RX23W Group Bluetooth Mesh Stack

Development Guide

Introduction

Bluetooth® Mesh Stack is the software library to build a mesh network that is compliant with Bluetooth Mesh Networking Specification and to perform many-to-many wireless communication.

This document describes the overview of software architecture and its layers of the Bluetooth Mesh Stack Package and how to develop a mesh application. For more information on how to get started with Bluetooth Mesh Stack Package, refer to "RX23W Group Bluetooth Mesh Stack Startup Guide" (R01AN4874).

Target Device

RX23W Group

Related Documents

The following documents are published on Renesas website.

Document Title	Document No.
RX23W Group User's Manual: Hardware	R01UH0823
CC-RX Compiler User's Manual	R20UT3248
Bluetooth Low Energy Protocol Stack Basic Package User's Manual	R01UW0205
RX23W Group Bluetooth Low Energy Application Developer's Guide	R01AN5504
RX23W Group Bluetooth Mesh Stack Startup Guide	R01AN4874
RX23W Group Bluetooth Mesh Stack Development Guide	R01AN4875 This document

Contents

1. Bluetooth Mesh Overview 4
1.1 Node 4
1.2 Element 4
1.3 Address 4
1.4 State 5
1.5 Model 5
1.5.1 Client and Server 5
1.5.2 Foundation Models 6
1.5.3 Configuration Model 6
1.5.4 Health Model 6
1.5.5 Publication and Subscription 7
1.6 Message 8
1.7 Mesh Bearer 9
1.8 Provisioning 10
1.9 Configuration 10
1.10 Optional Features 11
1.10.1 Relay 11
1.10.2 Proxy 12
1.10.3 Friendship 13

2. Bluetooth Mesh Stack Package 14
2.1 System Architecture 15
2.2 Mesh Application 16
2.2.1 Mesh Core Module 17
2.2.2 Mesh Model Module 17
2.2.3 Mesh Model Composition 17
2.2.3.1 Configuration Model 18
2.2.3.2 Health Model 20
2.2.3.3 Generic OnOff Model 21
2.2.3.4 Vendor Model 21
2.3 Bluetooth Mesh Stack 22
2.4 Bluetooth Bearer 24
2.4.1 Bearer Functions for Message Transmission and Reception 24
2.4.2 Bearer Functions for Connection Control 25
2.4.3 Mesh GATT Services 26
2.4.4 ADV Bearer Operation 27
2.4.5 GATT Bearer Operation 28
2.5 MCU Peripheral Functions 29
2.6 Mesh Sample Program Configuration 31
2.7 Bluetooth Bearer Configuration 33

2.8 Mesh Driver Configuration 36

3. Application Development 37

3.1 Main Routine 38
3.2 Node Composition 40
3.3 Provisioning 41
3.3.1 Provisioning Server 41
3.3.2 Provisioning Sequence 43
3.4 Proxy 47
3.4.1 Proxy Server 47
3.4.2 Proxy Client 48
3.4.3 Proxy Sequence 50
3.5 Friendship 52
3.5.1 Friend Node 52
3.5.2 Low Power Node 52
3.5.3 Low Power Node Sequence 54
3.5.4 Friend Node Sequence 56
3.6 Configuration 58
3.6.1 Configuration Server 58
3.6.2 Configuration Server Sequence 59
3.7 Health Model 60
3.7.1 Health Server 60
3.7.2 Health Server Sequence 62
3.8 Application Model 63
3.8.1 Server Model 63
3.8.2 Client Model 66
3.8.3 Generic OnOff Model Sequence 68
3.8.4 Vendor Model Sequence 68
3.8.5 Mesh Monitoring 69
3.8.5.1 Mesh Monitoring Sequence 72

4. Appendix 73

4.1 Command Line Interface Program 73

1. Bluetooth Mesh Overview

This chapter describes the basic concepts defined by Bluetooth Mesh Networking Specifications. For more information, refer to Mesh Networking Specifications. on Bluetooth SIG website.

Figure 1-1 shows the basic composition of Bluetooth mesh network.

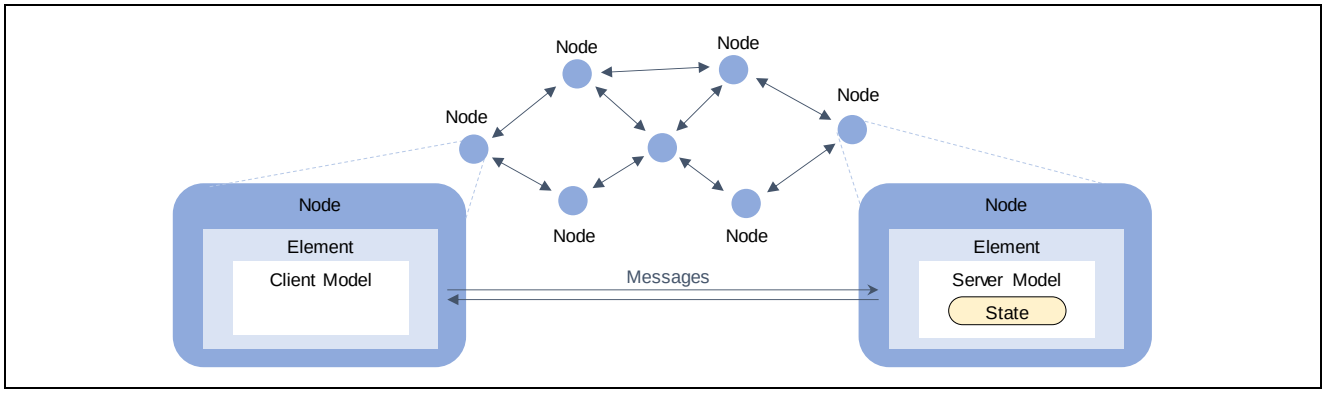

Figure 1-1 Basic Composition of Bluetooth Mesh Network

1.1 Node

A device joining a network is referred to as a Node. Network is a group of nodes sharing common address space and encryption keys. Communication among nodes is encrypted with Network Key. Each network is identified by Network ID generated by the Network Key. By default, one Network referred to as primary subnet is built. Multiple subnets can be also built to isolate communication scope.

1.2 Element

Element is a logical entity within a node. A node has at least one element and can also have multiple elements. First element is referred to as primary element.

