RX23W Group

Temperature and Humidity Sensor Data Communication Sample Code Using Bluetooth Mesh Network

Introduction

This application note describes a Mesh temperature and humidity sensor data communication demo (Mesh Sensor Demo) that utilizes a Sensor Model included in the Bluetooth® Mesh module for the RX23W.

Mesh Network (MtoM communication) of the Bluetooth Low Energy communication makes it possible to send data over much farther distances than conventional PtoP communication, easily expand the network by adding nodes, and build the network that is resistant to communication failures.

This application note and the associated program were created using RX23W Group Bluetooth Mesh Stack Development Guide (R01AN4875) as a reference.

Operation Confirmation Devices

- RX23W Group
- Temperature and humidity sensor
  Renesas Electronics HS300x Relative Humidity and Temperature Sensor (HS300x sensor)

Operation Confirmation Boards

- Target Board for RX23W (TB-RX23W board)
- HS3001 Sensor Board*1: Relative Humidity Sensor Pmod™ Board (US082-HS3001EVZ)
- Converter board*1: Interposer Board for Pmod Type 2A/3A to 6A (US082-INTERPEVZ)

Note: 1. Contact a Renesas representative for information on obtaining the Interposer Board.
Even if the HS3001 Sensor Board is not connected, it is possible to confirm the communication functionality using the Sensor Model because pseudo data transmission is possible.

Related Documents

This application note refers to and explains the following documents. In addition, the last 6 digits of the document number are omitted. The chapter structure may change when the document is updated. Please be careful when referencing.

Document Title	Document No.
RX Family Renesas Sensor Control Modules Firmware Integration Technology	R01AN5892EJ0110
RX Family Renesas HS300x Sensor Control Module Firmware Integration Technology	R01AN5893EJ0110
RX Family Renesas Sensor I2C Communication Middleware Control Module Firmware Integration Technology	R01AN5895EJ0110
RX23W Group Bluetooth Mesh Stack Startup Guide	R01AN4874EJ0120
RX23W Group Bluetooth Mesh Stack Development Guide	R01AN4875EJ0120
RX23W Group Bluetooth Mesh Module Using Firmware Integration Technology	R01AN4930EJ0120
RX23W Group BLE Module Firmware Integration Technology	R01AN4860EJ0230

Contents

1. Overview

1.1 System Overview

All the nodes used for Mesh network employ the RX23W. A Bluetooth Mesh network is constructed from a Sensor Server node, a Sensor Client node, nodes with Relay feature, and a node with Friend feature, and temperature and humidity data from the HS300x sensor connected to the Sensor Server node is transmitted to the Sensor Client node. Requests to change the send interval for sensor data (default 2 seconds ↔ 5 seconds) can be sent from the Sensor Client node to the Sensor Server node.

In addition, the Sensor Server node has Low Power feature to reduce power consumption.

The Renesas Bluetooth Smartphone application, a smartphone app for configuring Mesh node settings, and be used for provisioning and configuration of all the nodes.

1.1.1 Mesh Network Configuration and Data Flows

(1) Network Configuration for Mesh Sensor Demo

Table 1-1 lists the nodes in the Mesh Sensor Demo, and Figure 1-1 shows the Mesh network configuration.

For the configuration of elements and models for each node, refer to 4.2, ‘Mesh Models’.

Table 1-1 Nodes

Node	Description
Sensor Client	Node that operates as a Sensor Client Model The following operations are supported: • Reception of sensor data • Transmission of sensor data send interval change request values
Sensor Server _Low Power	Node that operates as a Sensor Server Model and has the optional Low Power feature In the description that follows, this node is sometimes referred to as the Sensor Server node or Low Power node. The following operations are supported: • Acquisition and transmission of measurement data from the connected sensor device • Reception of sensor data send interval change request values and switching of the send interval
Friend_Relay	Node with optional Friend and Relay feature In the description that follows, this node is sometimes referred to as the Friend node.
Relay	Node with optional Relay feature

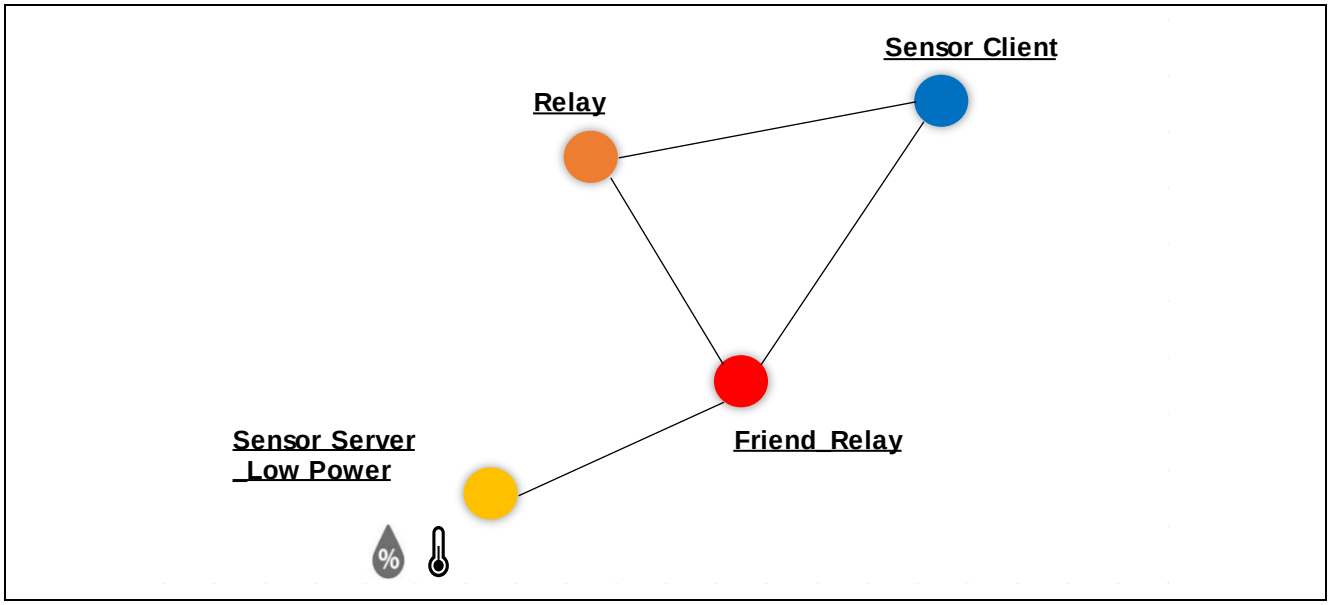


Figure 1-1 Network Configuration for Mesh Sensor Demo

(2) Flow of Messages (Sensor Data) from Sensor Server Node to Sensor Client Node

Figure 1-2 shows the flow of sensor data when the Sensor Client receives measurement data from the Sensor Server. Even if the Sensor Client node is outside direct radio range of the Sensor Server node, the Sensor Client node can receive the sensor data via nodes with Relay feature.

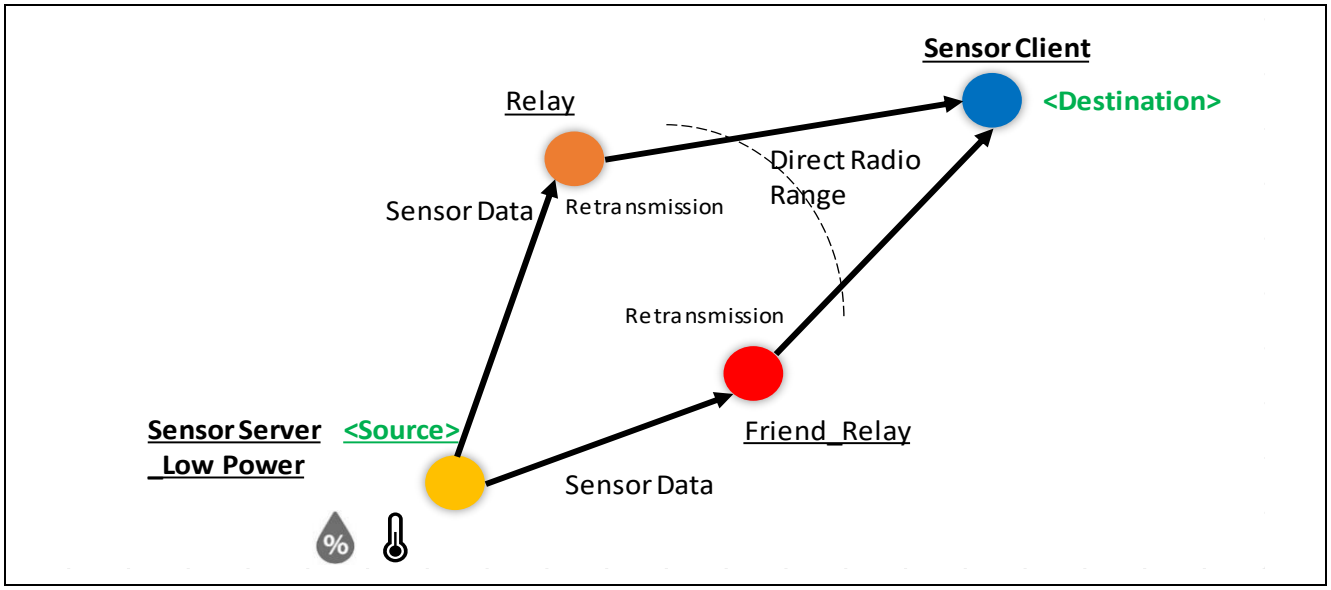


Figure 1-2 Data Flow when Sensor Client Node Is Outside Direct Radio Range of Sensor Server Node

(3) Flow of Messages (Sensor Data Send Interval Change Requests) from Sensor Client Node to Sensor Server Node

The Sensor Server (Sensor Server_Low Power) node with Low Power feature can run with Low Power feature by establishing a friendship relationship with the Friend node. For this reason, messages from the Sensor Client node are stored on the Friend node. The Sensor Server node can then receive the messages stored on the Friend node when scanning restarts.

Figure 1-3 shows the flow of messages (sensor data send interval change request values).

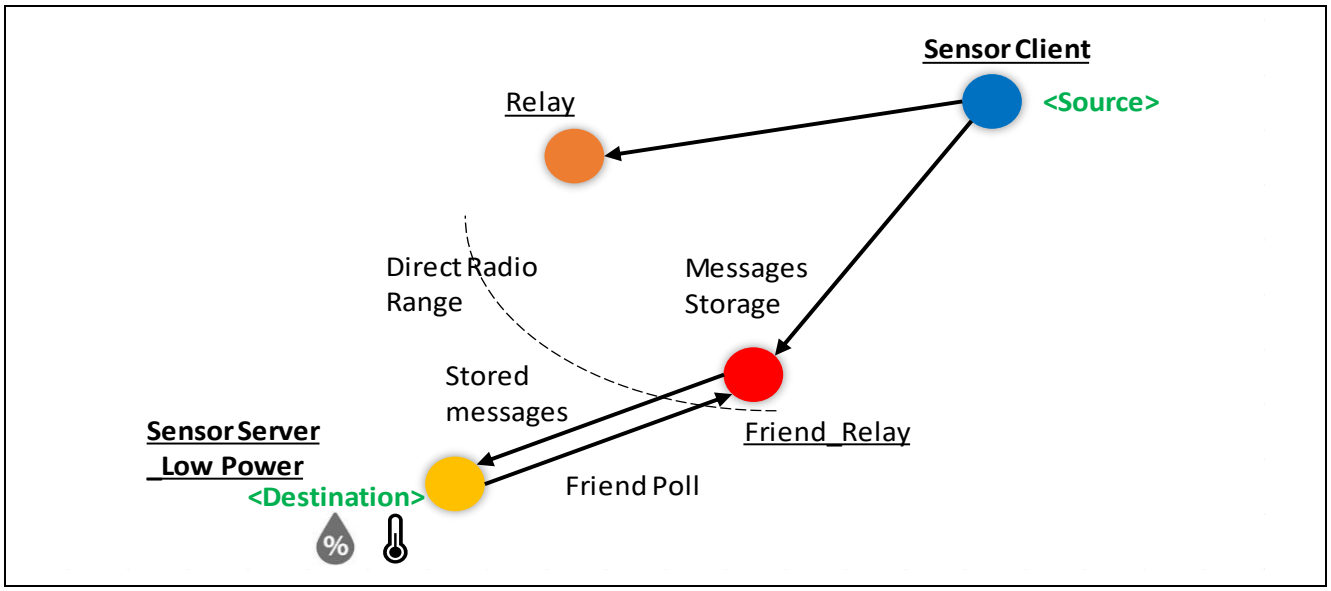


Figure 1-3 Flow of Messages from Sensor Client Node

1.1.2 MOT Files and e² studio Projects in Mesh Sensor Demo Package

An overview of the MOT files and e² studio projects in the package is presented below.

(1) MOT Files

The package contains the MOT files listed in Table 1-2.

After programming the MOT files to the TB-RX23W boards, you can run the demo immediately.

For the programming procedure, refer to 2.1, 'Programming the RX23W Using Renesas Flash Programmer'.

For the project settings used to create the MOT files, refer to 1.3, 'Operation Confirmation Environment'.

Table 1-2 MOT Files

MOT File Name	Description*1
tbrx23w_mesh_**sensor**_server.mot	Sensor Server model for TB-RX23W board
tbrx23w_mesh_**sensor**_client.mot	Sensor Client model for TB-RX23W board
tbrx23w_mesh_**sensor**_friend.mot*2	Friend_Relay node for TB-RX23W board Relay node for TB-RX23W board

Notes: 1. All files are stored in the **FITDemos\ROM_Files** folder.
2. The same MOT file is used for the Friend_Relay node and the Relay node.

(2) e² studio Projects

The package contains the Sensor Model projects listed in Table 1-3.

To run the demo, import each e² studio project and program it to the appropriate TB-RX23W board.

For the e² studio project settings, refer to 1.3, 'Operation Confirmation Environment'.

Table 1-3 Sensor Model Projects

Project Name	Description and Name of Storage Folder
tbrx23w_mesh_**sensor**_server	Sensor Server model project for TB-RX23W board Folder name: tbrx23w_mesh_**sensor**_server
tbrx23w_mesh_**sensor**_client	Sensor Client model project for TB-RX23W board Folder name: tbrx23w_mesh_**sensor**_client
tbrx23w_mesh_**sensor**_friend	Friend_Relay node and Relay node project for TB-RX23W board Folder name: tbrx23w_mesh_**sensor_friend**

Note: Each project uses BLE module Ver.2.30.

The above projects were created using the tbrx23w_mesh_server project and tbrx23w_mesh_client project contained in the 'Bluetooth Mesh Stack package' (R01AN4930) as a basis.

For the differences between the projects, refer to 4.6, 'Program Differences'.

1.2 Folder and File Structure

1.2.1 Folder Structure

The structure of the folders and files is shown below. Some folders and files have been omitted.

(1) r01an6129xx0101-rx23w-blemesh-sensor.zip

```
r01an6129xx0101-rx23w-blemesh-sensor
├─FITDemos
│    ├─ROM_Files
│    │   ├─tbrx23w_mesh_sensor_client.mot   MOT File for Sensor Client Node
│    │   ├─tbrx23w_mesh_sensor_friend.mot   MOT File for Friend_Relay Node
│    │   └─tbrx23w_mesh_sensor_server.mot   MOT File for Sensor Server Node
│    ├─tbrx23w_mesh_sensor_client.zip      Project Zip file for Sensor Client Node
│    ├─tbrx23w_mesh_sensor_friend.zip      Project Zip file for Friend_Relay Node
│    └─tbrx23w_mesh_sensor_server.zip      Project Zip file for Sensor Server Node
├─r01an6129ej0101-rx23w-blemesh-sensor
└─r01an6129jj0101-rx23w-blemesh-sensor
```

Figure 1-4 Package Folder and File Structure

(2) tbrx23w_mesh_sensor_client.zip

```
tbrx23w_mesh_sensor_client
├─.cproject
├─.project
├─.settings
├─src
│   ├─main.c
│   ├─mesh_appl.h
│   ├─mesh_core.c
│   ├─mesh_model.c
│   ├─smc_gen
│   │   ├─general
│   │   ├─r_ble_rx23w
│   │   ├─r_bsp
│   │   ├─r_byteq
│   │   ├─r_cmt_rx
│   │   ├─r_config
│   │   ├─r_flash_rx
│   │   ├─r_gpio_rx
│   │   ├─r_irq_rx
│   │   ├─r_lpc_rx
│   │   ├─r_mesh_rx23w
│   │   ├─r_pincfg
│   │   └─r_sci_rx
│   └─vendor_model
├─tbrx23w_mesh_client HardwareDebug.launch
├─tbrx23w_mesh_client.rcpc
└─tbrx23w_mesh_client.scfg
```

Figure 1-5 Folder and File Structure of tbrx23w_mesh_sensor_client.zip

(3) tbrx23w_mesh_sensor_friend.zip

```
tbrx23w_mesh_sensor_friend
├─.cproject
├─.project
├─.settings
├─src
│   ├─main.c
│   ├─mesh_appl.h
│   ├─mesh_core.c
│   ├─mesh_model.c
│   ├─smc_gen
│   │   ├─general
│   │   ├─r_ble_rx23w
│   │   ├─r_bsp
│   │   ├─r_byteq
│   │   ├─r_cmt_rx
│   │   ├─r_config
│   │   ├─r_flash_rx
│   │   ├─r_gpio_rx
│   │   ├─r_irq_rx
│   │   ├─r_lpc_rx
│   │   ├─r_mesh_rx23w
│   │   ├─r_pincfg
│   │   └─r_sci_rx
│   └─vendor_model
│       ├─vendor_api.h
│       ├─vendor_client.c
│       └─vendor_server.c
├─tbrx23w_mesh_client HardwareDebug.launch
├─tbrx23w_mesh_client.rcpc
└─tbrx23w_mesh_client.scfg
```

Figure 1-6 Folder and File Structure of tbrx23w_mesh_sensor_friend.zip

(4) tbrx23w_mesh_sensor_server.zip

```
tbrx23w_mesh_sensor_server
├─.cproject
├─.project
├─.settings
├─src
│   ├─hs300x
│   │   ├─RX_HS300X.c
│   │   └─RX_HS300X.h
│   ├─main.c
│   ├─mesh_appl.h
│   ├─mesh_core.c
│   ├─mesh_model.c
│   ├─smc_gen
│   │   ├─general
│   │   ├─r_ble_rx23w
│   │   ├─r_bsp
│   │   ├─r_byteq
│   │   ├─r_cmt_rx
│   │   ├─r_comms_i2c_rx  : I2C COMM Module
│   │   ├─r_config
│   │   ├─r_flash_rx
│   │   ├─r_gpio_rx
│   │   ├─r_hs300x_rx     : HS300x Module
│   │   ├─r_irq_rx
│   │   ├─r_lpc_rx
│   │   ├─r_mesh_rx23w
│   │   ├─r_pincfg
│   │   ├─r_riic_rx       : RIIC Module (Not used)
│   │   ├─r_sci_iic_rx    : SCI Simple I2C Mode
│   │   └─r_sci_rx
│   └─vendor_model
│       ├─vendor_api.h
│       ├─vendor_client.c
│       └─vendor_server.c
├─tbrx23w_mesh_sensor_server.x.launch
├─tbrx23w_mesh_server HardwareDebug.launch
├─tbrx23w_mesh_server.rcpc
└─tbrx23w_mesh_server.scfg
```

Figure 1-7 Folder and File Structure of tbrx23w_mesh_sensor_server.zip

1.2.2 Principal FIT Modules Used by Mesh Sensor Demo Project

Figure 1-8 shows the component settings of the Sensor Client project and Sensor Server project.

