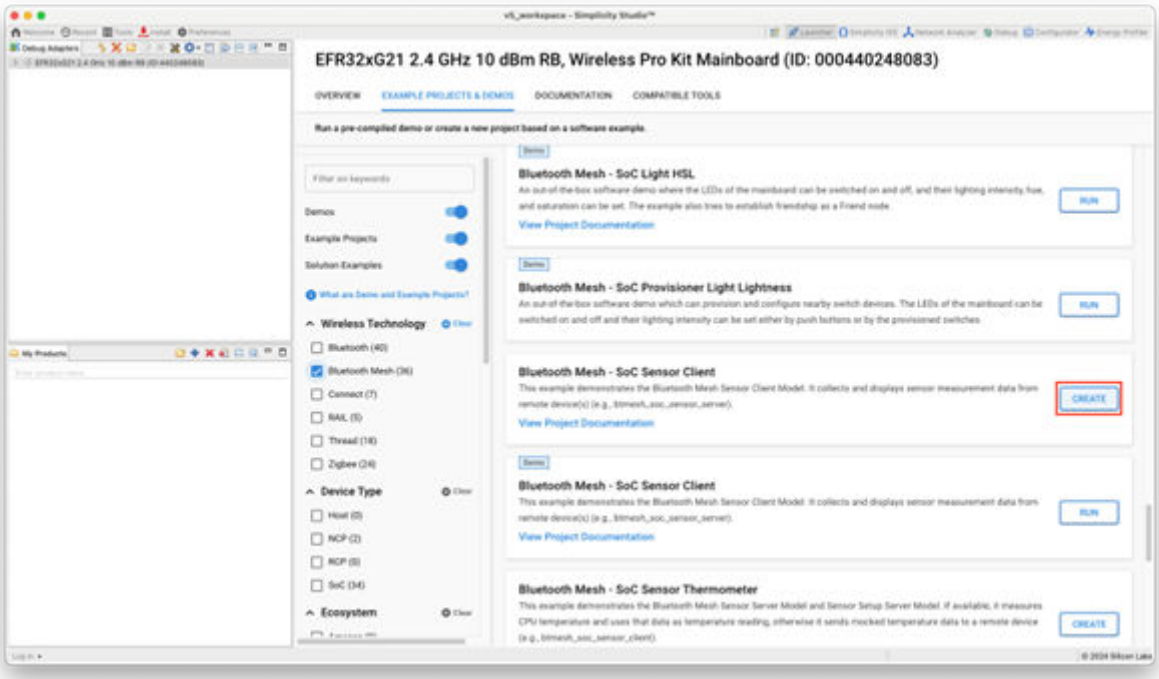


Connect your other device and repeat with the other application.