Each element is identified uniquely in a network by Unicast Address that is assigned by Provisioning.

1.3 Address

Address used in a network is 16-bit length. Unassigned address, Unicast Address, Virtual Address, and Group address are defined as address types.

Table 1-1 Address Types

Address Type	Value	Value Range
Unassigned Address	0b0000000000000000	0x0000
Unicast Address	0b0xxxxxxxxxxxxxxx (excluding 0b0000000000000000)	0x0001 to 0x7FFF
Virtual Address	0b10xxxxxxxxxxxxxx	0x8000 to 0xBFFF
Group Address	0b11xxxxxxxxxxxxxx	0xC000 to 0xFFFF

- **Unassigned Address**

  Unassigned Address is set to an element which Unicast Address has not been assigned yet. This address cannot be used as source address or destination address in a message.

- **Unicast Address**

  Unicast Address is an address to identify a single element. 32,767 Unicast Address can be used in a network. Unicast Address can be used for source address and destination address in a message.

- **Virtual Address**

  Virtual Address is a multicast address generated by a Label UUID. Virtual Address can be used for destination address in a message.

  Label UUID is a 128-bit value to categorize multiple elements. This value can be generated randomly and shared by OOB (Out-Of-Band) among devices. Also, Virtual address and Label UUID need not to be managed centrally.

- **Group Address**

  Group Address is a multicast address managed and assigned dynamically for any usage. Group Address can be used for destination address in a message. Also, Fixed Group Addresses such as all-nodes are defined.

Table 1-2 Fixed Group Addresses

Fixed Group Address	Value
all-proxies	0xFFFC
all-friends	0xFFFD
all-relays	0xFFFE
all-nodes	0xFFFF

1.4 State

State is a value representing a condition of an element. States that are composed of two or more values are known as composite states. Moreover, States having a relationship whereby one state changes in conjunction with other state is referred to as bound states.

The State can change instantaneously or over a period of time. Time from initial State to target State is referred to as transition time. Also, time from current State to target State is referred to as remaining time.

1.5 Model

Model is a standardized typical functionality so that nodes perform operations in accordance with application scenario. Model defines States, Messages that act upon a state, and associated behaviors.

1.5.1 Client and Server

Model is a Server - Client architecture. Server model have at least on state, while Client model does not have state.

Client model can get a state of Server model with GET message or set a new state to Server model with SET message or SET Unacknowledged message.

Server model sends STATUS message as an ACK when State is changed, or an GET message or SET message is received. Server model does not send STATUS message as an ACK when SET Unacknowledged message is received.

Node can have multiple elements. Also, each element can use multiple models that are different from each other. Server model controls states of the element by receiving message from client model.

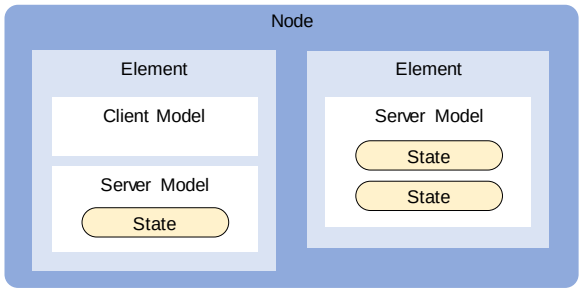

Figure 1-2 Node Composition

1.5.2 Foundation Models

Foundation Models are models to configure and manage operations of elements. Primary element of each node must have Configuration Server model and Health Server model.

Table 1-3 Fixed Group Addresses

Model	SIG Model ID
Configuration Server	0x0000
Configuration Client	0x0001
Health Server	0x0002
Health Client	0x0003

1.5.3 Configuration Model

Configuration Model is a model for configuring operations of node. Configuration values of a node and elements are defined as Configuration states.

Configuration Server Model is a model that has Configuration states. On the other hand, Configuration Client Model is a model for managing configurations of Configuration Server by Configuration messages. Each Configuration message is encrypted with a Device Key. Device Keys are different from each node.

1.5.4 Health Model

Health Model is a model for monitoring the physical condition of a node.

Health Server Model is a model that has Fault State for storing a physical fault information. On the other hand, Health Client Model is a model for monitoring fault information of Health Server by Health messages. Each Health message is encrypted with an Application Key.

1.5.5 Publication and Subscription

Transmission operation of model messages is referred to as Publication, and reception operation of them is referred to as Subscription respectively. Model can publish messages to multiple elements by setting a Multicast Address to destination address. Also, model can subscribe messages sent to Multicast Address selectively.

Figure 1-3 shows the message publication and Subscription by models. Each model sends messages in accordance with the Publish Address in Model Publication state. If the publish address is multicast address, each message is subscribed by multiple models in accordance with Subscription Addresses in Subscription List state.