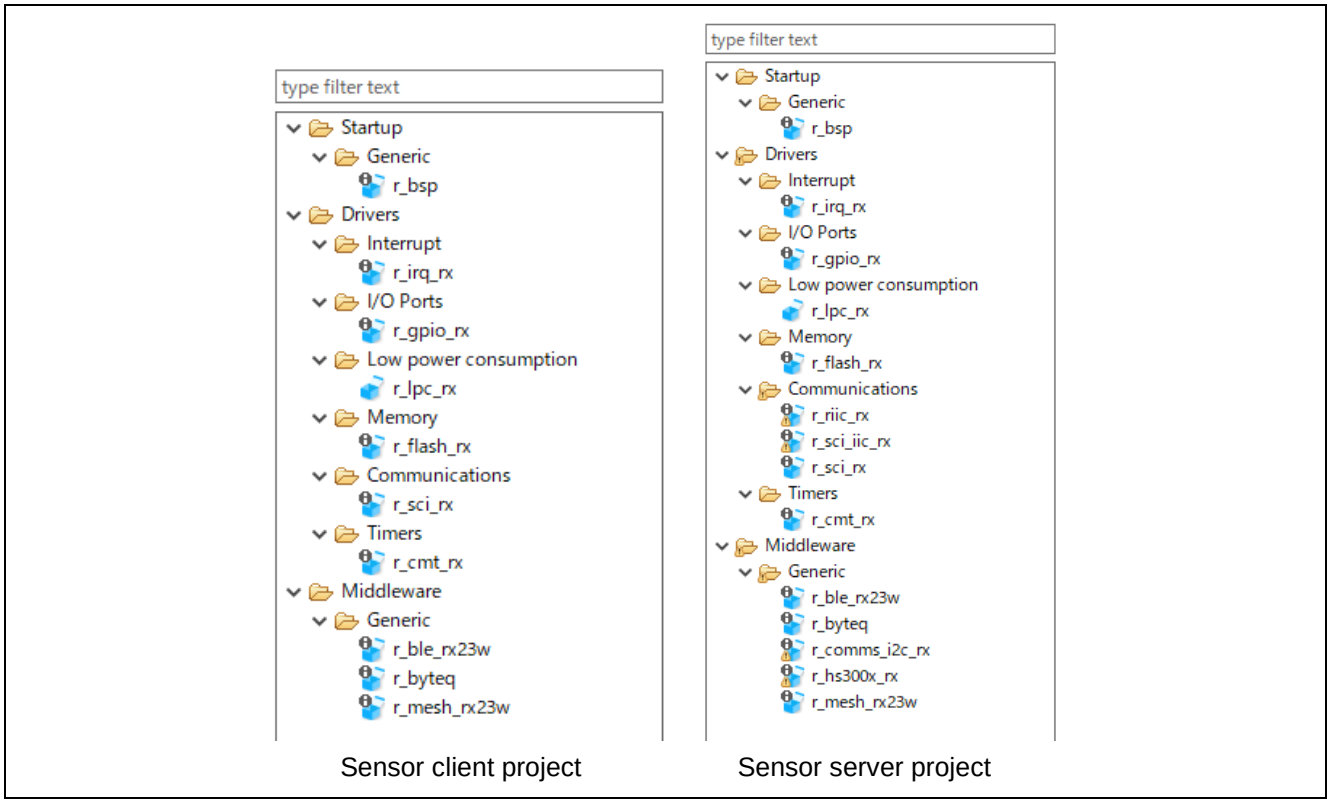


Figure 1-8 Project Component Settings

The principal FIT modules used are described below in Table 1-4.

For information on the other FIT modules (e.g., the BSP FIT module) required when using the FIT modules below, refer to the documentation of each FIT module.

For the configuration of the software layer of the FIT modules, refer to 4.1, ‘Software Configuration’. Also refer to 1.3, ‘Operation Confirmation Environment’.

Table 1-4 Principal FIT Modules used

Module Name	Module Ver.	Description
Bluetooth Mesh Module (Mesh FIT Module)	Ver.1.20	This module includes the Bluetooth Mesh stack layer and Bluetooth bearer wrapper layer. The Bluetooth Mesh stack package (R01AN4930xx0120) contains demo projects such as the tbrx23w_mesh_server project and tbrx23w_mesh_client project, as well as the smartphone application.
BLE Module (BLE FIT Module)	Ver.2.30	This module includes the Bluetooth Low Energy protocol stack layer.
HS300x Module	Ver.1.10	This module acquires data from an HS300x sensor connected to the I²C bus. This module controls the I²C module via the I²C COMMS module.
I2C COMMS Module	Ver.1.10	This module connects the HS300x module and the I²C module.
I2C Module	Ver.2.49	This module controls the I²C bus. It is intended for use with the RIIC module or SCI Simple I²C Mode module. The Mesh Sensor Demo uses the SCI Simple I2C Mode module. The RIIC module is included, but it is not used due to issues regarding the connector for connecting the sensor.

1.2.3 Smartphone App ‘Renesas Bluetooth Mesh Mobile’

The ‘Renesas Bluetooth Mesh Mobile’ (Mesh Mobile) included in the Bluetooth Mesh stack package (R01AN4930) is used with the demo.

Also refer to 1.3, ‘Operation Confirmation Environment’.

Mesh Mobile is a sample application provided for use with the demo. It can be used to perform provisioning and configuration of the nodes.

1.3 Operation Confirmation Environment

(1) Operation Confirmation Conditions

The operation of the Mesh Sensor Demo has been confirmed under the conditions listed below.

Table 1-5 Operation Confirmation Conditions

Item	Description
MCU	RX23W
MCU board	Target Board for RX23W (TB-RX23W board)
Sensor	HS3001
Sensor board	Relative Humidity Sensor Pmod Board (US082-HS3001EVZ)
Interface converter board	Interposer Board for Pmod Type 2A/3A to 6A (US082-INTERPEVZ) Converter board for connecting the PMOD connector on the MCU board to the sensor board.
IDE	e^2 studio 2021-07 or later
Toolchain	CC-RX V2.08.01
Firmware concatenation tool	Renesas Flash Programmer V3.08.03 or later
Smartphone	Handset running Android™ OS 8 or later
Mesh Mobile application	Renesas Bluetooth Mesh Mobile Included in Bluetooth Mesh stack package (R01AN4930). It has been confirmed to work with the one included in R01AN4930xx0120.

(2) Node Function Settings

The demo uses the four types of nodes shown in Figure 1-1, ‘Network Configuration for Mesh Sensor Demo’.

Table 1-6 lists the feature settings of each node.

Refer to 3.2, ‘Hardware Environment’, for information on implementing the demo with a small number of boards.

Table 1-6 Optional Feature Settings of Nodes

Node	Features
Sensor Client Node	Sensor Client feature
Sensor Server_Low Power Node	Sensor Server feature Low Power feature enabled
Friend_Relay Node*1	Friend feature and Relay feature enabled
Relay Node*1	Relay feature enabled

Notes: Nodes other than those listed above require features. Also refer to 4.2, ‘Mesh Models’.

1. These nodes can run as a Friend_Relay node or Relay node by means of the configuration described below.

1.4 Code Size

Figure 1-7 shows the code size of each node.

Table 1-7 Code Size Lists

Project	Total	Subtotal of BLE Section and MESH Section
tbrx23w_mesh_**sensor**_server	ROM: 315KB RAM: 45KB	ROM：227KB RAM：21KB
tbrx23w_mesh_**sensor**_client	ROM: 298KB RAM: 44KB	ROM：227KB RAM：21KB

2. Firmware Programming

The procedure for programming the RX23W using the Renesas Flash Programmer is described below.

2.1 Programming the RX23W Using Renesas Flash Programmer

You can use the Renesas Flash Programmer to program a MOT file to the RX23W on the board. Refer to Table 1-2, 'MOT Files', for a list of the bundled MOT files.

Obtain Renesas Flash Programmer V3.08.03 or later from the link below.

https://www.renesas.com/software-tool/renesas-flash-programmer-programming-gui

The programming procedure is described below.

(1) ESW1 Switch Setting when Programming TB-RX23W Board

Make the following setting.

- Firmware programming and debugging: Set the ESW1 2-4 to ON.

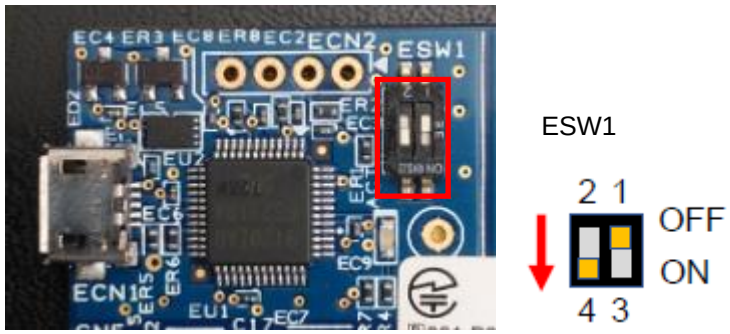


Figure 2-1 ESW1 Switch Setting when Programming TB-RX23W Board

(2) Connecting the PC to the TB-RX23W Board

Connect the PC to the emulator connector (ECN1) with a USB cable as shown in Figure 2-2.

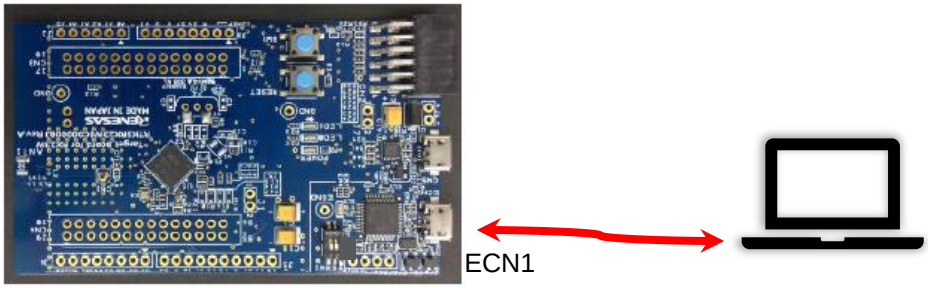


Figure 2-2 Connecting the PC to the TB-RX23W Board

(3) Renesas Flash Programmer startup and connection processing

Launch the Renesas Flash Programmer.

The procedure differs depending on whether or not a Renesas Flash Programmer project has already been created. The following two versions of the procedure are described here.

a. Procedure when Renesas Flash Programmer project has not yet been created
b. Procedure when Renesas Flash Programmer project has already been created

(a) Procedure when Renesas Flash Programmer project has not yet been created

1. Creating a New Project
   From the **File** menu, select **Create New Project…**.
   Make the following settings.
   - **Project Information** → **Microcontroller:** RX200
   - **Communication** → **Tool: E2 emulator Lite** (Specifies the onboard emulator on the TB-RX23W board.)
   - **Communication** → **Interface**: **FINE**

   Confirm that **Power: None** is displayed under **Communication**.

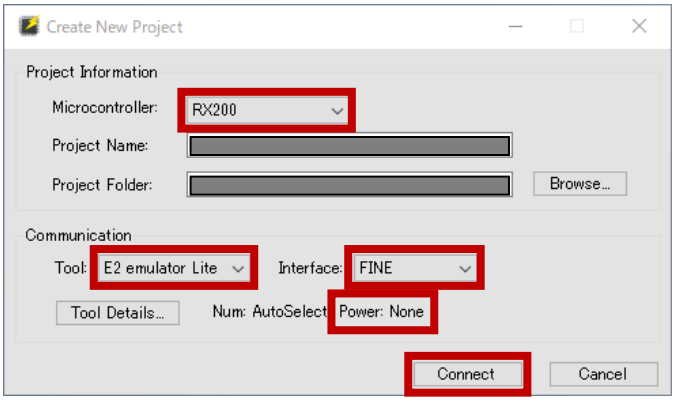


Figure 2-3 Project Information and Communication Settings

2. Connection Processing
   Click the **Connect** button shown in Figure 2-3. The Renesas Flash Programmer starts connection processing.
   Confirm that the following is displayed in the log output window.

```
Disconnecting the tool
Operation completed.
```

Figure 2-4 Display in Log Output Window upon Successful Connection

(b) Procedure when Renesas Flash Programmer project has already been created

1. Opening the Project
   From the **File** menu, select **Open Project…** and select the project file.
   Select the **Connect Settings** tab and confirm the following settings under **Communication**.
   - **Tool: E2 emulator Lite** (Specifies the onboard emulator on the TB-RX23W board.)
   - **Interface: FINE**
   - **Power: None**

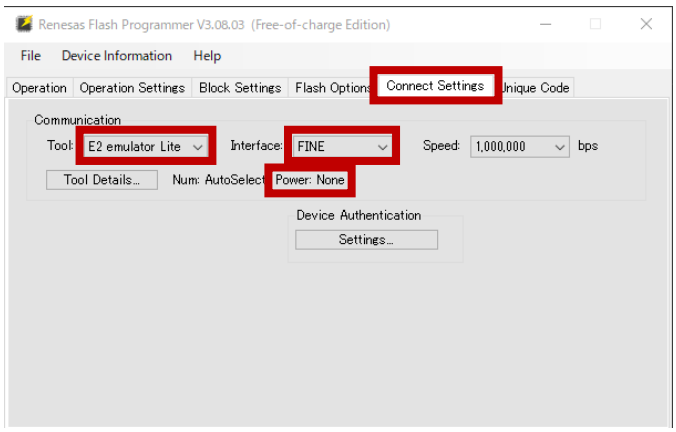


Figure 2-5 Settings on Connect Settings Tab

(4) Programming to the TB-RX23W Board

Follow the operation procedure of the Renesas Flash Programmer to program the firmware to the TB-RX23W board.

(5) ESW1 Switch Setting after Programming to TB-RX23W Board

Make the following setting to run the demo.

- To execute firmware without using the onboard emulator: Set the ESW1 2-4 to OFF.

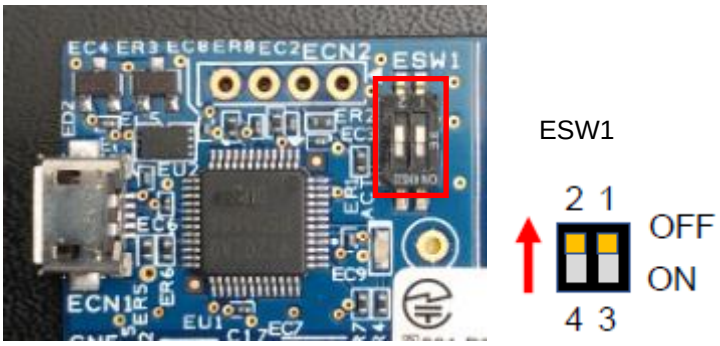


Figure 2-6 ESW1 Switch Setting after Programming to TB-RX23W Board

3. Demo

The procedure for running the Mesh Sensor Demo is described below.

3.1 Demo Overview

The nodes and network configuration shown in Figure 3-1, 'Overall Configuration of Demo' can be used to demonstrate sensor data communication from the Sensor Server node to the Sensor Client node and data send interval change request communication from the Sensor Client node to the Sensor Server node, as summarized in Table 3-1.

In addition, node state changes are output to logs via serial communication.

Table 3-1 Overview Description of Demo Operations

Node	Overview of Demo Operations
Sensor Server _Low Power	• Acquisition and transmission of sensor data, and establishing of Friendship with Friend node: Refer to 3.4, 'Sensor Data Communication Demo'. • Reception of sensor data send interval change requests and interval change processing: Refer to 3.5, 'Sensor Data Send Interval Change Demo'.
Sensor Client	• Reception of sensor data: Refer to 3.4, 'Sensor Data Communication Demo'. • Sensor data send interval change requests: Refer to 3.5, 'Sensor Data Send Interval Change Demo'.
Friend_Relay	• Low Power node operation change confirmation depending on the Friend node presence/absence: Refer to 3.6, 'Operation Change Based on Friend Node Presence/Absence Demo'. • Mesh network communication change confirmation depending on the Friend node presence/absence*1.
Relay	• Mesh network communication change confirmation depending on the Relay presence/absence*1.

Notes: It does not transition all nodes to low power software standby mode of the RX MCU's Low power consumption.

1. If all nodes are placed in close proximity, it is not possible to confirm the Mesh network changes depending on the presence or absence of the nodes.