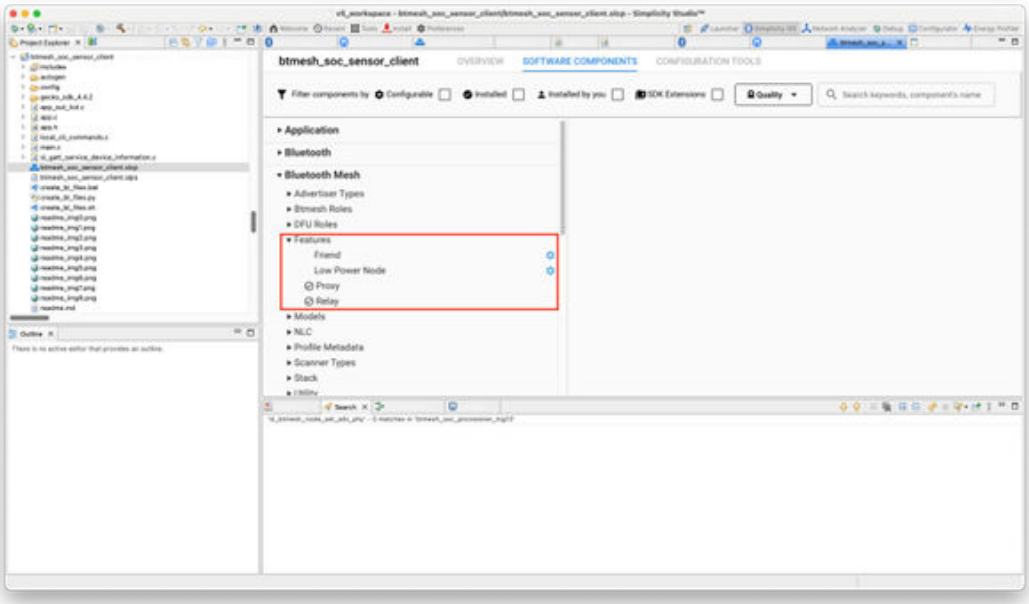
2.3 Starting with Example Projects

This section summarizes how to start the demonstration from example projects. See *QSG176: Bluetooth® Mesh Quick-Start Guide for SDK v2.x and v3.x* for an introduction to configuring and building your own projects, and for a guide to additional resources.

Open Simplicity Studio 5 with a compatible SoC wireless kit connected to the computer. Select the part in Debug Adapters view to open the Launcher perspective. Click the **Example Projects & Demos** tab. Under Technology Type, filter on **Bluetooth Mesh**. Next to the **Bluetooth Mesh - SoC Sensor Client** project, click **Create**.



Modify project settings, and click **Finish** to create the project. Select the project .slcp file if it is not already selected, click the **Software Components** tab, and expand the Bluetooth Mesh components group to see the installed features.



Project files autogenerate, with progress reflected in the lower right of the Simplicity IDE. Build and flash the project. Connect your other device and repeat for the **Bluetooth Mesh - SoC Sensor Thermometer** example.

For more information on how to configure a node in the Bluetooth Mesh SDK v6.x., refer to *UG472: Bluetooth® Mesh Stack and Bluetooth® Mesh Configurator User's Guide for SDK v2.x and Higher*.

3. Bluetooth Mesh Sensor Demonstration

3.1 Mesh Network Implementation

The demonstration implementation process can be divided into four main phases:

1. Unprovisioned mode – After the demo firmware is installed, the device starts in unprovisioned mode.
2. Provisioning – The devices are provisioned to a Bluetooth mesh network and network security is set up.
3. Configuration – The group, publish, and subscribe, and application security are configured.
4. Normal operation – The sensor server(s) can be controlled by the client(s).

In the first phase, all the devices are unprovisioned and transmitting unprovisioned beacons. They do not have any network keys or application keys configured, and publish and subscribe settings are not set. In this state, the devices are simply waiting for the provisioner to assign them into a Bluetooth mesh network, and configure publish and subscribe settings and mesh models. In this state, the devices can be detected by the smartphone application.

In the provisioning phase, the provisioner adds sensor servers and clients to the Bluetooth mesh network. A network key is generated and distributed to the nodes and each node is assigned a unicast address.

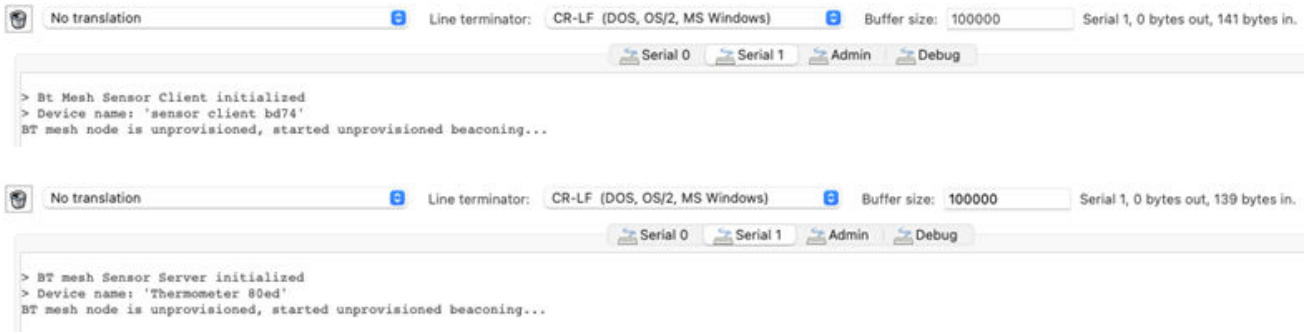
In the configuration phase, the provisioner configures groups, publish, and subscribe settings; generates application keys; and binds mesh models to application keys.