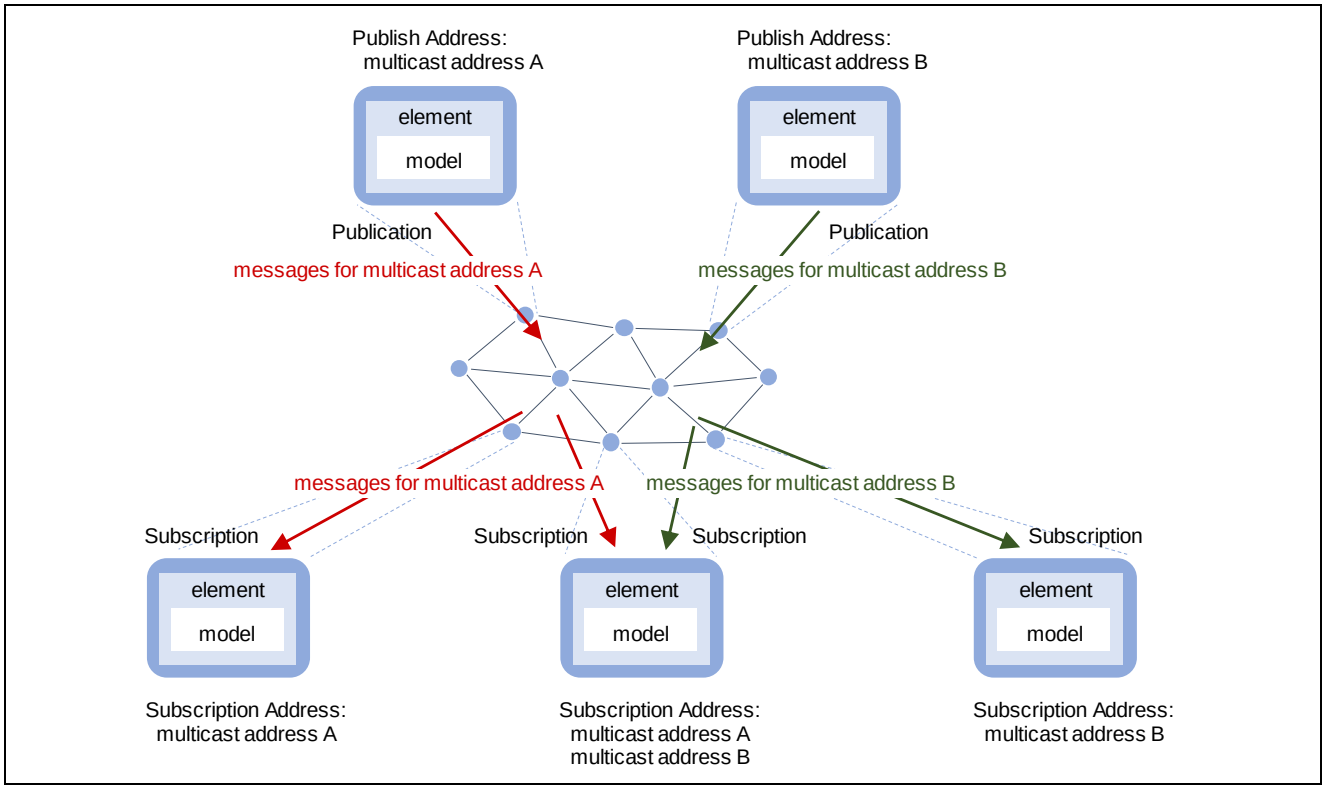

Figure 1-3 Message Publication and Subscription by models

1.6 Message

Data transmitted and received in a network is referred to as Message.

Unsegmented Message and Segmented Message are defined as message formats.

- **Unsegmented Message**

  Unsegmented message is a message to transport unsegmented data. It can transport Access PDU up to 11 bytes.

- **Segmented Message**

  Segmented Message is a message to transport each segmented data up to 32 segments. It can transport Access PDU up to 380 bytes. When receiving all Segmented Messages, destination node reassembles data.

  Figure 1-4 shows the segmentation and Reassembly of Access PDU. Each node transmits and receives Network PDU as a Mesh Message.

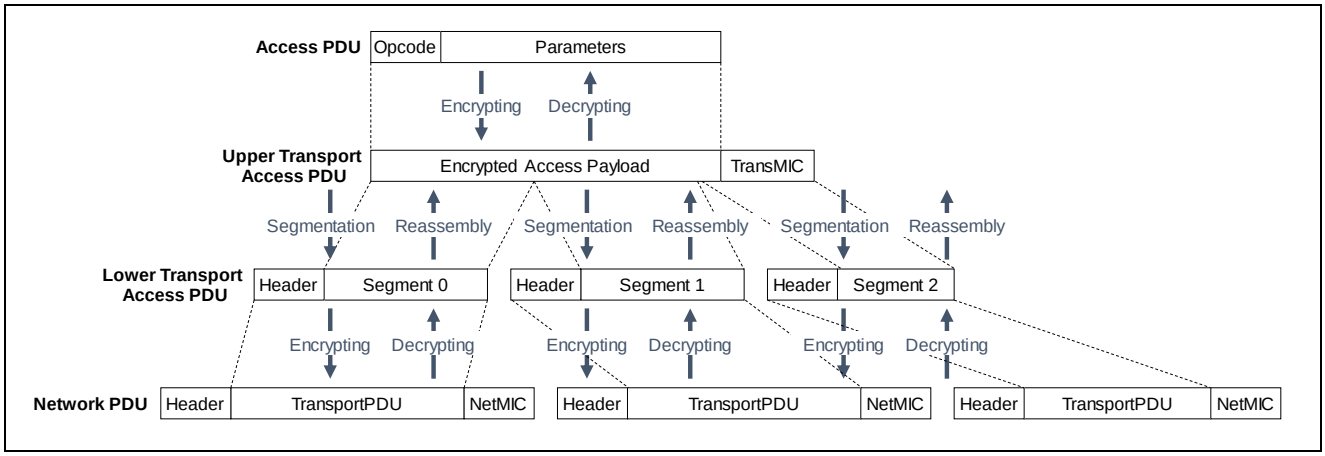

Figure 1-4 Segmentation and Reassembly of Access PDU

Header of Network PDU includes fields such as Source Address (SRC), Destination Address (DST), and Sequence Number (SEQ). Network PDUs are encrypted with a Network Key, so only devices joining same network can decrypt them. Also, Source Address and Destination Address of them are obfuscated, so other devices that does not have Network Key cannot trace them.

Header of Lower Transport PDU includes fields such as such as SEG to indicate whether Unsegmented or Segmented and SeqZero, SegO, and SegN to reassemble segmented data.

Access PDU is composed of two fields: Application Opcode and Application Parameters. Also, Access PDU is encrypted with Application Key or Device Key, so data can be share among only nodes that share the Application Key or Device Key. Applications Keys are generated and are distributed by Configuration Client.

1.7 Mesh Bearer

Mesh Bearer is a method to transport messages in a mesh network. Two bearers using Bluetooth Low Energy technology are defined as follows:

- **ADV bearer**

  This bearer sends messages by the Non-Connectable and Non-Scannable Undirected Advertising. Messages sent by ADV bearer can be received by many nodes simultaneously.

  Also, this bearer is referred to as PB-ADV when it transmits Provisioning PDUs during Provisioning.

- **GATT bearer**

  This bearer sends messages over GATT service. A node of Client side sends messages by Write Without Response and a node of Server side sends messages by Notification. Before communicating over the GATT service, establishing a connection is required. Messages sent by GATT bearer can be received by a connected peer node only.

  Also, this bearer is referred to as PB-GATT when it transmits Provisioning PDUs during Provisioning.