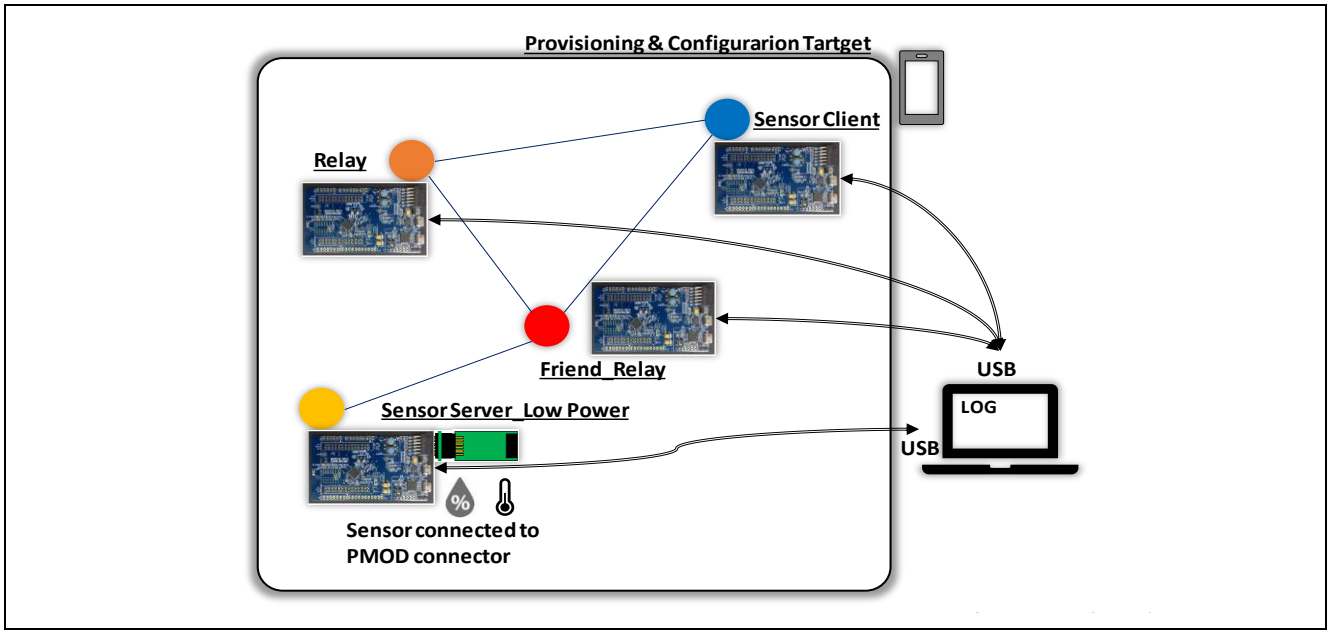


Figure 3-1 Overall Configuration of Demo

3.2 Hardware Environment

The hardware configuration of the demo is described below.

(1) TB-RX23W Board

TB-RX23W boards are used for the four types of nodes shown in Figure 3-1, 'Overall Configuration of Demo'.

Refer to section 2, 'Firmware Programming', and program the appropriate firmware to each board beforehand.

If an insufficient number of boards are available, you can still confirm the Sensor Model feature using a minimum of two boards.

Table 3-2 TB-RX23W Board Configuration

Node	Number of Boards	Remarks
Sensor Client	1 (required)	—
Sensor Server _Low Power	1 (required)	—
Friend_Relay	1 (optional) Priority: High	The Low Power feature of the Sensor Server_Low Power node will not work if this node is not present. If neither this node nor the Relay node is present, place the Sensor Client node and Sensor Server_Low Power node near enough to each other to allow direct one-to-one communication between them.
Relay	1 (optional) Priority: Low	If this node is not present, either place the Sensor Client node and Sensor Server_Low Power node near enough to each other to allow direct one-to-one communication between them, or place another node with Relay feature between them at the distance where communication is possible.

(2) Confirming ESW1 Switch Setting on TB-RX23W Board

Make the following setting to run the demo.

- To execute firmware without using the onboard emulator: Set ESW1 2-4 to OFF.

The demo will not operate properly if the ESW1 switch setting is incorrect.

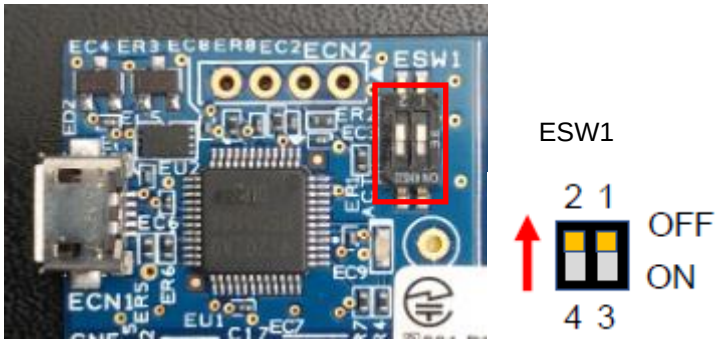


Figure 3-2 ESW1 Switch Setting on TB-RX23W Board for Demo

(3) Connection between TB-RX23W Board for Sensor Server_Low Power Node and HS3001 Sensor Board

Use a multi-stage connection to connect the HS3001 Sensor Board (US082-HS3001EVZ) to the PMOD connector (CN2) via the converter board (US082-INTERPEVZ), as shown in Figure 3-3.

Confirm the mark indicating pin 1 on the converter board (US082-INTERPEVZ) and make sure to plug it into the PMOD connector (CN2) correctly.

Short both pairs of jumper pins (J4 and J5) on the HS3001 Sensor Board (US082-HS3001EVZ) to enable pull-up processing of I^2C bus signals.

Note that if no HS3001 Sensor Board is present, pseudo data is generated in place of actual sensor data.

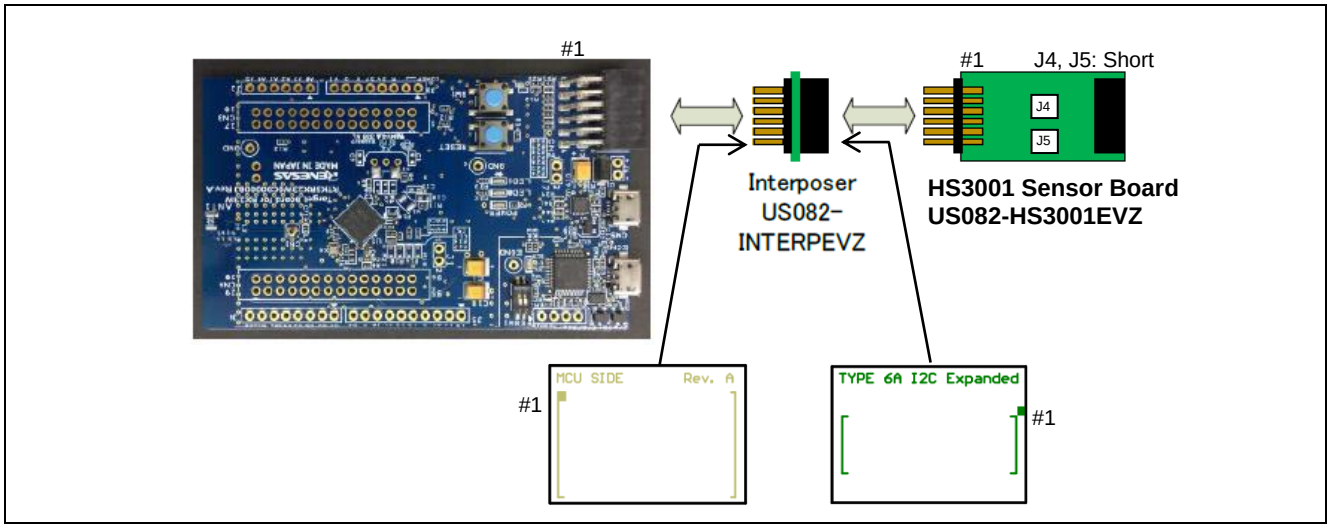


Figure 3-3 Connection between TB-RX23W Board for Sensor Server_Low Power Node and HS3001 Sensor Board

(4) Connecting the Serial Cable for Log Output to the PC and TB-RX23W Board

Each TB-RX23W board can produce log output.

Use a USB cable to connect the PC to the USB serial converter connector (CN5) as shown in Figure 3-4.

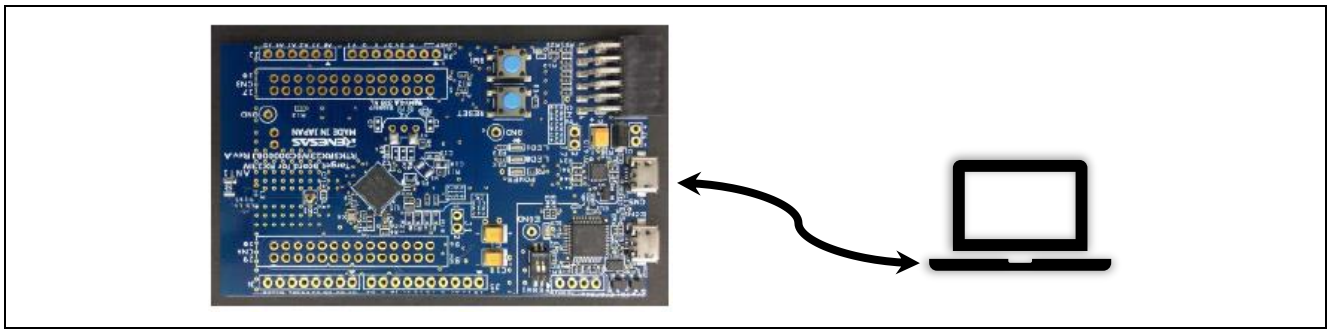


Figure 3-4 Serial Cable Connection between PC and TB-RX23W Board

(5) Smartphone

A smartphone is used for provisioning and configuration of each node. Refer to 1.3, ‘Operation Confirmation Environment’.

3.3 Software Environment

First, refer to section 2, ‘Firmware Programming’, and program the firmware to the TB-RX23W boards.

3.3.1 Terminal Emulator Software Settings

Terminal emulator software (e.g., Tera Term) is required to generate log output using serial communication. Note that a serial communication channel (SCI channel 8 in the case of the TB-RX23W boards) must be enabled in order to implement log output.

For the setting procedure, refer to 5.4.2, ‘r_ble_rx23w’, and 5.4.3, ‘r_sci_rx’, in ‘RX23W Group: Bluetooth Mesh Module Using Firmware Integration Technology’ (R01AN4930).

(1) Serial Port Settings

Table 3-3 lists the serial port settings.

Table 3-3 Serial port settings

Item	Setting
Baud rate	115,200 bps
Data	8 bits
Parity	None
Stop	1 bit
Flow control	None

3.3.2 Installing Mesh Mobile

Mesh Mobile is a tool for provisioning and configuration of Mesh nodes. It is contained in the Bluetooth Mesh stack package (R01AN4930).

If you have not already done so, install Mesh Mobile on the smartphone. An example of the installation procedure is described below.

Step 1: Copy the package file (apk file) located in the following folder of the Bluetooth Mesh stack package (R01AN4930) from the PC to the smartphone via USB.

FITDemos\mesh_mobile\android-debug.apk

Step 2: Use a file manager application to run the apk file on the smartphone.

3.4 Sensor Data Communication Demo

Communication of sensor data is started by the following processing sequence. Also refer to section 7, 'Demonstration', in 'RX23W Group Bluetooth Mesh Stack Startup Guide' (R01AN4874).

1. Provisioning and configuration using Mesh Mobile
2. Establishment of friendship between Low Power (Sensor Server_Low Power with Low Power feature enabled) node and Friend (Friend_Relay) node
3. Sensor data transmission by Sensor Server_Low Power node
4. Sensor data reception by Sensor Client node

3.4.1 Provisioning and Configuration using Mesh Mobile

Supply power to all boards and launch Mesh Mobile.

Note: You will need to enable the following settings in order to use Mesh Mobile on an Android smartphone.

- Location information
- Storage

The setting procedure for provisioning and configuration is described below.

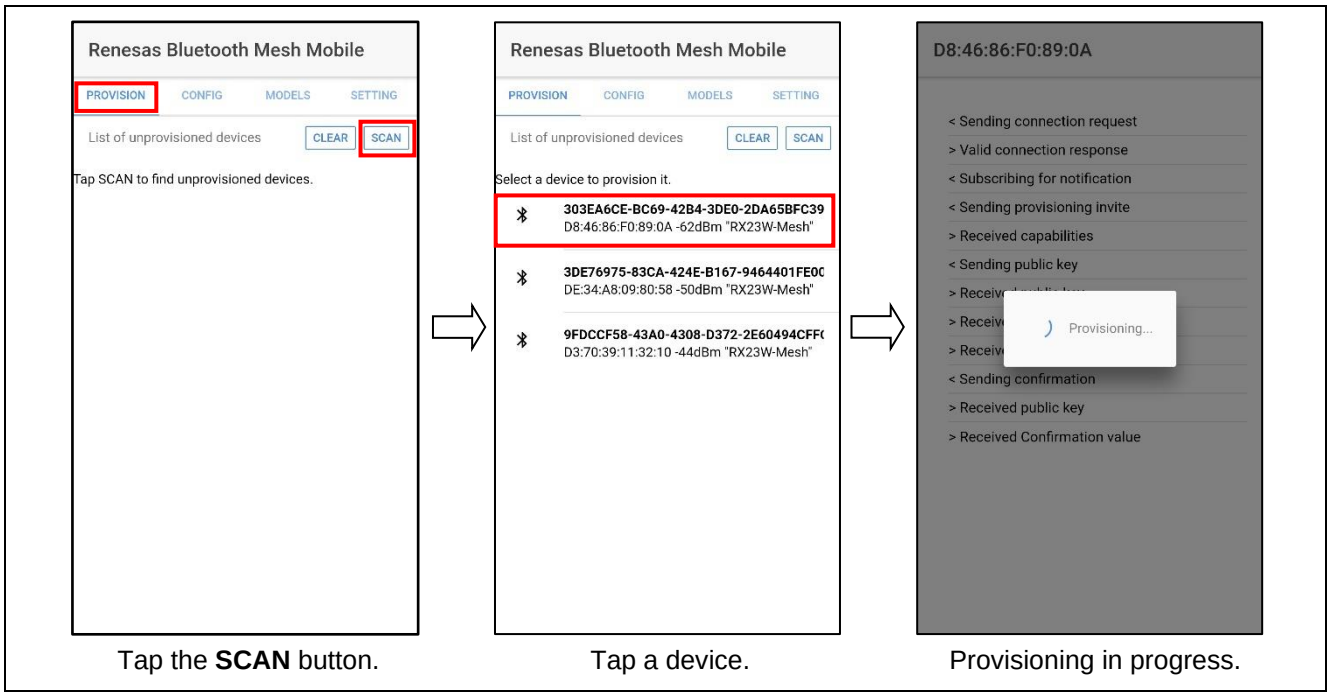
If the setting procedure does not proceed as indicated below, refer to section 5, 'Troubleshooting'.

(1) Provisioning

This step involves adding each device to the Mesh network and registering it as a node.

Carry out the following procedure for each board.

1. Select the **PROVISION** tab and tap the **SCAN** button to search for unprovisioned devices.
2. From the search results, select a device to perform provisioning.
3. After a connection is established, provisioning is executed.



Tap the **SCAN** button. Tap a device. Provisioning in progress.

Figure 3-5 Step 1 Provisioning Screens

(2) Configuration

This step configures the provisioned nodes.

Configuration activates each node and enables Mesh model communication.

The following setting are supported.

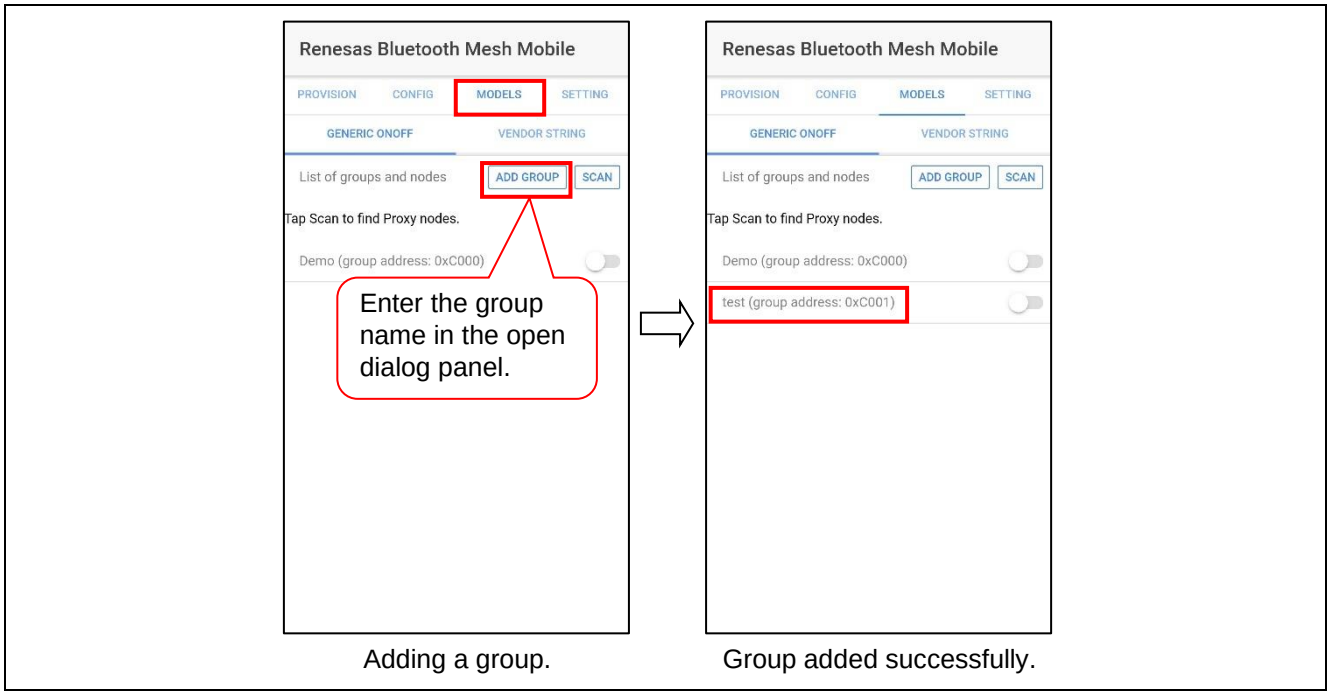
- Setting multiple nodes as a group
  You can add multiple nodes to a specified group that can then be manipulated.
  It is possible to create groups without depending on Mesh models.
- Configuring settings for optional features (firmware dependent) of nodes

(a) Adding a Group

Due to usage limitations, you should skip the group addition setting procedure described below. Later, we will make use of a 'Demo' group. Modifications are planned for the next Bluetooth Mesh stack package or later.