After provisioning and configuration, the Bluetooth mesh network is operational, and clients can be used to configure and request data from the sensors.

3.2 Running the Example

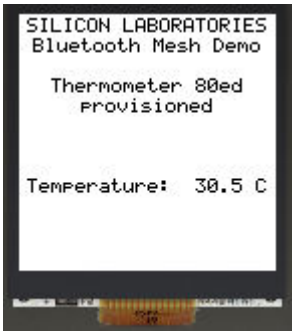
This section assumes you have installed the **BT Mesh – Sensor Client Example** demo binary to one of the devices and the **BT Mesh – Sensor Thermometer Example** to the other.

- Provision and configure the sample apps using the Silicon Labs Bluetooth Mesh mobile application as described in *QSG176: Bluetooth® Mesh Quick-Start Guide for SDK v2.x and v3.x*.
- Open a serial console in Simplicity Studio for each. If you reset the devices, this is what you will observe on the console for the client and server before provisioning:

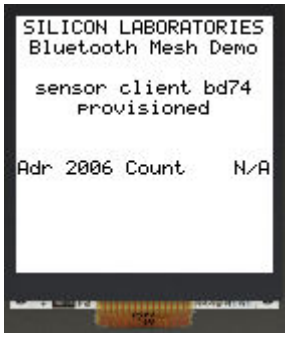


- And this is what you will see in the device display for the client and server, respectively:

- After provisioning, the displays should be as below:



- Press PB0 on the client device to select the temperature sensor. Now the LCD on the client device will display the temperature reported by the server. Pressing it again will show the people count and lightness (these are not added to the server side, but can be installed as a component and configured):



4. Code Walkthrough

As of BT Mesh SDK v6.x, the code structure of both the stack API and the code examples have been reworked. The Bluetooth Mesh API now abstracts away much of the event handling in generated files and allows the user to focus on application development. Much functionality is now hidden in the components, making the user application simpler.

The sections below describe the code in the application source of the examples (app.c and components).

4.1 Unprovisioned Mode, Provisioning, and Configuration

In unprovisioned mode, both examples behave the same way. The unprovisioned device simply starts sending unprovisioned beacons and waits for a provisioner to provision and configure it.

When the `app_init` is being called, the application checks if a button is pressed in the `handle_reset_conditions` function. If yes, -depending on the pressed button(s)- it calls either the function `sl_btmesh_initiate_full_reset`(), which halts the system and performs a factory reset by erasing the NVM3 storage, or the function `sl_btmesh_initiate_node_reset`(), which does the same, except performs full NVM storage erasure. The factory reset is also done after receiving a node_reset event (`sl_btmesh_evt_node_reset_id`). This is handled within the Factory Reset component. If no button is pressed, then the name of the device is set based on the Bluetooth address, within the function `sl_btmesh_provisionee_on_init`(), when it is called to signal the initialization of the Bluetooth mesh node stack.

The callback `sl_btmesh_provisionee_on_init()` indicates that the Bluetooth mesh node stack initialization is complete. In the background, the `sl_btmesh_evt_node_initialized_id` is being handled within the Provisionee component. The application first checks the provisioning status. If the node is not provisioned (the default state when the device is first powered up after programming), then the application starts unprovisioned beaconing by calling `sl_btmesh_node_start_unprov_beaconing`().

TheAPI call `sl_btmesh_node_start_unprov_beaconing`takes one parameter (bearer) that selects which bearers are used (PB- ADV, PB-GATT, or both). In this example, both bearers are used. Because the PB-GATT bearer is enabled, the device will begin advertising its provisioning GATT service. This allows the smartphone application to detect unprovisioned nodes.

When unprovisioned beaconing has been started the application waits for the provisioner (in this case, the smartphone app) to start provisioning. Start of provisioning is indicatedwith the event `sl_btmesh_evt_node_provisioning_started_id`(see sl_btmesh_provisioning_decorator.c). This is handled in the application code through the `sl_btmesh_on_node_provisioning_started()`callback.