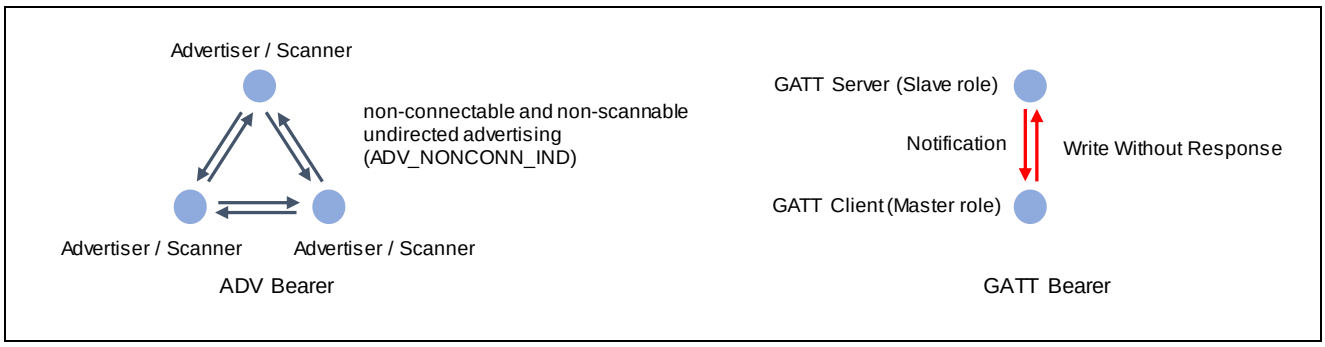

Figure 1-5 ADV Bearer and GATT Bearer

1.8 Provisioning

Provisioning is a process for joining a network. In provisioning, Provisioning Data that includes Network Key and Unicast Addresses of each element is distributed. Provisioning Data contains the following information.

- Network Key and Network Key Index
- Flags: Key Refresh Flag and IV Update Flag
- Current IV Index
- Unicast Address of the primary element

A device that is not joined yet is referred to as Unprovisioned Device. Each Unprovisioned Device is identified by 128-bit Device UUID.

A device that invites other devices and distributes Provisioning Data is referred to as Provisioning Client or Provisioner. Generally, Provisioning Client is a smart phone or other mobile computing device.

A device that receives Provisioning Data and joins a network is referred to as Provisioning Server or Provisionee. The device that has joined a network is referred to as a Node.

1.9 Configuration

To communicate with other nodes by using Models, each node needs Configuration. By Configuration process, information required for Model operation such as Application Keys, Publish Address, Subscription Address is configured.

Figure 1-6 shows a typical lifecycle of a node.

Newly introduced device is provisioned by Provisioner and joins a network. Furthermore, this device is configured by Configuration Client and becomes to be able to communicate with other nodes with Mesh Model. Generally, Configuration Client is a smart phone or other mobile computing device.

Configuration Client removes a node from a network by sending Config Node Reset message. Besides, Configuration Client updates encryption keys used in the network, and the removed node becomes unable to communicate with other nodes.

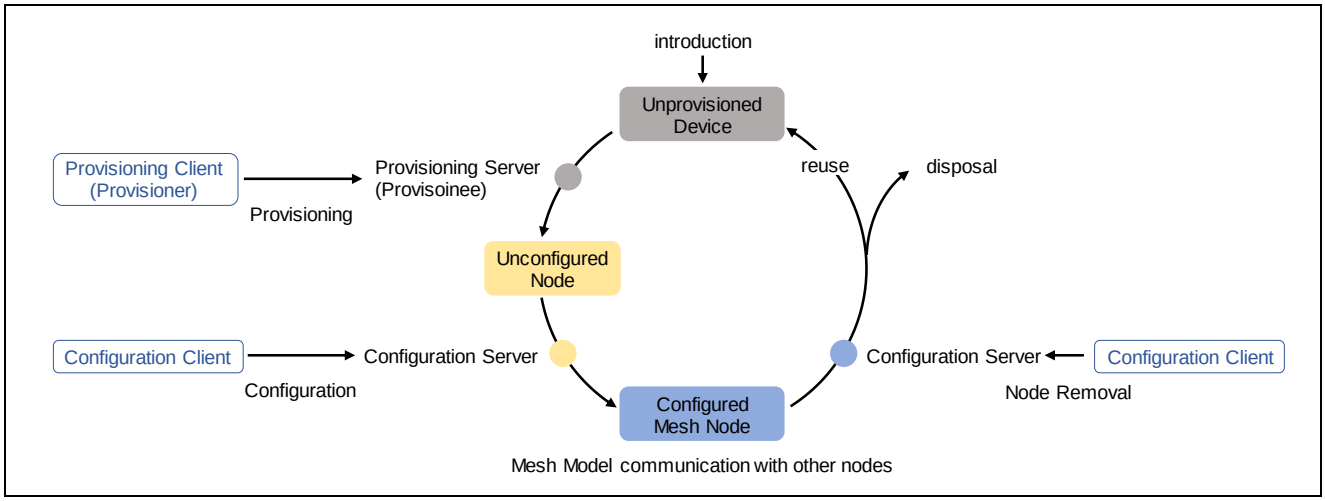

Figure 1-6 Lifecycle of a node

1.10 Optional Features

The following features are defined as Optional Features.

- Relay feature
- Proxy feature
- Friend feature
- Low Power feature

It is possible to form various mesh network by enabling each optional features of nodes.