Carry out the following procedure to create a group.

1. Select the **MODELS** tab and tap the **ADD GROUP** button.
   Groups are not dependent on Mesh models, so you can add a group when either the **GENERIC ONOFF** tab or the **VENDOR STRING** tab is selected.
2. On the **ADD GROUP** dialog panel, enter a group name, such as "Kitchen."
3. Confirm that the group has been added.



Adding a group. Group added successfully.

Figure 3-6 Step 2 Configuration (Add Group) Screen

(b) Group Registration Settings and Node Settings for Each Board

Carry out the following procedure to register each board with a group.

Due to usage limitations, you should register the boards with the 'Demo' group.

1. Select the **CONFIG** tab and tap the **SCAN** button to search for nodes.
2. Nodes that are available for connection are displayed in green. Tap a node displayed in green to establish a connection and perform configuration.
3. After the configuration information is displayed, select the **CONFIGURATION** tab.

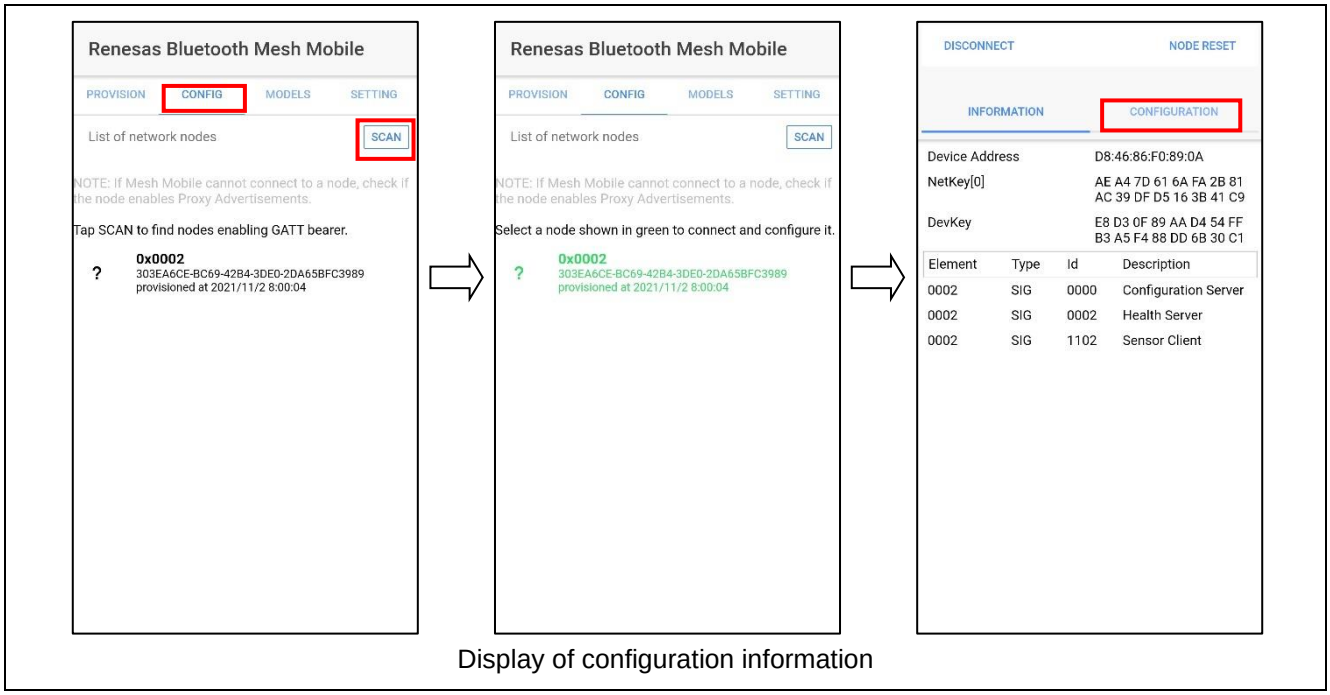


Figure 3-7 Step 2 Configuration (Group Registration and Node Settings) Screen (1/2)

4. Select the appropriate group and node settings on the **CONFIGURATION** tab.
5. Tap the **APPLY** button.

Table 3-4 lists the settings for each node used in the demo. Figure 3-8 shows screenshots of the settings.

Table 3-4 Nodes

Sensor Client Node	Sensor Server _Low Power Node	Friend_Relay Node	Relay Node
Select the **Group**.		• Select **Friend** and **Relay**. • The **Group** setting is not needed.	• Select **Relay**. • The **Group** setting is not needed.

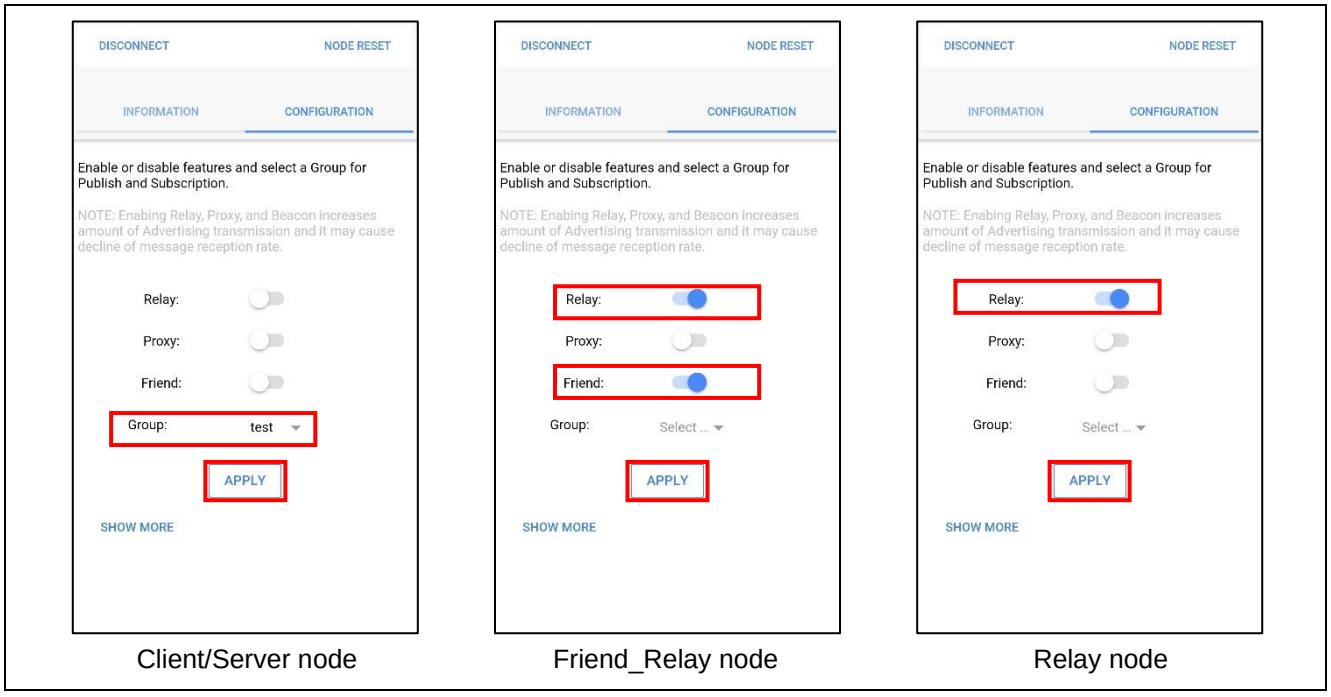


Figure 3-8 Step 2 Configuration (Group Registration and Node Settings) Screen (2/2)

6. After finishing configuration, tap the **DISCONNECT** button to go to the **CONFIG** tab.

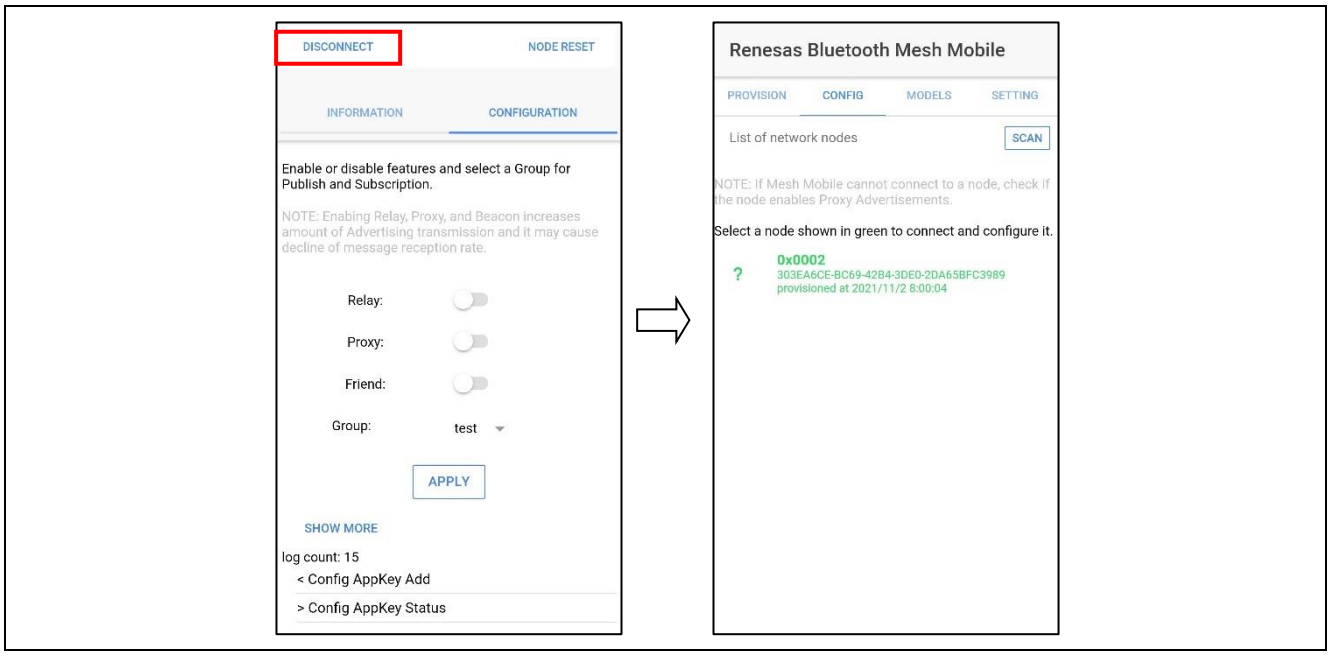


Figure 3-9 Step 2 Configuration Screen (DISCONNECT)

3.4.2 Establishment of Friendship between Low Power (Sensor Server_Low Power with Low Power Feature Enabled) Node and Friend (Friend_Relay) Node

When provisioning and configuration of the Low Power node are complete, processing to establish a friendship with the Friend node starts.
After the friendship is established, the Low Power node automatically suspends and resumes scanning and polls the Friend node. Also refer to Figure 4-6, 'Operation of Lower-Power Node and Friend Node'.

Figure 3-10 shows log output during processing by the Low Power node to establish a friendship.

```
[LPN] MS_trn_lpn_setup_friendship() status:0x0000
[LPN] MS_TRN_FRIEND_SETUP_CNF status:0x0000
[ACCESS] MS_access_cm_get_all_model_subscription_list() status:0x0000
…
[LPN] MS_trn_lpn_subscrn_list_add() status:0x0000
        0xC001
```

Figure 3-10 Log Output during Processing by Low Power Node when Establishing Friendship

The log sequence is described below.

(1) Low Power Node Log

Table 3-5 lists the items in the log of the Low Power node.

Also refer to 3.5.2, ‘Low Power Node’, and 3.5.3, ‘Low Power Node Sequence’, in ‘RX23W Group Bluetooth Mesh Stack Development Guide’ (R01AN4875).

Table 3-5 Contents of Low Power Node Log

Log Entry	Description
[LPN] MS_trn_lpn_setup_friendship()	Indicates that sending of a Friend Request to establish a friendship with a Friend node has started.
[LPN] MS_TRN_FRIEND_SETP_CNF	Friendship establishment completion event
[ACCESS] MS_access_cm_get_all_model _subscription_list()	Indicates acquisition of all subscription addresses of the Low Power node.
[LPN] MS_trn_lpn_subscrn_list_add()*1	Indicates addition of all subscription addresses of the Low Power node to the friend subscription list of the Friend node
[LPN] MS_TRN_FRIEND_SUBSCRNLIST _CNF	All subscription address registration events on the Friend Subscription List.

Note: 1. This enables the Friend node to store messages to a Low Power node.

(2) Friend Node Log

No log output is generated when a friendship is established.

3.4.3 Sensor Data Transmission by Sensor Server (Sensor Server_Low Power) Node

The Sensor Server node with Low Power feature enabled repeats 1 to 4 below periodicary.

1. Acquisition of sensor data produced by sensor operation
2. Transmission of sensor data
3. The Low Power feature restarts scan and queries the Friend node.
4. The Low Power feature suspends scan.

Figure 3-11 shows log output during transmission of sensor data by the Sensor Server node.

Note that the values shown in the figure are pseudo data values produced when no HS3001 sensor is connected. Figure 3-12 shows the log output when the Sensor Client node receives these values.

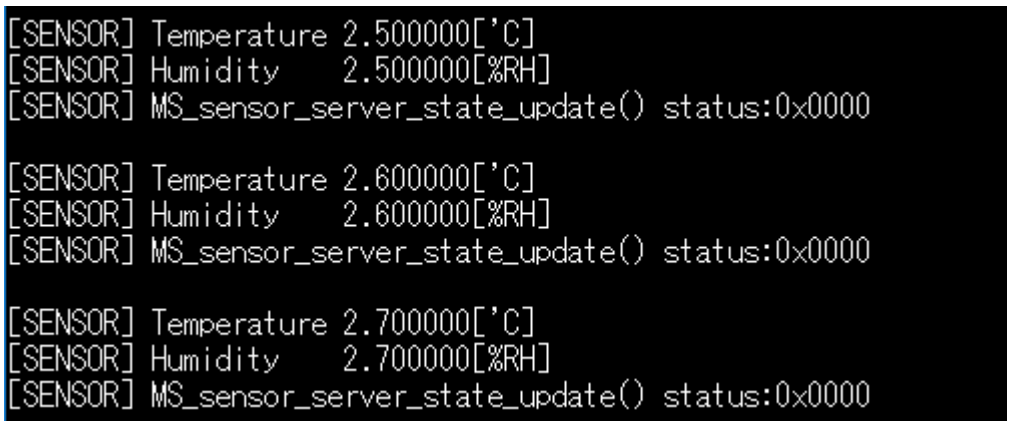


Figure 3-11 Log Output during Transmission of Sensor Data (Pseudo data) by Sensor Server Node

When a HS3001 sensor is connected, the measurement values obtained from the HS300x sensor are output.

(1) Sensor Server Node Log

Table 3-6 lists the items in the log of the Sensor Server node.

Table 3-6 Contents of Sensor Server Node Log

Log Entry	Description
[SENSOR] Temperature XX.XX['C]	Indicates the temperature [°C].
[SENSOR] Humidity XX.XX[%RH]	Indicates the relative humidity [%RH].
[SENSOR] MS_sensor_server_state_update()	Indicates transmission of measurement data.

3.4.4 Sensor Data Reception by Sensor Client Node

The Sensor Client node repeatedly receives sensor data.

Figure 3-12 shows log output during reception of sensor data by the Sensor Client node. In this example the transmitted data is that shown in Figure 3-11, ‘Log Output during Reception of Sensor Data (Pseudo data) by Sensor Client Node’.

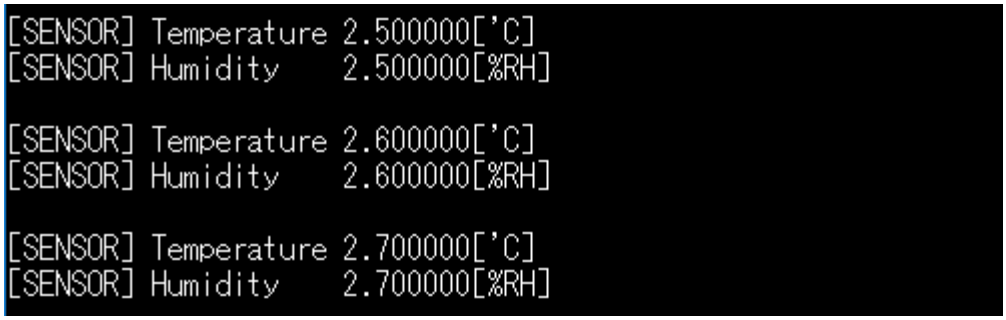


Figure 3-12 Log Output during Reception of Sensor Data (Pseudo data) by Sensor Client Node

When a HS3001 sensor is connected, the measurement values obtained from the HS300x sensor are output.

(1) Sensor Client Node Log

Table 3-7 lists the items in the log of the Sensor Client node.

Table 3-7 Contents of Sensor Client Node Log

Log Entry	Description
[SENSOR] Temperature XX.XX[‘C]	Indicates the temperature [°C].
[SENSOR] Humidity XX.XX[%RH]	Indicates the relative humidity [%RH].

3.5 Sensor Data Send Interval Change Demo

By pressing SW1 on the TB-RX23W board of the Sensor Client node, you can toggle the sensor data send interval between 2 seconds and 5 seconds (default*1).

Note: 1. For setting values, refer to 4.2.3(3), 'Changing the Sensor Data Send Interval'.

Figure 3-13 shows log output during changing of the sensor data send interval between the Sensor Client node and Sensor Server node.