During provisioning, no actions are required from the user application. The configuration of network keys and other operations are handledautomatically by the Bluetooth mesh stack. Both the sensor server and client applications simply start blinking the two LEDs on the WSTK to indicate that provisioning is in progress. Then they wait for the event `sl_btmesh_evt_node_provisioned_id` (see sl_btmesh_provisioning_decorator.c for more details) that indicates provisioning is complete. This is handled in the application code through the `sl_btmesh_on_node_provisioned()`callback.

4.2 Sensor Thermometer Example

This section describes basic operation of the **Bluetooth Mesh – SoC Sensor Thermometer**. It is assumed that the node is already provisioned and publish-subscribe settings have been configured by the smartphone app.

The sensor server may support multiple types of sensors present on the development board, such as a People Count sensor, or the Present Ambient Temperature sensor. The People Count sensor can be simulated by the buttons on the starter kit: PB0 decreases the count and PB1 increases the count, or via cli implementation. The count is maintainedin a 16-bit unsigned integer. The Present Ambient Temperature sensor is a Silicon Labs Si7021. Temperature is reported in units of 0.5 degrees Celsius as a signed 8-bit integer.

Upon receiving the `sl_btmesh_evt_node_initialized_id` event and call to the corresponding callback `sl_btmesh_handle_sensor_server_events()`, the sensor server node initializes the sensors by calling `sl_btmesh_sensor_server_node_init()`. This occurs in the `btmesh_sensor_server` component (see `sl_btmesh_sensor_server.c`). This function sets the people count to 0 and initializes the temperature sensor hardware. Otherwise, the node starts unprovisioned beaconing and waits for a provisioner. Once provisioned and initialized, the sensor server node simply waits for messages from the client. Events generated by messages from the client are handled in the sl_btmesh_sensor_server.c.

Please note that sensor settings and cadence are not supported at this time so the message handlers are stubs.

- Get requests are handled by **handle_sensor_server_get_request().** The property_id is queried. If the value is non-zero and is a supported value then the sensor data for that property is returned. If the property_id is zero, then all supported sensor data is returned. If the value is non-zero and is an unsupported value, the data length is set to zero to indicate an unsupported property. In all cases, `sl_btmesh_sensor_server_send_status()` is called to send the status to the client.
- Get Series requests are handled by **handle_sensor_server_get_series_request()**. Neither sensor properties supported in this example include either Series State so only the property_id is sent back to the client .
- Get Column requests are handled by **handle_sensor_server_get_column_request().** Neither of the sensor properties include Column State so the same data is sent back to the client, per the specification.
- Publishing sensor data is handled by **handle_sensor_server_publish_event()** when the publish period expires. Data from both sensors is published.
- The Cadence and Settings States for the properties in this example are not included so their handlers simple return the property ID, per the specification.

4.3 Sensor Client Example

This section describes basic operation of the **Bluetooth Mesh – SoC Sensor Client.** It is assumed that the node is already provisioned and publish-subscribe settings have been configured by the smartphone app. The main purpose of the sensor client is to request sensor data from the sensor server. The sensor client supports two types of sensors: a people counting sensor and a temperature sensor.

Upon receiving the `sl_btmesh_evt_mesh_node_initialized_id` event, the sensor client node performs the following actions:

- Initializes the sensor client model by calling `sl_btmesh_sensor_client_init()`.
- Handles the buttons on the device through calling `app_button_press_cb()`.
- Requests a sensor descriptor by calling `sl_btmesh_sensor_client_get_descriptor()` with address 0x0000 to publish the message using the publish parameters set by the configuration client.
- Starts a timer to periodically request sensor data.

The device buttons are used as follows:

- PB0 to select the sensor property_id to interact with.
- PB1 to reset the list of registered devices.