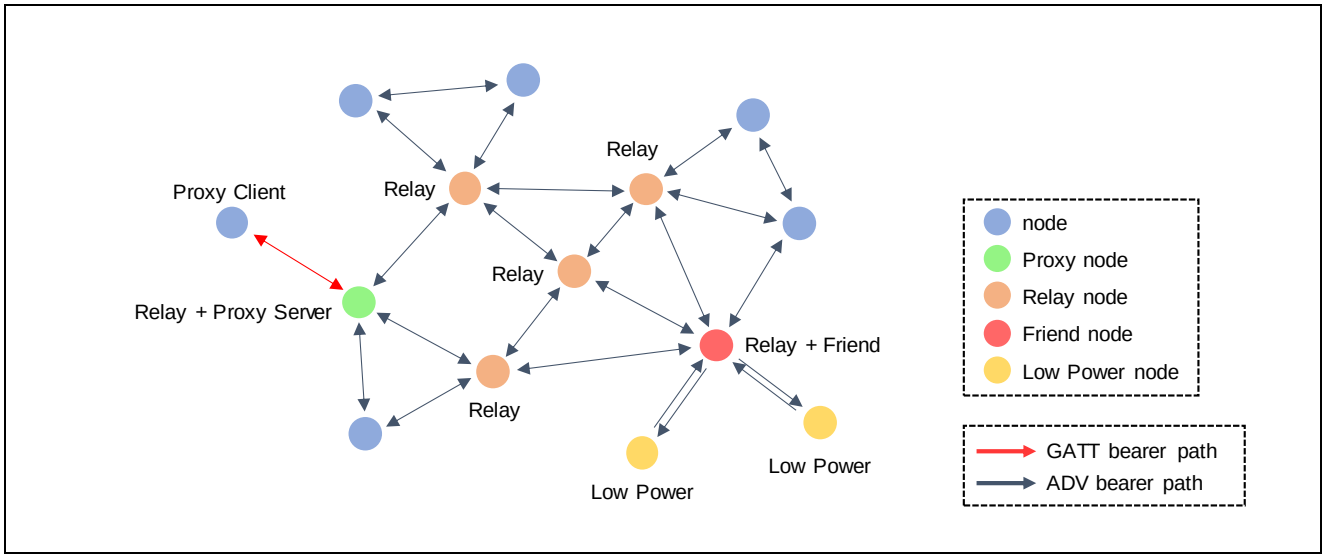

Figure 1-7 Mesh Network

1.10.1 Relay

Relay feature is the ability that a node supporting ADV bearer relays messages by retransmit them received over ADV bearer. Even if destination node is out direct radio range of an originator, messages are relayed by other nodes and spreads throughout a network, then the messages can reach the destination node.

A node that relays message is referred to as a Relay node.

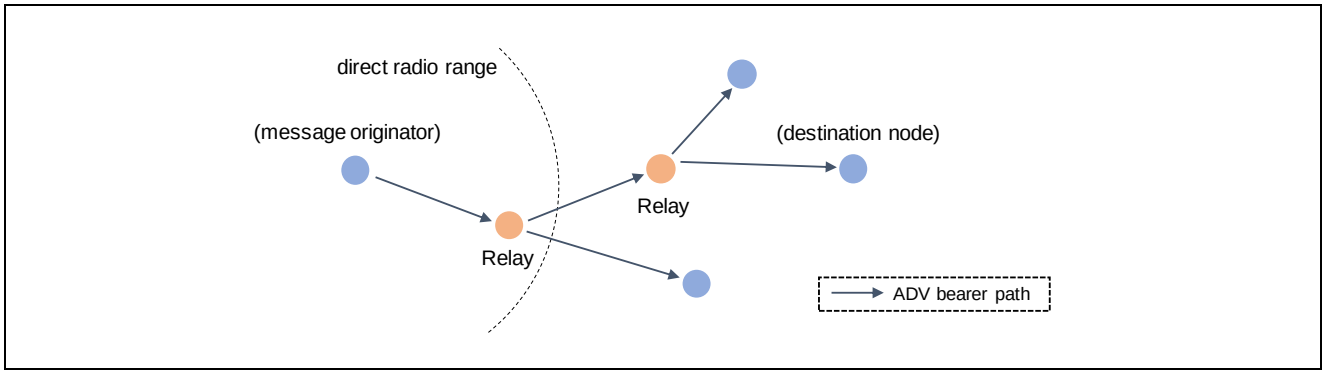

Figure 1-8 Relay

1.10.2 Proxy

Proxy feature is the ability that a node supporting both GATT bearer and ADV bearer forwards messages between both bearers.

A node supporting only GATT bearer communicates with a connected peer node only. In this case, this node establishes a connection with a node that supports Proxy feature. Thereby messages sent by this node are forwarded by ADV bearer of the Proxy node, then they can reach the destination node. Moreover, messages sent by other nodes are forwarded by GATT bearer of the Proxy node, and they can reach this node.

A node that transmits messages between both GATT bearer and ADV bearer is referred to as a Proxy Server. Also, A node that connects with Proxy Server and then transmits and receives messages over GATT bearer is referred to as a Proxy Client.

Proxy Server has a list to manage Subscription Addresses of Proxy Client, and it is referred to as a Proxy Filter List. Either white list filter or black list filter can be set as a Proxy Filter Type. When Proxy Filter Type is white list filter, Proxy Server forwards only messages addressed to the address registered in the list. When Proxy Filter Type is black list filter, Proxy Server does not forward messages addressed to the address registered in the list.

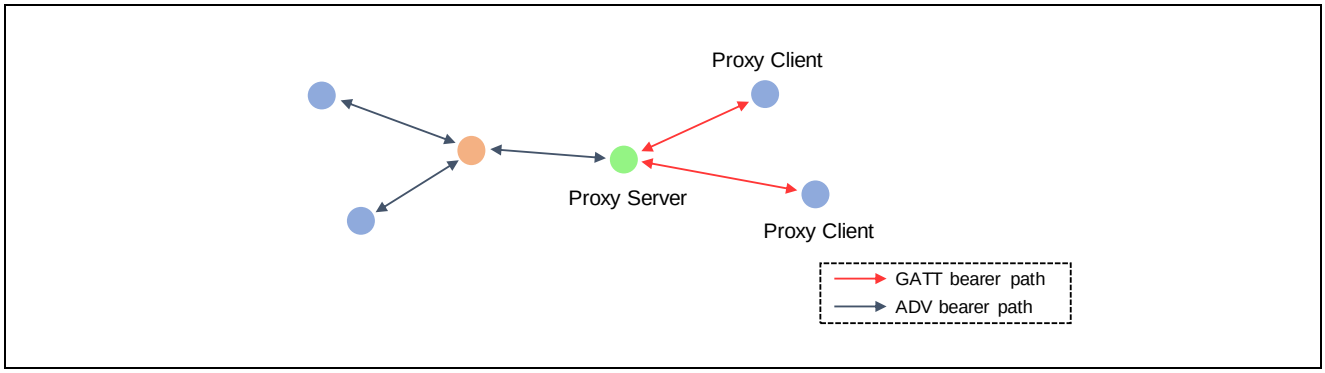

Figure 1-9 Proxy

1.10.3 Friendship

In general, a node supporting ADV bearer always perform Scan, to receive Advertising packets including messages. Low Power feature is the ability to reduce Scan duty cycle. A node supporting Low Power feature can reduce power consumption by suspending Scan.