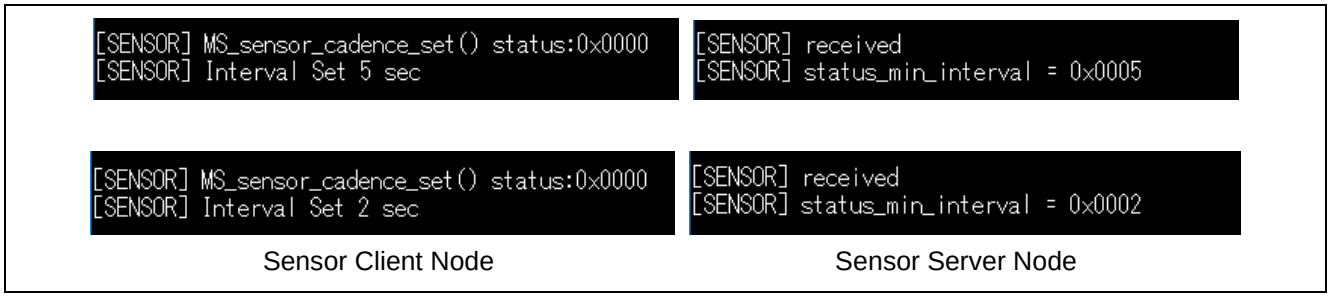


Figure 3-13 Log Output during Changing of Sensor Data Send Interval

(1) Sensor Client Node Operation Description

When SW1 on the board is pressed, a sensor data send interval change request is sent using a Sensor Cadence Set Unacknowledged message.

(2) Sensor Server Node Operation Description

When the sensor data send interval change request using a Sensor Cadence Set Unacknowledged message is received, the sensor data send interval is switched.

3.6 Operation Change Based on Friend Node Presence/Absence Demo

Figure 3-14 shows log output when, after a friendship has been established, powering off of the Friend node causes the friendship to be severed. After the friendship is severed, a friendship request is sent.

```
[LPN] MS_TRN_FRIEND_TERMINATE_IND status:0x0000
[LPN] MS_trn_lpn_setup_friendship() status:0x0000
```

Figure 3-14 Log Output of Low Power (Sensor Server_Low Power) Node when Friend Node Is Powered Off

Figure 3-15 shows log output when the Friend node is powered on again and the friendship is reestablished.

```
[ACCESS] MS_access_cm_get_all_model_subscription_list() status:0x0000
[LPN] MS_trn_lpn_subscrn_list_add() status:0x0000
        0xC001
[LPN] MS_TRN_FRIEND_SUBSCRNLIST_CNF status:0x0000
```

Figure 3-15 Log Output of Low Power (Sensor Server_Low Power) Node when Friend Node Is Powered On Again

Table 3-8 lists the items in the log of the Low Power node.

Also refer to 3.5.2, ‘Low Power Node’, and 3.5.3, ‘Low Power Node Sequence’, in ‘RX23W Group Bluetooth Mesh Stack Development Guide’ (R01AN4875).

Table 3-8 Contents of Low Power Node Log

Log	Description
[LPN] MS_TRN_FRIEND_TERMINATE_IND	A friendship severed event
[LPN] MS_trn_lpn_setup_friendship()	Indicates that sending of a Friend Request to establish a friendship with a Friend node has started.
[ACCESS] MS_access_cm_get_all_model _subscription_list()	Indicates acquisition of all subscription addresses of the Low Power node.
[LPN] MS_trn_lpn_subscrn_list_add()*1	Indicates addition of all subscription addresses of the Low Power node to the friend subscription list of the Friend node
[LPN] MS_TRN_FRIEND_SUBSCRNLIST _CNF	All subscription address registration events on the Friend Subscription List.

Note: 1. This enables the Friend node to store messages to a Low Power node.

3.7 Operation Changes Due to Presence/Absence of Relay Node

Operation when the Relay node is present or absent can be confirmed by powering off the Relay node and then powering it on again.

If all the nodes are placed close together, confirmation of changes in Mesh network operation due to the presence or absence of other nodes cannot be confirmed.

Therefore, you can confirm the operation of the Relay node by installing it outside the wireless range of the Sensor Client node and the Friend node as shown in Figure 3-16, and installing the Relay node between them.

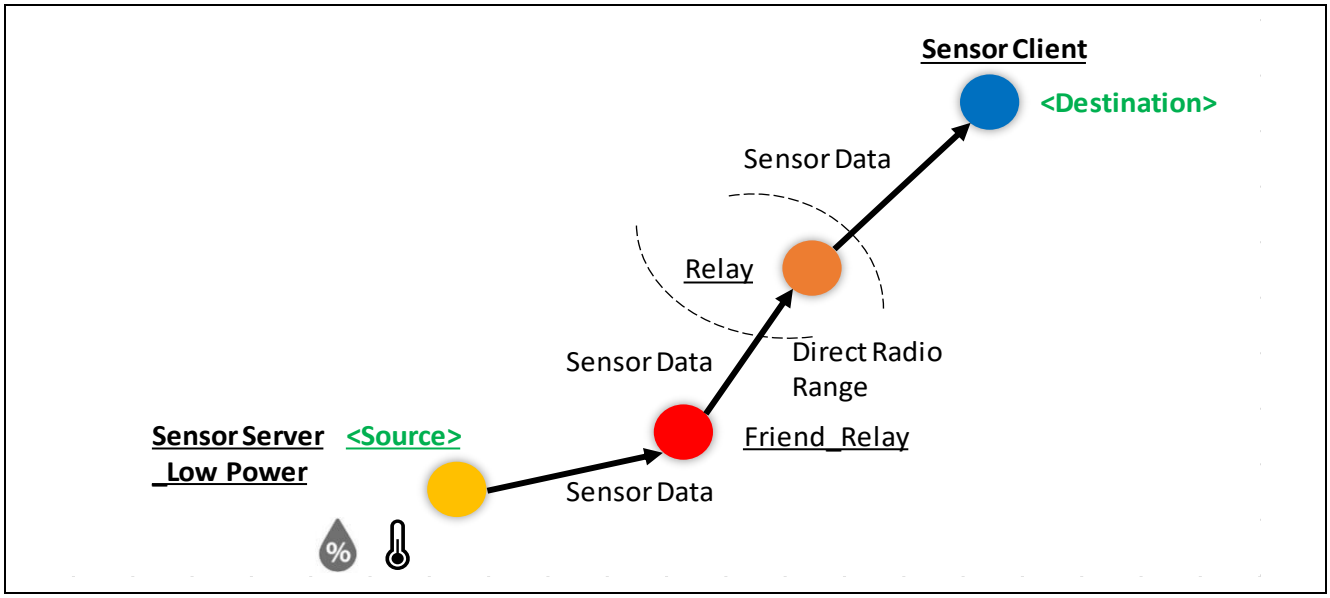


Figure 3-16 Data Flow when Relay Node is Installed between Sensor Client Node and Friend Node Outside Radio Range

4. Program Description

A program of the Mesh Sensor Demo software is presented below.

4.1 Software Configuration

4.1.1 Software Configuration of Sensor Server_Low Power Node

The software configuration of the Sensor Server is shown below.

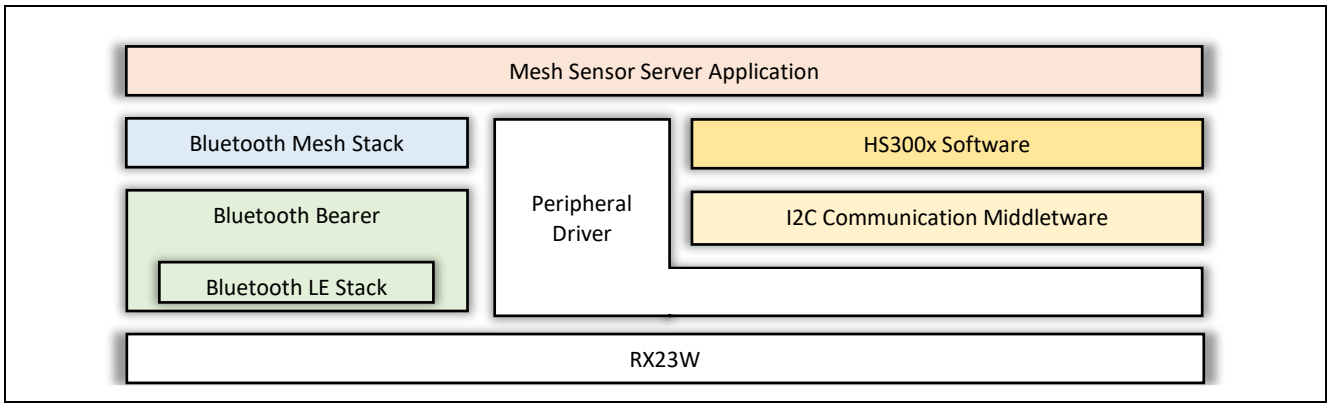


Figure 4-1 Software Configuration of Sensor Server_Low Power Node

- Mesh Sensor Server Application

Controls the Bluetooth Mesh stack and the HS300x module. Control of other sensors can be added easily.

For information on software other than the above, refer to the related documentation.

4.1.2 Software Configuration of Sensor Client Node

The software configuration of the Sensor Client is shown below.

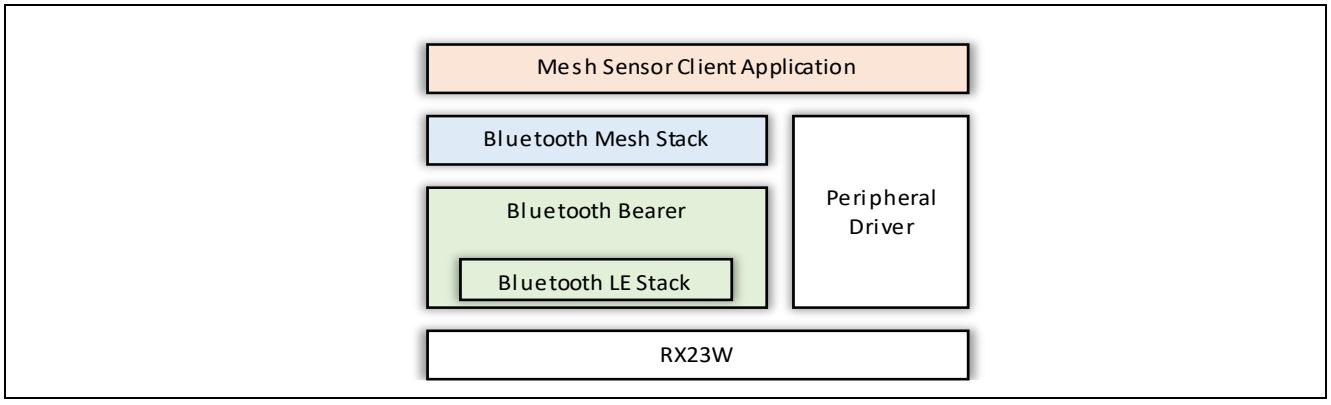


Figure 4-2 Software Configuration of Sensor Client Node

4.2 Mesh Models

The Mesh models used in the Mesh Sensor Demo are implemented in the file mesh_model.c.

The description in this section mainly concerns the Sensor Models. For information on the Configuration Server Model and Health Server Model, refer to 'RX23W Group Bluetooth Mesh Stack Development Guide' (R01AN4875).

4.2.1 Mesh Model Configuration of Each Node

Each node must have the Configuration Server/Health Server feature of the Foundation Models.

Optional Relay, Proxy, Friend, or Low Power feature can be added. Also refer to 4.3, 'Optional Features Node Settings'.

Figure 4-3 shows the Mesh model configuration of the nodes used in the Mesh Sensor Demo.

The Sensor Server node uses the Sensor Server Model of the Mesh models.

The Sensor Client node uses the Sensor Client Model of the Mesh models.

Since the Mesh Sensor Demo does not use the Relay node or Friend_Relay node for other purposes, it is not necessary to embed Mesh models other than the Configuration Server Model and Health Server Model.

Note that the same project is used for the Relay node and the Friend_Relay node. The features of each node are activated by configuration.

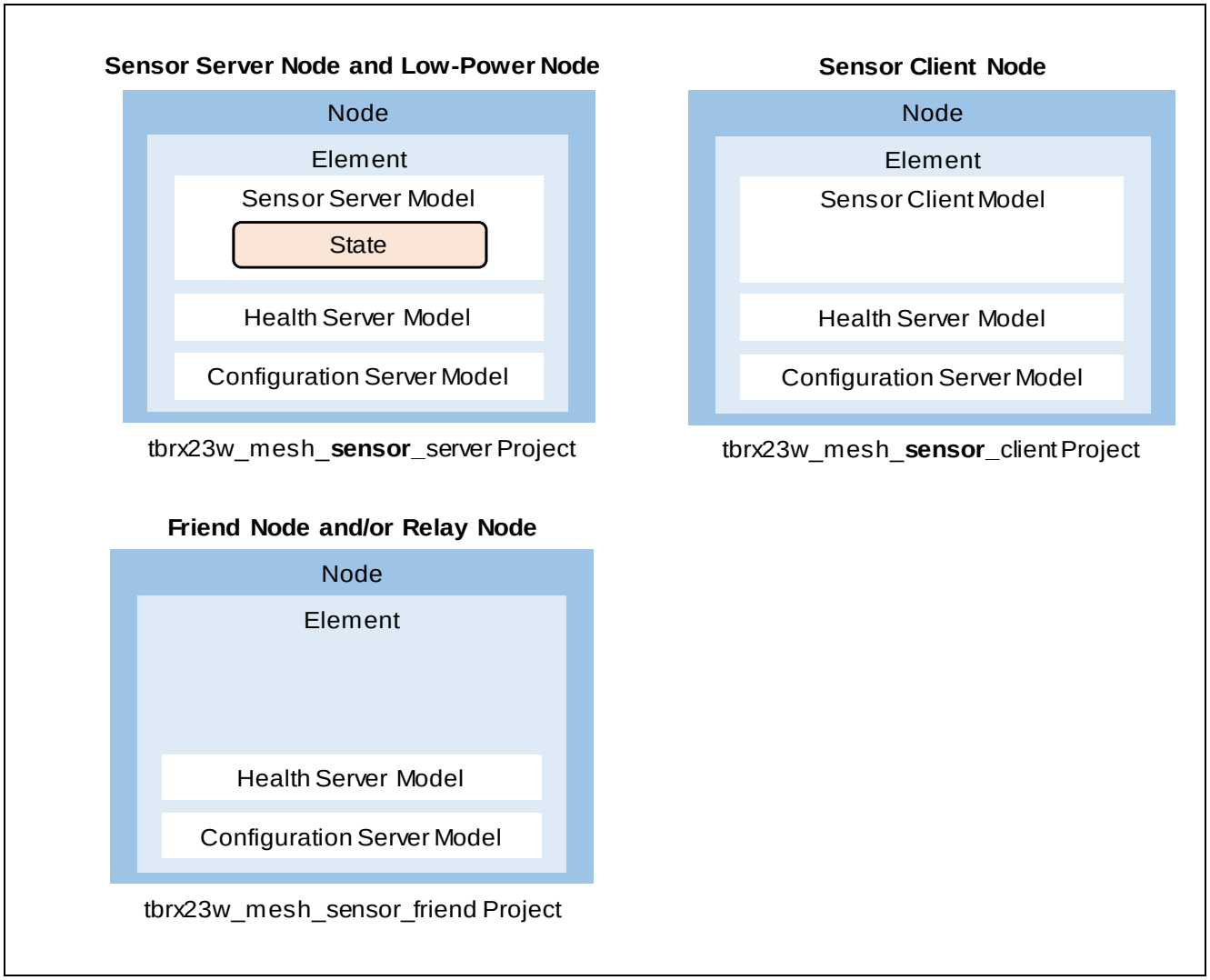


Figure 4-3 Mesh Model Configurations of Nodes Used in Mesh Sensor Demo

4.2.2 Sensor Model (Sensor Server/Sensor Client) Settings

Sensor Models from the Bluetooth Specification **Mesh Model** are used. Table 4-1 lists the settings.

Table 4-1 Sensor Model Settings

Model (Project Name)	Macro Definition for Build Options
Sensor Server Model (tbrx23w_mesh_sensor_server Project)	DEMO_SENSOR_SERVER_MODEL
Sensor Client Model (tbrx23w_mesh_sensor_client Project)	DEMO_SENSOR_CLIENT_MODEL

4.2.3 Sensor Models

The Sensor Models have four sensor states (Sensor Descriptor state, Sensor Setting and Sensor Cadence states, Sensor Data state, and Sensor Series Column state).

The Mesh Sensor Demo uses the Sensor Setting and Sensor Cadence states and the Sensor Data state.

(1) Sensor Server Node

The sample program for the Sensor Server node can perform the following operations.

- Measurement of temperature and humidity data from the HS300x sensor at the sensor data send interval and display of temperature and humidity.
- Transmission of sensor data to the Sensor Client node.
  The Sensor Data state is used as the transmission of sensor data using the Sensor Setting Set Unacknowledged message.
- Reception of sensor data send interval change requests from the Sensor Client node and updating of the sensor data send interval.
  The Sensor Setting and Sensor Cadence states are used as the sensor data send interval change request using the Sensor Cadence Set Unacknowledged message.
  The sensor data send interval is specified by the value of the Status Min Interval field of these states.

(a) Transmission of Sensor Data

The operation of the Sensor Models during transmission of sensor data is shown below.