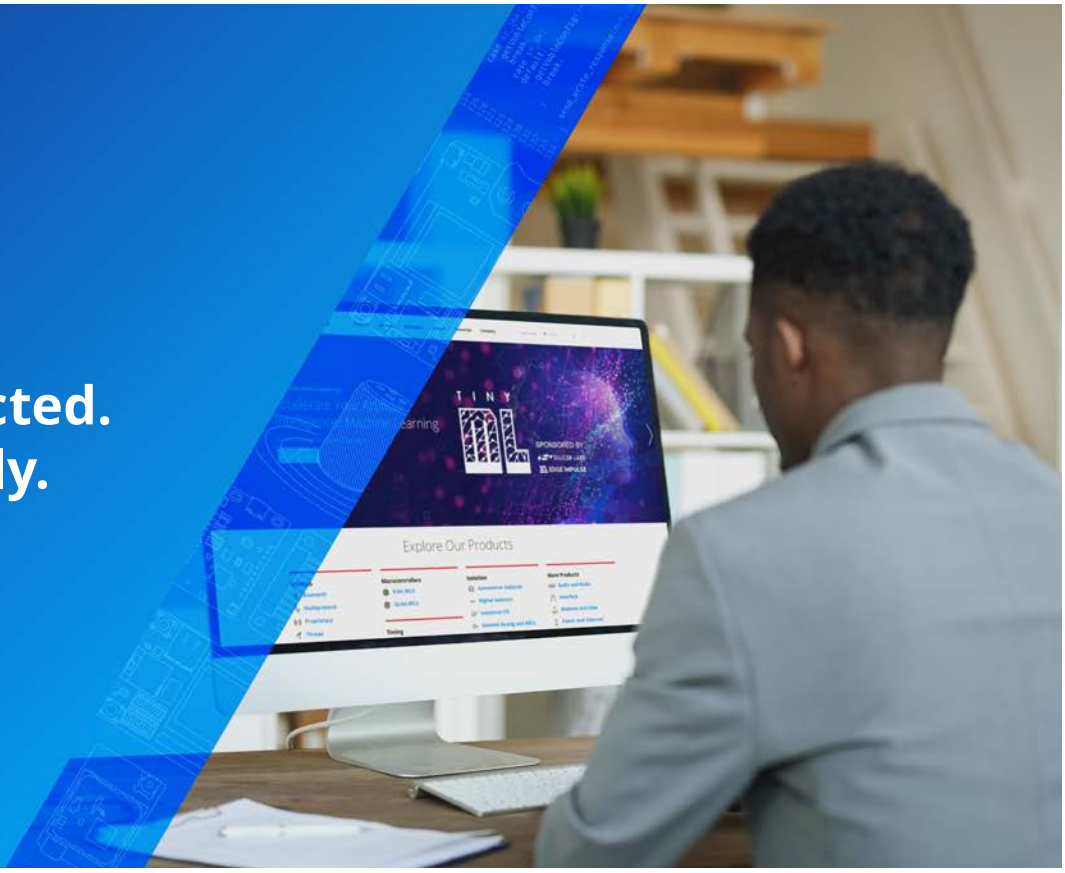
When PB0 is pressed, `sensor_client_change_current_property()` is called to handle the change. This function increments the variable current_property, which is used to index the properties array containing a list of the supported properties.

When PB1 is pressed `update_registered_devices()` is called to find and register the devices that support the currently selected property. This is done by calling `sl_btmesh_sensor_client_get_descriptor()` with the selected property as a parameter. A recurring timer is started with a 2000 ms interval. This timer is used to request the sensor data by calling `sl_btmesh_sensor_client_get_sensor_data()`.

When a message is received from a sensor server, one of the sensor events are triggered. These events are as follows:

Event	Description	Behavior
sl_btmesh_evt_sensor_client_descriptor_status_id	Indicates that a descriptor status message has been received.	Adds the server to the list of registered devices if it was not previously registered.
sl_btmesh_evt_sensor_client_status_id	Indicates that a sensor status message has been received.	Verifies that the data came from a registered device, determines which type of sensor the data comes from, then saves and displays it.

Smart. Connected. Energy-Friendly.



IoT Portfolio
www.silabs.com/products

Quality
www.silabs.com/quality

Support & Community
www.silabs.com/community

SILICON LABS

Silicon Laboratories Inc.
400 West Cesar Chavez
Austin, TX 78701
USA

www.silabs.com