To perform Low Power feature, the node must establish a Friendship with one node supporting Friend feature. Friend feature is the ability that stores incoming messages needed by Low Power node and then forwards them when Low Power node requests.

First, Low Power node requests a Friend node to become its friend. When the Friend node accepts it, Friendship is established. After establishing, Low Power node can suspend Scan, while Friend node must store received messages addressed to Low Power node.

Friend node has a list to manage Subscription Addresses of Low Power node, and it is referred to as a Friend Subscription List. After establishing a Friendship, Friend node stores messages addressed to Subscription Addresses registered in the list.

Low Power node polls Friend node intermittently if any messages are stored and resumes Scan only within a polling period. Friend node forwards the stored messages at this timing.

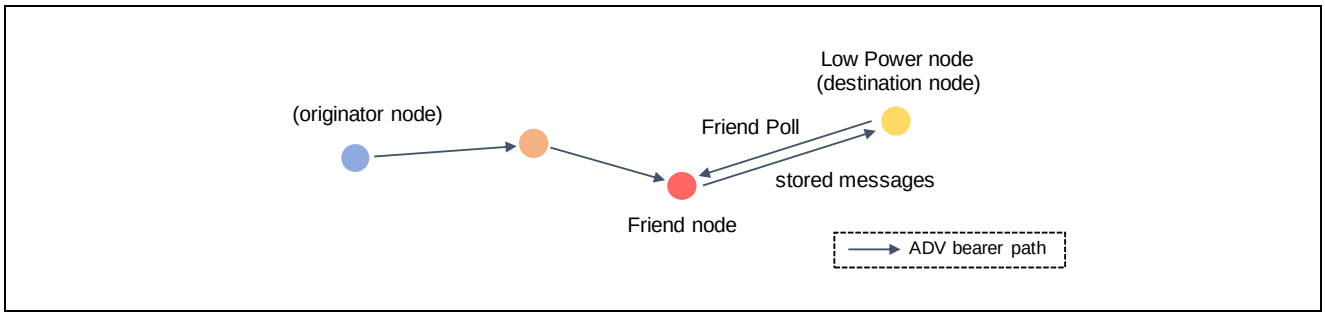

Figure 1-10 Friendship

2. Bluetooth Mesh Stack Package

This chapter explains the overview of the software included in Bluetooth Mesh Stack Package.

Bluetooth Mesh Stack package includes not only sample program of Mesh Application but also Bluetooth Mesh Stack, Bluetooth Bearer, Bluetooth Low Energy Protocol Stack, and other FIT Modules that are needed to build the sample program. In addition, the following demo projects are included.

- tbrx23w_mesh_client: Project for Target Board - Client Models
- tbrx23w_mesh_server: Project for Target Board - Server Models
- rsskrx23w_mesh_client: Project for Renesas Solution Starter Kit - Client Models
- rsskrx23w_mesh_server: Project for Renesas Solution Starter Kit - Server Models

Composition of demo project is shown as below. This document describes software indicated in bold. For details of other FIT Modules, refer to each application note.

```
{project}\                              :
    +---src\                            :
        |   main.c                      :  Mesh Sample Program
        |   mesh_appl.h                 :  Mesh Sample Header
        |   mesh_core.c                 :  Mesh Core Module
        |   mesh_model.c                :  Mesh Model Module
        |                               :
        +---vendor_model\               :
        |       vendor_api.h            :  Vendor Model Header
        |       vendor_client.c         :  Vendor Client Module
        |       vendor_server.c         :  Vendor Server Module
        +---smc_gen\                    :
            +---r_mesh_rx23w\           :  Mesh FIT Module (R01AN4930)
            |   +---lib\                :  - Mesh Stack Library
            |   +---src\                :
            |       +---bearer\         :  - Bluetooth Bearer
            |       +---drivers\        :  - Mesh Driver
            |       +---include\        :  - Mesh Stack Header
            +---r_ble_rx23w\            :  BLE FIT Module (R01AN4860)
            |   +---lib\                :  - Bluetooth Low Energy Protocol Stack Library
            |   +---src\                :
            |       +---app_lib\        :  - Application Library
            |       +---platform\       :  - Platform Module
            +---r_bsp\                  :  BSP FIT Module (R01AN1685)
            +---r_byteq\                :  BYTEQ FIT Module (R01AN1683)
            +---r_cmt_rx\               :  CMT FIT Module (R01AN1856)
            +---r_flash_rx\             :  FLASH FIT Module (R01AN2184)
            +---r_gpio_rx\              :  GPIO FIT Module (R01AN1721)
            +---r_irq_rx\               :  IRQ FIT Module (R01AN1668)
            +---r_lpc_rx\               :  LPC FIT Module (R01AN2769)
            +---r_sci_rx\               :  SCI FIT Module (R01AN1815)
            +---r_config\               :  FIT Module Configuration
            +---r_pincfg\               :  MCU Pin Configuration
```

Regarding how to setup an environment for building sample program, refer to Chapter 6 in "RX23W Group Bluetooth Mesh Stack Startup Guide" (R01AN4874)

2.1 System Architecture

Figure 2-1 shows the system architecture of Bluetooth Mesh Stack Package.