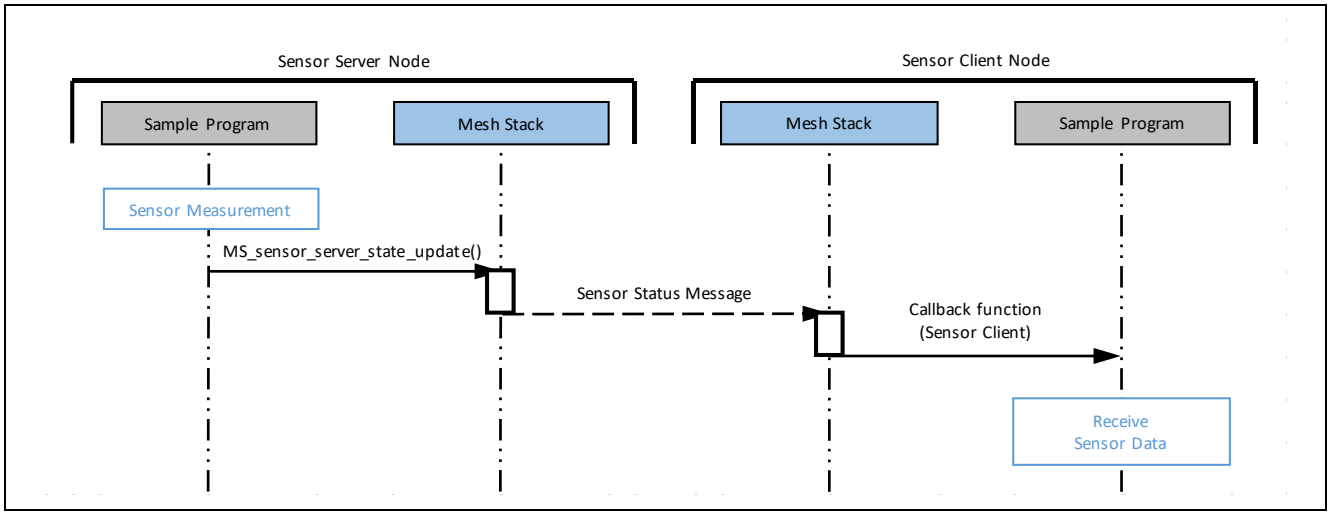


Figure 4-4 Sensor Model Operation during Transmission of Sensor Data

(b) Reception of Sensor Data Send Interval Change Requests

The operation of the Sensor Models during reception of a sensor data send interval change request is shown below.

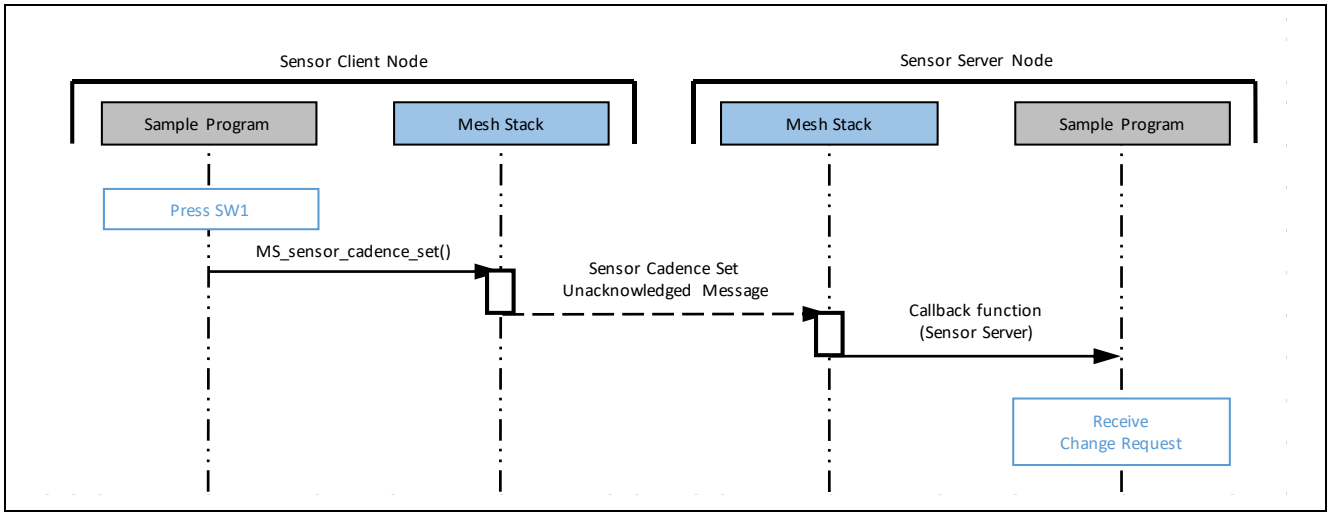


Figure 4-5 Sensor Model Operation during Reception of Sensor Data Send Interval Change Request

(2) Sensor Client Node

The sample program for the Sensor Client node can perform the following operations.

- Reception of sensor data from the Sensor Server node.
- Transmission of sensor data send interval change requests using the Sensor Cadence Set Unacknowledged message when SW1 on the Sensor Client node board is pressed.

(a) Reception of Sensor Data

For operation of the Sensor Models during reception of sensor data, refer to Figure 4-4, ‘Sensor Model Operation during Transmission of Sensor Data’.

(b) Transmission of Sensor Data Send Interval Change Requests

For operation of the Sensor Models during transmission of a sensor data send interval change request, refer to Figure 4-5, ‘Sensor Model Operation during Reception of Sensor Data Send Interval Change Request’.

(3) Changing the Sensor Data Send Interval

It is possible to define two send intervals for sensor data. Pressing SW1 on the Sensor Client node toggles between the two send intervals.

If you wish to change the interval settings, modify the following macros in the mesh_appl.h file of the Sensor Client project.

Table 4-2 Sensor Data Send Interval Setting Macros

Macro Name	Default Value (Unit: Seconds)
SENSOR_DATA_SEND_INTRERVAL01	2
SENSOR_DATA_SEND_INTRERVAL02	5

4.3 Optional Features Node Settings

The nodes with optional features (Low Power, Friend, or Relay) used in the Mesh Sensor Demo are described below.

For information on each type of features, refer to ‘RX23W Group Bluetooth Mesh Stack Development Guide’ (R01AN4875).

4.3.1 Relay Node

(1) Enabling Relay Feature

A node is activated as a Relay node through configuration by Mesh Mobile, which acts as the configuration client.

4.3.2 Low Power Node

(1) Enabling Low Power Feature

Enabling or disabling Low Power feature is accomplished by setting the value of the LOW_POWER_FEATURE_EN macro in the mesh_appl.h file.

The Sensor Server project contains a setting to enable Low Power feature. Also refer to 4.6.2.1(2), ‘mesh_appl.h’.

When a friendship is established, Low Power feature is activated on the Low Power node.

Table 4-3 LOW_POWER_FEATURE_EN Macro

Macro Name	Setting Value	Project
LOW_POWER_FEATURE_EN	1	Sensor Server project
	0	Projects other than the above

(2) Scan Suspend Interval

Suspending and resuming scanning and polling of the Friend node take place automatically. The Friend node is polled when scan resume occurs, and data reception takes place if there is stored data.

Scanning is suspended again after the data is transmitted. The interval from scan suspend to scan resume is controlled by the following macro.

Also refer to Figure 4-6, ‘Operation of Lower-Power Node and Friend Node’.

Table 4-4 Scan Suspend Interval Macro

Macro Name	Default Value (Unit: 100 ms)	Minimum Value	Maximum Value
CORE_FRIEND_POLLTIMEOUT	50	1	345,600 (96 hours)

Setting a long scan suspend interval lengthens the duration when scanning is halted and reduces power consumption, but it also lengthens the duration before requests from the Client node are received. This results in a delay before operations requested by the Sensor Client node start.

Also refer to 3.5.3, ‘Low Power Node Sequence’, in ‘RX23W Group Bluetooth Mesh Stack Development Guide’ (R01AN4875).

4.3.3 Friend Node

(1) Enabling Friend Feature

A node is activated as a Friend node through configuration by Mesh Mobile, which acts as the configuration client.

(2) Friend Node Operation

Figure 4-6 shows the operation of the Low Power node and the Friend node in the Mesh Sensor Demo.

For information on Friend node operation, refer to 1.10.3, ‘Friendship’, and 3.5, ‘Friendship’, in ‘RX23W Group Bluetooth Mesh Stack Development Guide’ (R01AN4875).

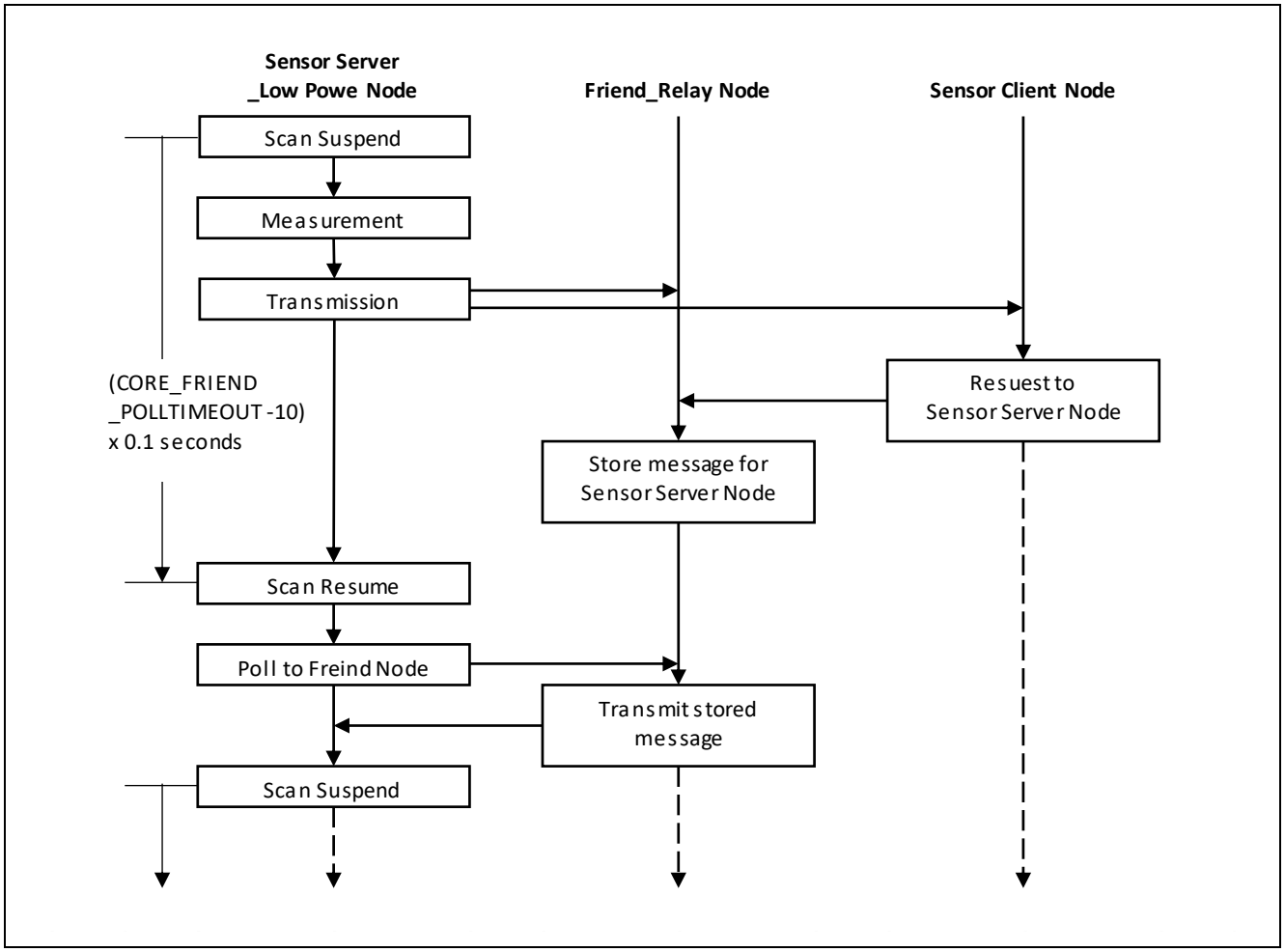


Figure 4-6 Operation of Lower-Power Node and Friend Node

4.4 Node Configuration Settings

The node configuration settings are described below. In this example the Mesh sample program mesh_model.c implements the settings. Configuring node configuration settings requires implementation of at least the following.

(1) Create node: MS_access_create_node()
(2) Register element: MS_access_register_element()
(3) Add Configuration Server Model: MS_config_server_init()
(4) Add Health Server Model: MS_health_server_init()
(5) Add Application Model

For information on (1) to (4), refer to 3.2, 'Node Composition, in the application note RX23W Group Bluetooth Mesh Stack Development Guide' (R01AN4875).

For specifics of (5), 'Add Application Model', refer to 4.6.2.1(3), 'mesh_model.c' and 4.6.2.2(3), 'mesh_model.c'.

4.5 Each Node Project and Mesh Sample Header File (mesh_appl.h) Macro Settings

The macro settings in the Mesh sample header file (mesh_appl.h) differ as shown below.

Table 4-5 Nodes and Projects Used

Node	**Sensor Server_Low Power**	**Sensor Client**	**Friend_Relay**	**Relay**
Project	tbrx23w_mesh_**sensor**_server	tbrx23w_mesh_**sensor**_client	tbrx23w_mesh_**sensor_friend**	

Table 4-6 Macro Settings in Mesh Sample Header File (mesh_appl.h) of Each Project

Macro Name	**tbrx23w_mesh _sensor_server**	**tbrx23w_mesh _sensor_client**	**tbrx23w_mesh _sensor_friend**
Enabling of IV update start processing IV_UPDATE_INITIATION_EN	(1)	(1)	(1)
Enabling of Low Power feature LOW_POWER_FEATURE_EN	(1)	(0)	(0)
Mesh monitor settings CONSOLE_MONITOR_LOG	(0)	(0)	(0)
Console output settings CONSOLE_OUT_EN	(1)	(1)	(1)
ANSI CSI output to console settings ANSI_CSI_EN	(1)	(1)	(1)
Enabling of CPU utilization rate measurement CPU_USAGE_EN	(0)	(0)	(0)

4.6 Program Differences

The differences among the projects are described below.

Table 4-7 lists the projects used as the basis for the Mesh Sensor Demo projects. Also refer to 1.1.2, 'MOT Files and e^2 studio Projects in Mesh Sensor Demo Package'.

Table 4-7 Projects Used as Basis for Mesh Sensor Demo Projects

Project Name	Node	Project Used as Basis
tbrx23w_mesh_**sensor**_**server**	Sensor Server_Low Power	tbrx23w_mesh_server
tbrx23w_mesh_**sensor**_**client**	Sensor Client	tbrx23w_mesh_client
tbrx23w_mesh_**sensor**_**friend**	Friend_Relay Relay	

In addition, the following change applies to all the projects.

— Update of BLE module to Ver. 2.30

4.6.1 Settings in r_xxx_config.h File Related to Mesh FIT Module

The following settings are configured in all the projects. The setting values are the same as those in the projects used as a basis.

(1) r_mesh_rx23w_config.h

The default values listed in 3.1, Mesh FIT Module, in Bluetooth Mesh Module Using Firmware Integration Technology (R01AN4930) are used.

(2) r_bsp_config.h

The values for the Mesh FIT module listed in 3.2, BSP FIT Module, in Bluetooth Mesh Module Using Firmware Integration Technology (R01AN4930) are used.

(3) r_ble_rx23w_config.h

The values for the Mesh FIT module listed in 3.3, BLE FIT Module, in Bluetooth Mesh Module Using Firmware Integration Technology (R01AN4930) are used.

4.6.2 tbrx23w_sensor_mesh_server and tbrx23w_sensor_mesh_client Projects

The differences related to incorporation of the Sensor Models and the HS300x module are shown below.

In order to simplify things when adding Sensor Model control, the structures and definitions for the Vendor Model are used.

Changes have been made under the following identifiers.

- DEMO_SENSOR_MODEL : Macro for Sensor Model
- DEMO_SENSOR_SERVER_MODEL : Macro for Sensor Server Model
- DEMO_SENSOR_CLIENT_MODEL : Macro for Client Model

Table 4-8 lists the modified files. The changes for each project are shown below.

Table 4-8 Modified Files

Finename	tbrx23w_mesh_sensor_server	tbrx23w_mesh_sensor_client
maim.c	Common files for Sensor Server and Sensor Client	
mesh_appl.h	For Sensor Server only (Same except for the value of the LOW_POWER_FEATURE_EN macro)	For Sensor Client only
meshu_model.c	Common files for Sensor Server and Sensor Client	
r_irq_rx.c		
Pin.c	For Sensor Server only	

4.6.2.1 tbrx23w_sensor_mesh_server Project

For files that are common to Sensor Server and Sensor Client, only the changes for Sensor Server are described.

(1) main.c

Processing to read the header file for the HS300x module.

Lines 48 to 50

```
#ifdef DEMO_SENSOR_SERVER_MODEL
#include "RX_HS300X.h"
#endif
```

Processing to change variable declarations and add variables for the Sensor Server Model.

Lines 114 to 121

```
#ifdef DEMO_SENSOR_SERVER_MODEL
#define SENSOR_DEMO_ADD (0.1)
static UINT32 gs_send_interval_timer_hdl;
static float g_demo_value_cnt = 0.0;
UINT16 g_send_Interval = 2000;
bool   g_interval_change_flg = false;
bool   g_send_flg = false;
#endif /* DEMO_SENSOR_SERVER_MODEL */
```

Processing to add a structure for the Sensor Models.

Lines 164 to 171

```
#ifdef DEMO_SENSOR_SERVER_MODEL
static void mesh_sensor_server_set_cb
                (
```

```
                    MS_ACCESS_MODEL_REQ_MSG_CONTEXT * ctx,
                    MS_STATE_VENDOR_STRUCT          * state
                );
MS_ACCESS_MODEL_REQ_MSG_CONTEXT g_ctx;
#endif /* DEMO_SENSOR_SERVER_MODEL */
```

Processing to set callback functions for the Sensor Models.

Lines 206 to 208

```
#ifdef DEMO_SENSOR_SERVER_MODEL
.sensor_server_set_cb = mesh_sensor_server_set_cb,
#endif /* DEMO_SENSOR_SERVER_MODEL */
```

Processing to transmit sensor data for the Sensor Server Model.

Note that pseudo data values are generated if no HS300x sensor is connected. The data starts from 0 and is then incremented by 0.1 with each transmission. Float data is used, so there may be errors in the smaller values.

Lines 299 to 357

```
#ifdef DEMO_SENSOR_SERVER_MODEL
static void sever_send_data_timer_cb(UINT32 timer_hdl)
{
    API_RESULT  retval;
    static MS_STATE_SENSOR_DATA_STRUCT  param;
    static MS_ACCESS_MODEL_STATE_PARAMS current_state_params;
    UCHAR  dummy_data[10] = {0};
    UINT16 dummy_len;
    float consol_dummy_01;
    float consol_dummy_02;
    uint8_t demo_sensro_less_value[sizeof(float)] = {0};

    if (true == g_send_flg)
    {
        if (0 == *(const float*)g_sensor_temp_value)
        {
            < Operation without HS300x Sensor >
        }
        else
        {
            < Operation with HS300x Sensor >
        }
        dummy_len = 8;
        param.property_id_1 = 0xAA;
        param.raw_value_1 = dummy_data;
        param.raw_value_1_len = dummy_len;
        current_state_params.state_type = MS_STATE_SENSOR_DATA_T;
        current_state_params.state = &param;

        retval = MS_sensor_server_state_update(&g_ctx, &current_state_params, NULL,
0, NULL, 0, 0);
        CONSOLE_OUT ("[SENSOR] Temperature %f['C]\n", consol_dummy_01);
        CONSOLE_OUT ("[SENSOR] Humidity    %f[%RH]\n", consol_dummy_02);
        CONSOLE_STATUS("[SENSOR] MS_sensor_server_state_update()", retval);
        CONSOLE_OUT ("\n");
    }
```

```
}
#endif /* DEMO_SENSOR_SERVER_MODEL */
```

Processing for the Sensor Models.

Lines 583 to 599

```
#ifdef DEMO_SENSOR_SERVER_MODEL
/***********************************************************************
* @brief Callback function to receive a new Vendor state
***********************************************************************/
static void mesh_sensor_server_set_cb
                (
                    MS_ACCESS_MODEL_REQ_MSG_CONTEXT * ctx,
                    MS_STATE_VENDOR_STRUCT          * state
                )
{
       --- Omitted ---
}
#endif /* DEMO_SENSOR_SERVER_MODEL */
```

Processing to configure timer settings when transmitting sensor data for the Sensor Server Model.

Lines 758 to 761

```
int main(void)
{
       --- Omitted ---
#ifdef DEMO_SENSOR_SERVER_MODEL
    R_BLE_TIMER_Create(&gs_send_interval_timer_hdl, g_send_Interval,
BLE_TIMER_PERIODIC, sever_send_data_timer_cb);
    R_BLE_TIMER_Start(gs_send_interval_timer_hdl);
#endif /* DEMO_SENSOR_SERVER_MODEL */
       --- Omitted ---
}
```

Processing to acquire sensor data for the Sensor Server Model.

Lines 780 to 783

```
int main(void)
{
       --- Omitted ---
#ifdef DEMO_SENSOR_SERVER_MODEL
    /* Initialize HS3001 Sensor Fit */
    sensor_hs3001_init();
#endif /* DEMO_SENSOR_SERVER_MODEL */
       --- Omitted ---
}
```

Processing to implement updating of the sensor data send interval by the Sensor Server node in response to a request from the Sensor Client node.

Processing to acquire sensor data using sensor_hs3001_main().

Lines 840 to 850

```
int main(void)
```

```
{
        --- Omitted ---
    /* main loop */
    while (1)
    {
            --- Omitted ---
#ifdef DEMO_SENSOR_SERVER_MODEL
        if (true == g_interval_change_flg)
        {
            R_BLE_TIMER_Stop(gs_send_interval_timer_hdl);
            R_BLE_TIMER_Create(&gs_send_interval_timer_hdl, g_send_Interval,
BLE_TIMER_PERIODIC, sever_send_data_timer_cb);
            R_BLE_TIMER_Start(gs_send_interval_timer_hdl);
            g_interval_change_flg = false;
        }

        sensor_hs3001_main();
#endif /* DEMO_SENSOR_SERVER_MODEL */
    }
        --- Omitted ---
}
```

(2) mesh_appl.h

Header file read processing is as follows.

- Vendor Model API: Some structures and definitions defined in the Vendor Model are used.
- Sensor Model API:
- Access Model API: Used to produce log output for subscription address acquisition, etc.

Lines 35 to 39

```
#if defined(DEMO_SENSOR_SERVER_MODEL) || defined(DEMO_SENSOR_CLIENT_MODEL)
#include "vendor_model/vendor_api.h"
#include "MS_sensor_api.h"
#include "MS_access_api.h"
#endif /* defined(VENDOR_SERVER_MODEL) || defined(VENDOR_CLIENT_MODEL) */
```

This setting enables Low Power feature. Low Power feature is enabled in the Sensor Server project. Also refer to 4.3.2(1), ‘Enabling Low Power Feature’.

Line 53

```
#define LOW_POWER_FEATURE_EN            (1)
```

Function declarations for callback functions.

Lines 217 to 227

```
#ifdef DEMO_SENSOR_SERVER_MODEL
void (*sensor_server_set_cb)(
                MS_ACCESS_MODEL_REQ_MSG_CONTEXT * ctx,
                MS_STATE_VENDOR_STRUCT          * state);
#endif /* DEMO_SENSOR_SERVERT_MODEL */
```

(3) mesh_model.c

As with the settings in the projects used as a basis, this code is also shared, with processing added for the Sensor Server Model and Sensor Client Model. The Vendor Model is used as a reference and Sensor Model professing has been added. Also, some structures and definitions defined in the Vendor Model are used.

Processing to declare variables for the Sensor Models.

Lines 62 to 69

```
#ifdef DEMO_SENSOR_SERVER_MODEL
static MS_ACCESS_MODEL_HANDLE   gs_sensor_server_model_handle;
static MS_ACCESS_MODEL_HANDLE   gs_sensor_setup_server_model_handle;
extern UINT16                   g_send_Interval;
extern bool                     g_interval_change_flg;
extern bool                     g_send_flg;
extern MS_ACCESS_MODEL_REQ_MSG_CONTEXT g_ctx;
#endif /* DEMO_SENSOR_SERVER_MODEL */
```

Processing to declare structures for the Sensor Models.

Lines 113 to 122

```
#ifdef DEMO_SENSOR_SERVER_MODEL
static API_RESULT mesh_model_sensor_server_cb
                (
                    MS_ACCESS_MODEL_REQ_MSG_CONTEXT * ctx,
                    MS_ACCESS_MODEL_REQ_MSG_RAW     * msg_raw,
                    MS_ACCESS_MODEL_REQ_MSG_T       * req_type,
                    MS_ACCESS_MODEL_STATE_PARAMS    * state_params,
                    MS_ACCESS_MODEL_EXT_PARAMS      * ext_params
                );
#endif /* DEMO_SENSOR_SERVER_MODEL */
```

Reception processing for the Sensor Server Model.

Lines 607 to 769

```
#ifdef DEMO_SENSOR_SERVER_MODEL
/*************************************************************************
* @brief Callback function to receive events for Vendor Server model
*************************************************************************/
static API_RESULT mesh_model_sensor_server_cb
                (
                    MS_ACCESS_MODEL_REQ_MSG_CONTEXT * ctx,
                    MS_ACCESS_MODEL_REQ_MSG_RAW     * msg_raw,
                    MS_ACCESS_MODEL_REQ_MSG_T       * req_type,
                    MS_ACCESS_MODEL_STATE_PARAMS    * state_params,
                    MS_ACCESS_MODEL_EXT_PARAMS      * ext_params
                )
{
    --- Omitted ---
}

#define MS_MAX_NUM_STATES    3
#define MS_MAX_SENSORS       1
#define CONSOLE_PRINT(...)

typedef struct _MS_SENSOR_STRUCT
```

```
{
    --- Omitted ---
} MS_SENSOR_STRUCT;

bool debug_sensor_led = false;
/**
 * Check if the sensor data to be published, based on the current value of the sensor data,
 * Fast Cadence High, Low range
 */
API_RESULT appl_handle_sensor_publish_timeout(/* IN */ MS_ACCESS_MODEL_HANDLE
* handle)
{
    --- Omitted ---
}

/**
 * \brief Access Layer Model Publication Timeout Callback.
 *
 * \par Description
 * Access Layer calls the registered callback to indicate Publication Timeout
 * for the associated model.
 *
 * \param [in]  handle        Model Handle.
 * \param [out] blob          Blob if any or NULL.
 */
API_RESULT mesh_sensor_server_publish_timout_cb
           (
                /* IN */ MS_ACCESS_MODEL_HANDLE * handle,
                /* IN */ void                   * blob
           )
{
    --- Omitted ---
}
#endif /* DEMO_SENSOR_SERVER_MODEL */
```

Configuration processing for the Sensor Server Model.

Lines 897 to 926

```
static API_RESULT mesh_model_config_server_cb
                (
                    MS_ACCESS_MODEL_REQ_MSG_CONTEXT * ctx,
                    UINT32                    opcode,
                    UCHAR                   * data_param,
                    UNIT16                    data_len
                )
{
    --- Omitted ---
        #ifdef DEMO_SENSOR_SERVER_MODEL
        if (opcode == gs_config_opcode_string_table[20].opcode)
        {
            g_send_flg = true;

            /* make MS_ACCESS_MODEL_REQ_MSG_CONTEXT for sending Sensor Status message
*/
            /** Model Handle - for which request is received */
            g_ctx.handle = gs_sensor_server_model_handle;
```

```
            /** Source Address - originator of request */
            MS_ACCESS_PUBLISH_INFO  publish_info;
            MS_access_cm_get_model_publication(gs_sensor_server_model_handle,
&publish_info);
            g_ctx.saddr = publish_info.addr.addr; // It assumes that Publish Address
is a address other than Virtual Address

            /** Destination Address - of the request */
            MS_access_cm_get_primary_unicast_address(&g_ctx.daddr);

            /** Associated Subnet Identifier */
            g_ctx.subnet_handle = 0x0000; // hard-coded: primary subnet

            /** Associated AppKey Identifier */
            g_ctx.appkey_handle = 0x0000; // hard-coded: first appkey

            CONSOLE_OUT("g_ctx.handle = 0x%04X\n", g_ctx.handle);
            CONSOLE_OUT("g_ctx.saddr = 0x%04X\n", g_ctx.saddr);
            CONSOLE_OUT("g_ctx.daddr = 0x%04X\n", g_ctx.daddr);
            CONSOLE_OUT("g_ctx.subnet_handle = 0x%04X\n", g_ctx.subnet_handle);
            CONSOLE_OUT("g_ctx.appkey_handle = 0x%04X\n", g_ctx.appkey_handle);
        }
        #endif /* DEMO_SENSOR_SERVER_MODEL */
```

Model registration processing for the Sensor Models.

Lines 1146 to 1163

```
static API_RESULT mesh_application_model_register(void)
{
    --- Omitted ---

    #ifdef DEMO_SENSOR_SERVER_MODEL
    retval = MS_sensor_server_init
                (
                    gs_element_handle,
                    &gs_sensor_server_model_handle,
                    mesh_model_sensor_server_cb,
                    mesh_sensor_server_publish_timout_cb
                );
    CONSOLE_STATUS("[SENSOR] MS_sensor_server_init()", retval);

    retval = MS_sensor_setup_server_init
                (
                    gs_element_handle,
                    &gs_sensor_setup_server_model_handle,
                    mesh_model_sensor_server_cb
                );
    CONSOLE_STATUS("[SENSOR] MS_sensor_setup_server_init()", retval);
    #endif /* DEMO_SENSOR_SERVER_MODEL */
```

(4) Pin.c

This settings file enables the HS300x sensor on the Sensor Server node to connect to the I²C bus. It is created using the “Generate Code”.

Therefore, there is no DEMO_SENSOR_SERVER_MODEL identifier.

Changes apply to the Pin.c file of the Sensor Server Mode project.

Lines 68 to 74

```
    /* Set SSCL1 pin */
    MPC.P30PFS.BYTE = 0x0AU;
    PORT3.PMR.BYTE |= 0x01U;

    /* Set SSDA1 pin */
    MPC.P26PFS.BYTE = 0x0AU;
    PORT2.PMR.BYTE |= 0x40U;
```

(5) r_irq_rx.c

Processing of the detection flag when SW1 is pressed for the Sensor Client Model.

This is for the Sensor Client Model, but the same file is used in the Sensor Server Mode project.

It does not work on the Sensor Server.

4.6.2.2 tbrx23w_sensor_mesh_server Project

For files that are common to Sensor Server and Sensor Client, only the changes for Sensor Client are described.

(1) main.c

Processing to read the header file for the HS300x module.

Lines 48 to 50

```
#ifdef DEMO_SENSOR_SERVER_MODEL
#include "RX_HS300X.h"
#endif
```

Processing to change variable declarations for the Sensor Client Model.

Lines 106 to 112

```
#ifdef DEMO_SENSOR_CLIENT_MODEL
static UCHAR  gs_sci_buf[MS_VENDOR_VALUE_MAX_SIZE];
static UINT16 gs_sci_max_len;
static UINT16 gs_sci_rcv_len;
static sci_comp_cb gs_sci_comp_cb = NULL;
extern unsigned char  g_server_setting_flg;
#endif /* DEMO_SENSOR_CLIENT_MODEL */
```

Processing to add a structure for the Sensor Models.

Lines 172 to 178

```
#ifdef DEMO_SENSOR_CLIENT_MODEL
static void mesh_sensor_client_status_cb
                (
                    MS_ACCESS_MODEL_REQ_MSG_CONTEXT * ctx,
                    MS_VENDOR_STATUS_STRUCT         * status
                );
#endif /* DEMO_SENSOR_CLIENT_MODEL */
```

Processing to set callback functions for the Sensor Models.

Lines 209 to 211

```
#ifdef DEMO_SENSOR_CLIENT_MODEL
.sensor_client_status_cb = mesh_sensor_client_status_cb,
#endif /* DEMO_SENSOR_CLIENT_MODEL */
```

Processing for the Sensor Models.

Lines 601 to 677

```
#ifdef DEMO_SENSOR_CLIENT_MODEL
/*****************************************************************************
* @brief Callback function to receive a state from Vendor Server model
*****************************************************************************/
static void mesh_sensor_client_status_cb
                (
                    MS_ACCESS_MODEL_REQ_MSG_CONTEXT * ctx,
```

```
                        MS_VENDOR_STATUS_STRUCT          * status
                    )
{
        --- Omitted ---
}

/**************************************************************************
* @brief Callback function for receiving and handling string from SCI
**************************************************************************/
static void sci_rcv_string(void)
{
        --- Omitted ---
}
#endif /* DEMO_SENSOR_CLIENT_MODEL */
```

Processing to repurpose processing for the Vendor Model for the sensor of the Sensor Models.

Lines 796 to 799

```
int main(void)
{
        --- Omitted ---
    /* main loop */
    while (1)
    {
            --- Omitted ---
        #ifdef DEMO_SENSOR_CLIENT_MODEL
        /* Receive and Handle String from SCI */
        sci_rcv_string();
        #endif /* DEMO_SENSOR_CLIENT_MODEL */
            --- Omitted ---
    }
        --- Omitted ---
}
```

Processing to request a change in the sensor data send interval by pressing SW1 for the Sensor Client Model.

Lines 808 to 839

```
int main(void)
{
        --- Omitted ---
    /* main loop */
    while (1)
    {
            --- Omitted ---
#ifdef DEMO_SENSOR_CLIENT_MODEL
        MS_SENSOR_CADENCE_SET_STRUCT param;
        static UCHAR  dummy_data = SENSOR_DATA_SEND_INTRERVAL01;
        API_RESULT    retval;
        if (1 == g_server_setting_flg)
        {
            if (SENSOR_DATA_SEND_INTRERVAL02 == dummy_data)
            {
                dummy_data = SENSOR_DATA_SEND_INTRERVAL01;
            }
            else if (SENSOR_DATA_SEND_INTRERVAL01 == dummy_data)
```

```
            {
                dummy_data = SENSOR_DATA_SEND_INTRERVAL02;
            }
            --- Omitted ---
        }
#endif /* DEMO_SENSOR_CLIENT_MODEL */
            --- Omitted ---
    }
        --- Omitted ---
}
```

(2) mesh_appl.h

Header file read processing is as follows.

- Vendor Model API: Some structures and definitions defined in the Vendor Model are used.
- Sensor Model API:
- Access Model API: Used to produce log output for subscription address acquisition, etc.

Lines 35 to 39

```
#if defined(DEMO_SENSOR_SERVER_MODEL) || defined(DEMO_SENSOR_CLIENT_MODEL)
#include "vendor_model/vendor_api.h"
#include "MS_sensor_api.h"
#include "MS_access_api.h"
#endif /* defined(VENDOR_SERVER_MODEL) || defined(VENDOR_CLIENT_MODEL) */
```

Two send intervals for sensor data are defined. Also refer to 4.2.3(3), ‘Changing the Sensor Data Send Interval’.

Lines 179 to 182

```
#ifdef DEMO_SENSOR_CLIENT_MODEL
#define SENSOR_DATA_SEND_INTRERVAL01 (2)
#define SENSOR_DATA_SEND_INTRERVAL02 (5)
#endif /* DEMO_SENSOR_CLIENT_MODEL */
```

Function declarations for callback functions.

Lines 223 to 227

```
#ifdef DEMO_SENSOR_CLIENT_MODEL
void (*sensor_client_status_cb)(
                MS_ACCESS_MODEL_REQ_MSG_CONTEXT * ctx,
                MS_VENDOR_STATUS_STRUCT         * state);
#endif /* DEMO_SENSOR_CLIENT_MODEL */
```

Function declarations for the Sensor Server Client.

Lines 261 to 265

```
#ifdef DEMO_SENSOR_CLIENT_MODEL
API_RESULT mesh_model_sensor_client_get(void);
API_RESULT mesh_model_sensor_client_set_unack(UCHAR * value, UINT16 len);
#endif /* DEMO_SENSOR_CLIENT_MODEL */
```

(3) mesh_model.c

As with the settings in the projects used as a basis, this code is also shared, with processing added for the Sensor Server Model and Sensor Client Model. The Vendor Model is used as a reference and Sensor Model professing has been added. Also, some structures and definitions defined in the Vendor Model are used.