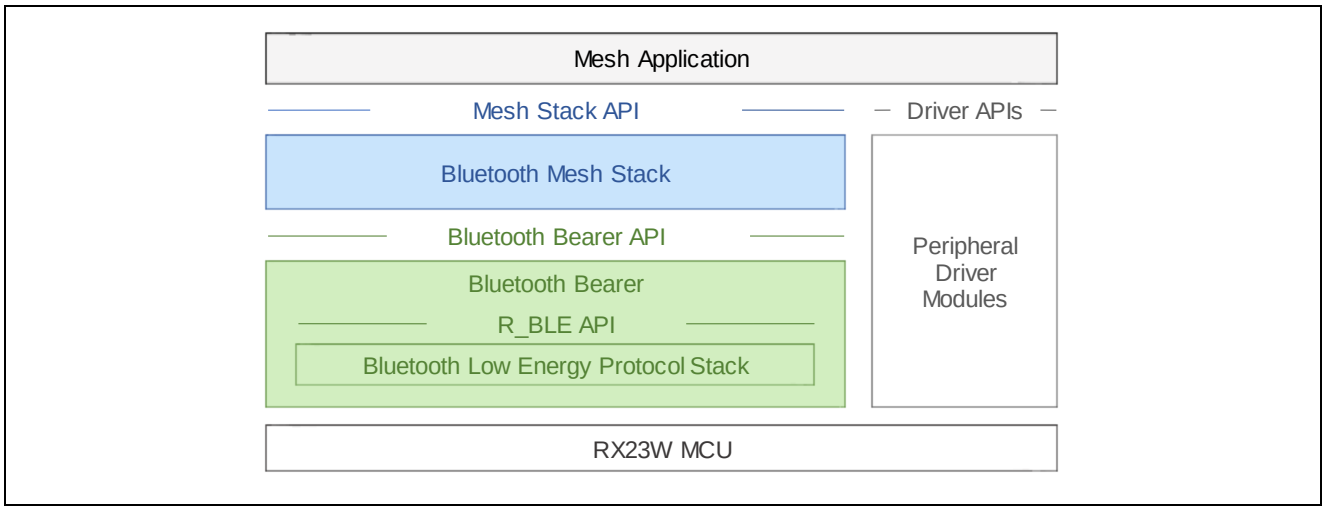

Figure 2-1 System Architecture of Bluetooth Mesh

Bluetooth Mesh Stack Package is composed of the following software:

- **Mesh Application**

  The Mesh Application is an application program that performs Bluetooth mesh communication features. Users are required to understand specification of Mesh Stack API and Bluetooth Bearer API to develop own Mesh Applications. Also, sample program of Mesh Application is included in Bluetooth Mesh Stack package.

- **Bluetooth Mesh Stack**

  The Bluetooth Mesh Stack (hereinafter referred to as "Mesh Stack") is the software stack that provides applications with many-to-many wireless communication features which is compliant with the Bluetooth Mesh Networking specifications. This stack has Mesh Stack API to use mesh network communication features. Also, Mesh Stack is included in RX23W Group Mesh FIT Module.

- **Bluetooth Bearer**

  The Bluetooth Bearer is the abstraction layer that provides the Bluetooth Mesh Stack and application with wrapper functions of Bluetooth Low Energy Protocol Stack. Also, Bluetooth Bearer is included in RX23W Group Mesh FIT Module.

- **Bluetooth Low Energy Protocol Stack**

  The Bluetooth Low Energy Protocol Stack (hereinafter referred to as " Bluetooth LE Stack") is the software that provides upper layers with wireless communication features which is compliant with the Bluetooth Low Energy specifications. This stack has R_BLE API to use Bluetooth Low Energy communication features. Also, Bluetooth LE Stack is included in RX23W Group BLE FIT Module.

- **Peripheral Driver Modules**

  Application, Mesh Stack, Bluetooth LE Stack use peripheral functions of microcontroller. Peripheral drivers that are provided as Firmware Integration Technology (FIT) Modules can used for developing software for RX microcontrollers.

2.2 Mesh Application

Users is required to develop own Mesh Application for performing wireless communication capability with Bluetooth Mesh. Bluetooth Mesh Stack package includes source code of sample program that can be used as a reference for developing Mesh Applications.

The sample program of Mesh Application (hereinafter referred to as "Mesh Sample Program) uses the API of Mesh Stack and performs Provisioning and basic operations as a mesh node. This section describes the detail of Mesh Sample Program.

Notable features of Mesh Sample Program are shown as below:

- Unprovisioned Device operation: supports both PB-ADV bearer and PB-GATT bearer
- Configuration Server operation: stores Configuration information in Data Flash memory
- Generic OnOff Client operation: sends Generic OnOff Set message when on-board switch is pushed
- Generic OnOff Server operation: controls on-board LED when Generic OnOff Set message is received.
- Vendor Client operation: sends Vendor Set message with character string input over UART
- Vendor Server operation: outputs character string included in Vendor Set message received
- Low Power operation: establishes a Friendship to Friend node and registers Subscription List with Friend Subscription List
- Proxy Server operation: establish a connection to Proxy Client and forwards messages over GATT bearer
- IV Update Initiation functionality: monitors sequence number of messages and initiates IV update procedure when the sequence number exceeds threshold value.
- Mesh Monitoring functionality: monitors incoming and outgoing messages and outputs log to console over UART

This sample program includes the following two modules:

- **Core Mesh Module**

  This module performs Provisioning as a Provisioning Server and enables GATT bearer as a Proxy Server after Provisioning. In addition, this module controls a Friendship as a Low Power Node.

- **Mesh Model Module**

  This module performs operations associated with Generic OnOff models and original Vendor models as well as Configuration Server model and Health Server model.

2.2.1 Mesh Core Module

Mesh Core Module included in Mesh Sample Program performs the following operations. This module is implemented in "mesh_core.c".

- Provisioning process
- Proxy feature
- Low Power feature
- IV Update process
- Mesh Monitoring functionality

2.2.2 Mesh Model Module

Mesh Model module included in Mesh Sample Program performs the following operations. This module is implemented in "mesh_model.c".

- Mesh Model Composition
- Configuration Model
- Generic OnOff Model
- Vendor Model

2.2.3 Mesh Model Composition

This sample program uses the following model.

- Configuration Server model
- Health Server model
- Generic OnOff Server model
- Generic OnOff Client model
- Vendor Server model
- Vendor Client model

Figure 2-2 show the model compositions of Mesh Sample Program of each project. Generic OnOff Client model, Generic OnOff Server model, Vendor Client model, and Vendor Server model as well as Configuration Server model and Health Server model are located on the Primary element.