Processing to declare variables for the Sensor Models.

Lines 70 to 72

```
w#ifdef DEMO_SENSOR_CLIENT_MODEL
static MS_ACCESS_MODEL_HANDLE   gs_sensor_client_model_handle;
#endif /* DEMO_SENSOR_CLIENT_MODEL */
```

Processing to declare structures for the Sensor Models.

Lines 123 to 131

```
#ifdef DEMO_SENSOR_CLIENT_MODEL
static API_RESULT mesh_model_sensor_client_cb
                (
                    MS_ACCESS_MODEL_REQ_MSG_CONTEXT * ctx,
                    UINT32                    opcode,
                    UCHAR                   * data_param,
                    UINT16                    data_len
                );
#endif /* DEMO_SENSOR_CLIENT_MODEL */
```

Reception processing for the Sensor Client Model.

Lines 770 to 872

```
#ifdef DEMO_SENSOR_CLIENT_MODEL
/******************************* ***************************************
* @brief Callback function to receive events for Vendor Client model
*****************************************************************************/
static API_RESULT mesh_model_sensor_client_cb
                (
                    MS_ACCESS_MODEL_REQ_MSG_CONTEXT * ctx,
                    UINT32                    opcode,
                    UCHAR                   * data_param,
                    UINT16                    data_len
                )
{
    --- Omitted ---
}

/*****************************************************************************
 * @brief Sends Vendor Get message
*****************************************************************************/
API_RESULT mesh_model_sensor_client_get(void)
{
    --- Omitted ---
}

/*****************************************************************************
 * @brief Sends Vendor Set message
*****************************************************************************/
API_RESULT mesh_model_sensor_client_set(UCHAR * value, UINT16 len)
{
```

```
    --- Omitted ---
}

/*****************************************************************************
* @brief Sends Vendor Set Unacknowledged message
*****************************************************************************/
API_RESULT mesh_model_sensor_client_set_unack(UCHAR * value, UINT16 len)
{
    --- Omitted ---
}
#endif /* DEMO_SENSOR_CLIENT_MODEL */
```

Model registration processing for the Sensor Models.

Lines 1165 to 1173

```
static API_RESULT mesh_application_model_register(void)
{
    --- Omitted ---

    #ifdef DEMO_SENSOR_CLIENT_MODEL
    retval = MS_sensor_client_init
                (
                    gs_element_handle,
                    &gs_sensor_client_model_handle,
                    mesh_model_sensor_client_cb
                );
    CONSOLE_STATUS("[SENSOR] MS_sensor_client_init()", retval);
    #endif /* DEMO_SENSOR_CLIENT_MODEL */

    --- Omitted ---
}
```

(4) r_irq_rx.c

Processing of the detection flag when SW1 is pressed for the Sensor Client Model.

This is for the Sensor Client Model, but the same file is used in the Sensor Server Mode project.

Lines 1015 to 1017 and 1027 to 1029

```
#ifdef DEMO_SENSOR_CLIENT_MODEL
unsigned char g_server_setting_flg = 0;
#endif
R_BSP_ATTRIB_INTERRUPT void irq5_isr(void)
{
    /* check callback address */
    if((FIT_NO_FUNC != (*(g_irq5_handle.pirq_callback))) && (NULL !=
(*(g_irq5_handle.pirq_callback))))
    {
        /* casting void * type to callback* type is valid */
        (*(g_irq5_handle.pirq_callback))((void*)&(g_irq5_handle.irq_num));
    }

#ifdef DEMO_SENSOR_CLIENT_MODEL
    g_server_setting_flg = 1;
#endif
```

```
} /* End of function irq5_isr */
```

4.6.3 tbrx23w_sensor_mesh_friend Project

This is the project for the following nodes.

- Friend_Relay node for TB-RX23W board
- Relay node for TB-RX23W board

There are no changes for the Sensor Models.

4.7 Global Variables

Table 4-9 lists the global variables added for the Sensor Models.

Table 4-9 Global Variables

Variable Name	Type	Remarks
g_send_Interval	UINT16	Stores the sensor data send interval change request value from the Sensor Client node.
g_interval_change_flg	bool	Set to “true” when a sensor data send interval change request value has been received from the Sensor Client node.
g_send_flg	bool	A value of “true” is input when entering of configuration settings from the smartphone app finishes.

4.8 Main Processing

A flowchart of the main processing is shown below.

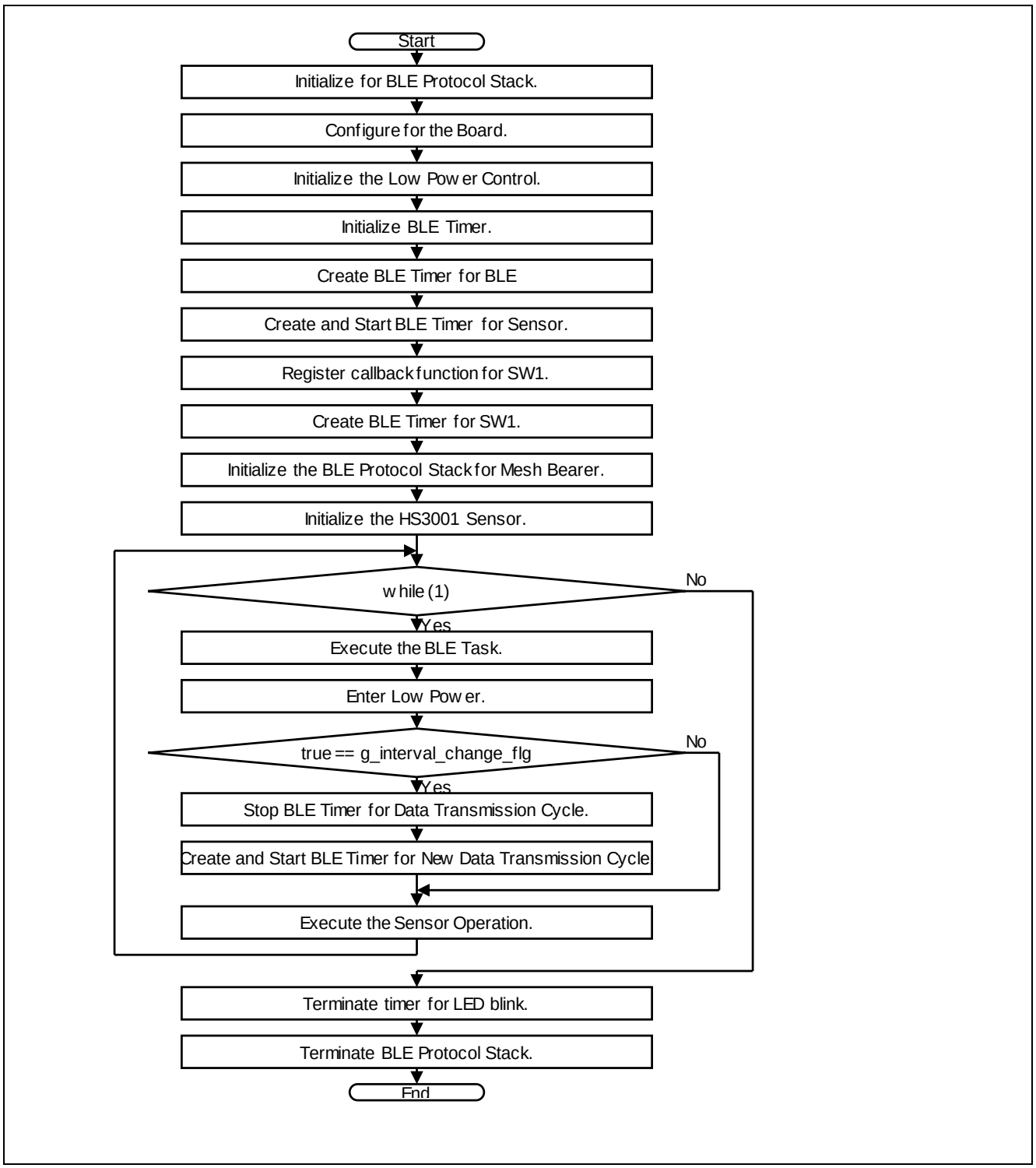


Figure 4-7 Main Processing Flowchart

5. Troubleshooting

Troubleshooting tips are presented below. Refer to them as needed when evaluating the application.

1	• Cannot find device for provisioning.
	— Try tapping **SCAN** once again.
2	• When I press **DISCONNECT** after configuring one node, the next unconfigured node is not found.
	— Try tapping **SCAN** once again.

Trademarks and Copyrights

The Bluetooth® word mark and logos are registered trademarks owned by Bluetooth SIG, Inc., and any use of such marks by Renesas Electronics Corporation is under license. Other trademarks or registered trademarks are the property of their respective owners.

RX23W Group Bluetooth Mesh stack uses the following open source software.

crackle: AES-CCM and AES 128-bit functionality

Android™ is a trademark of Google LLC.

Pmod™ is a trademark of Digilent Inc. of the United States of America.

Revision History

Rev.	Date	Description	
		Page	Summary
1.01	Feb. 7, 2022	—	First edition issued

General Precautions in the Handling of Microprocessing Unit and Microcontroller Unit Products

The following usage notes are applicable to all Microprocessing unit and Microcontroller unit products from Renesas. For detailed usage notes on the products covered by this document, refer to the relevant sections of the document as well as any technical updates that have been issued for the products.

1. Precaution against Electrostatic Discharge (ESD)

   A strong electrical field, when exposed to a CMOS device, can cause destruction of the gate oxide and ultimately degrade the device operation. Steps must be taken to stop the generation of static electricity as much as possible, and quickly dissipate it when it occurs. Environmental control must be adequate. When it is dry, a humidifier should be used. This is recommended to avoid using insulators that can easily build up static electricity. Semiconductor devices must be stored and transported in an anti-static container, static shielding bag or conductive material. All test and measurement tools including work benches and floors must be grounded. The operator must also be grounded using a wrist strap. Semiconductor devices must not be touched with bare hands. Similar precautions must be taken for printed circuit boards with mounted semiconductor devices.
2. Processing at power-on

   The state of the product is undefined at the time when power is supplied. The states of internal circuits in the LSI are indeterminate and the states of register settings and pins are undefined at the time when power is supplied. In a finished product where the reset signal is applied to the external reset pin, the states of pins are not guaranteed from the time when power is supplied until the reset process is completed. In a similar way, the states of pins in a product that is reset by an on-chip power-on reset function are not guaranteed from the time when power is supplied until the power reaches the level at which resetting is specified.
3. Input of signal during power-off state

   Do not input signals or an I/O pull-up power supply while the device is powered off. The current injection that results from input of such a signal or I/O pull-up power supply may cause malfunction and the abnormal current that passes in the device at this time may cause degradation of internal elements. Follow the guideline for input signal during power-off state as described in your product documentation.
4. Handling of unused pins

   Handle unused pins in accordance with the directions given under handling of unused pins in the manual. The input pins of CMOS products are generally in the high-impedance state. In operation with an unused pin in the open-circuit state, extra electromagnetic noise is induced in the vicinity of the LSI, an associated shoot-through current flows internally, and malfunctions occur due to the false recognition of the pin state as an input signal become possible.
5. Clock signals

   After applying a reset, only release the reset line after the operating clock signal becomes stable. When switching the clock signal during program execution, wait until the target clock signal is stabilized. When the clock signal is generated with an external resonator or from an external oscillator during a reset, ensure that the reset line is only released after full stabilization of the clock signal. Additionally, when switching to a clock signal produced with an external resonator or by an external oscillator while program execution is in progress, wait until the target clock signal is stable.
6. Voltage application waveform at input pin

   Waveform distortion due to input noise or a reflected wave may cause malfunction. If the input of the CMOS device stays in the area between V_{IL} (Max.) and V_{IH} (Min.) due to noise, for example, the device may malfunction. Take care to prevent chattering noise from entering the device when the input level is fixed, and also in the transition period when the input level passes through the area between V_{IL} (Max.) and V_{IH} (Min.).
7. Prohibition of access to reserved addresses

   Access to reserved addresses is prohibited. The reserved addresses are provided for possible future expansion of functions. Do not access these addresses as the correct operation of the LSI is not guaranteed.
8. Differences between products

   Before changing from one product to another, for example to a product with a different part number, confirm that the change will not lead to problems. The characteristics of a microprocessing unit or microcontroller unit products in the same group but having a different part number might differ in terms of internal memory capacity, layout pattern, and other factors, which can affect the ranges of electrical characteristics, such as characteristic values, operating margins, immunity to noise, and amount of radiated noise. When changing to a product with a different part number, implement a system-evaluation test for the given product.

Notice

1. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation or any other use of the circuits, software, and information in the design of your product or system. Renesas Electronics disclaims any and all liability for any losses and damages incurred by you or third parties arising from the use of these circuits, software, or information.
2. Renesas Electronics hereby expressly disclaims any warranties against and liability for infringement or any other claims involving patents, copyrights, or other intellectual property rights of third parties, by or arising from the use of Renesas Electronics products or technical information described in this document, including but not limited to, the product data, drawings, charts, programs, algorithms, and application examples.
3. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
4. You shall be responsible for determining what licenses are required from any third parties, and obtaining such licenses for the lawful import, export, manufacture, sales, utilization, distribution or other disposal of any products incorporating Renesas Electronics products, if required.
5. You shall not alter, modify, copy, or reverse engineer any Renesas Electronics product, whether in whole or in part. Renesas Electronics disclaims any and all liability for any losses or damages incurred by you or third parties arising from such alteration, modification, copying or reverse engineering.
6. Renesas Electronics products are classified according to the following two quality grades: “Standard” and “High Quality”. The intended applications for each Renesas Electronics product depends on the product’s quality grade, as indicated below.
   "Standard": Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; industrial robots; etc.
   "High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control (traffic lights); large-scale communication equipment; key financial terminal systems; safety control equipment; etc.

   Unless expressly designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not intended or authorized for use in products or systems that may pose a direct threat to human life or bodily injury (artificial life support devices or systems; surgical implantations; etc.), or may cause serious property damage (space system; undersea repeaters; nuclear power control systems; aircraft control systems; key plant systems; military equipment; etc.). Renesas Electronics disclaims any and all liability for any damages or losses incurred by you or any third parties arising from the use of any Renesas Electronics product that is inconsistent with any Renesas Electronics data sheet, user’s manual or other Renesas Electronics document.
7. No semiconductor product is absolutely secure. Notwithstanding any security measures or features that may be implemented in Renesas Electronics hardware or software products, Renesas Electronics shall have absolutely no liability arising out of any vulnerability or security breach, including but not limited to any unauthorized access to or use of a Renesas Electronics product or a system that uses a Renesas Electronics product. RENESAS ELECTRONICS DOES NOT WARRANT OR GUARANTEE THAT RENESAS ELECTRONICS PRODUCTS, OR ANY SYSTEMS CREATED USING RENESAS ELECTRONICS PRODUCTS WILL BE INVULNERABLE OR FREE FROM CORRUPTION, ATTACK, VIRUSES, INTERFERENCE, HACKING, DATA LOSS OR THEFT, OR OTHER SECURITY INTRUSION (“Vulnerability Issues”). RENESAS ELECTRONICS DISCLAIMS ANY AND ALL RESPONSIBILITY OR LIABILITY ARISING FROM OR RELATED TO ANY VULNERABILITY ISSUES. FURTHERMORE, TO THE EXTENT PERMITTED BY APPLICABLE LAW, RENESAS ELECTRONICS DISCLAIMS ANY AND ALL WARRANTIES, EXPRESS OR IMPLIED, WITH RESPECT TO THIS DOCUMENT AND ANY RELATED OR ACCOMPANYING SOFTWARE OR HARDWARE, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE.
8. When using Renesas Electronics products, refer to the latest product information (data sheets, user’s manuals, application notes, “General Notes for Handling and Using Semiconductor Devices” in the reliability handbook, etc.), and ensure that usage conditions are within the ranges specified by Renesas Electronics with respect to maximum ratings, operating power supply voltage range, heat dissipation characteristics, installation, etc. Renesas Electronics disclaims any and all liability for any malfunctions, failure or accident arising out of the use of Renesas Electronics products outside of such specified ranges.
9. Although Renesas Electronics endeavors to improve the quality and reliability of Renesas Electronics products, semiconductor products have specific characteristics, such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Unless designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not subject to radiation resistance design. You are responsible for implementing safety measures to guard against the possibility of bodily injury, injury or damage caused by fire, and/or danger to the public in the event of a failure or malfunction of Renesas Electronics products, such as safety design for hardware and software, including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult and impractical, you are responsible for evaluating the safety of the final products or systems manufactured by you.
10. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. You are responsible for carefully and sufficiently investigating applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive, and using Renesas Electronics products in compliance with all these applicable laws and regulations. Renesas Electronics disclaims any and all liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
11. Renesas Electronics products and technologies shall not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations. You shall comply with any applicable export control laws and regulations promulgated and administered by the governments of any countries asserting jurisdiction over the parties or transactions.
12. It is the responsibility of the buyer or distributor of Renesas Electronics products, or any other party who distributes, disposes of, or otherwise sells or transfers the product to a third party, to notify such third party in advance of the contents and conditions set forth in this document.
13. This document shall not be reprinted, reproduced or duplicated in any form, in whole or in part, without prior written consent of Renesas Electronics.
14. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products.

(Note1) “Renesas Electronics” as used in this document means Renesas Electronics Corporation and also includes its directly or indirectly controlled subsidiaries.

(Note2) “Renesas Electronics product(s)” means any product developed or manufactured by or for Renesas Electronics.

(Rev.5.0-1 October 2020)

Corporate Headquarters

TOYOSU FORESIA, 3-2-24 Toyosu,
Koto-ku, Tokyo 135-0061, Japan
www.renesas.com

Trademarks

Renesas and the Renesas logo are trademarks of Renesas Electronics Corporation. All trademarks and registered trademarks are the property of their respective owners.

Contact information

For further information on a product, technology, the most up-to-date version of a document, or your nearest sales office, please visit:
www.renesas.com/contact/.