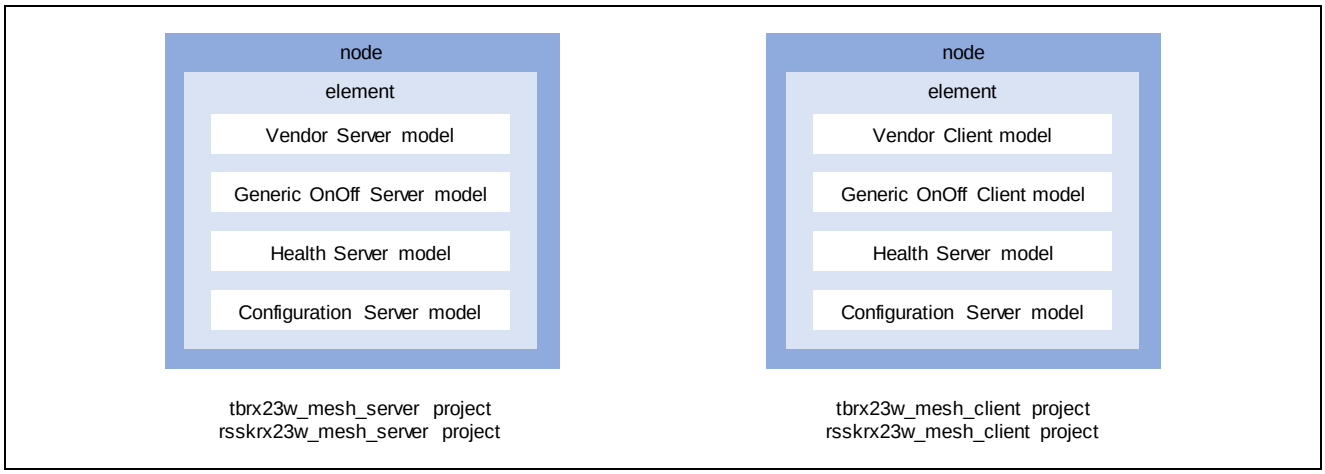

Figure 2-2 Model Composition of Mesh Sample Program

2.2.3.1 Configuration Model

Configuration model is the model to configure a node. Configuration Server has multiple Configuration states storing configurations of operation of node, element, and model. These states are operated by messages from Configuration Client.

Table 2-1 States of Configuration Model

Model Name	SIG Model ID (16bits)	State
Configuration Server	0x0000	Secure Network Beacon Composition Data Default TTL GATT Proxy Friend Relay Model Publication Subscription List NetKey List AppKey List Model to AppKey List Node Identity Key Refresh Phase Heartbeat Publish Heartbeat Subscription Network Transmit Relay Retransmit PollTimeout List
Configuration Client	0x0001	-

Table 2-2 Configuration Messages

State	Message Name	Opcode	Direction
Secure Network Beacon	Config Beacon Get	0x8009	Client → Server
	Config Beacon Set	0x800A	Client → Server
	Config Beacon Status	0x800B	Server → Client
Composition Data	Config Composition Data Get	0x8008	Client → Server
	Config Composition Data Status	0x02	Server → Client
Default TTL	Config Default TTL Get	0x800C	Client → Server
	Config Default TTL Set	0x800D	Client → Server
	Config Default TTL Status	0x800E	Server → Client
GATT Proxy	Config GATT Proxy Get	0x8012	Client → Server
	Config GATT Proxy Set	0x8013	Client → Server
	Config GATT Proxy Status	0x8014	Server → Client
Friend	Config Friend Get	0x800F	Client → Server
	Config Friend Set	0x8010	Client → Server
	Config Friend Status	0x8011	Server → Client
Relay Relay Retransmit	Config Relay Get	0x8026	Client → Server
	Config Relay Set	0x8027	Client → Server
	Config Relay Status	0x8028	Server → Client
Model Publication	Config Model Publication Get	0x8018	Client → Server
	Config Model Publication Set	0x03	Client → Server
	Config Model Publication Virtual Address Set	0x801A	Client → Server
	Config Model Publication Status	0x8019	Server → Client
Subscription List	Config Model Subscription Add	0x801B	Client → Server
	Config Model Subscription Virtual Address Add	0x8020	Client → Server
	Config Model Subscription Delete	0x801C	Client → Server
	Config Model Subscription Virtual Address Delete	0x8021	Client → Server
	Config Model Subscription Virtual Address Overwrite	0x8022	Client → Server

	Config Model Subscription Overwrite	0x801E	Client → Server
	Config Model Subscription Delete All	0x801D	Client → Server
	Config Model Subscription Status	0x801F	Server → Client
	Config SIG Model Subscription Get	0x8029	Client → Server
	Config SIG Model Subscription List	0x802A	Server → Client
	Config Vendor Model Subscription Get	0x802B	Client → Server
	Config Vendor Model Subscription List	0x802C	Server → Client
NetKey List	Config NetKey Add	0x8040	Client → Server
	Config NetKey Update	0x8045	Client → Server
	Config NetKey Delete	0x8041	Client → Server
	Config NetKey Status	0x8044	Server → Client
	Config NetKey Get	0x8042	Client → Server
	Config NetKey List	0x8043	Server → Client
AppKey List	Config AppKey Add	0x00	Client → Server
	Config AppKey Update	0x01	Client → Server
	Config AppKey Delete	0x8000	Client → Server
	Config AppKey Status	0x8003	Server → Client
	Config AppKey Get	0x8001	Client → Server
	Config AppKey List	0x8002	Server → Client
Model to AppKey List	Config Model App Bind	0x803D	Client → Server
	Config Model App Unbind	0x803F	Client → Server
	Config Model App Status	0x803E	Server → Client
	Config SIG Model App Get	0x804B	Client → Server
	Config SIG Model App List	0x804C	Server → Client
	Config Vendor Model App Get	0x804D	Client → Server
	Config Vendor Model App List	0x804E	Server → Client
Node Identity	Config Node Identity Get	0x8046	Client → Server
	Config Node Identity Set	0x8047	Client → Server
	Config Node Identity Status	0x8048	Server → Client
-	Config Node Reset	0x8049	Client → Server
	Config Node Reset Status	0x804A	Server → Client
Key Refresh Phase	Config Key Refresh Phase Get	0x8015	Client → Server
	Config Key Refresh Phase Set	0x8016	Client → Server
	Config Key Refresh Phase Status	0x8017	Server → Client
Heartbeat Publication	Config Heartbeat Publication Get	0x8038	Client → Server
	Config Heartbeat Publication Set	0x8039	Client → Server
	Config Heartbeat Publication Status	0x06	Server → Client
Heartbeat Subscription	Config Heartbeat Subscription Get	0x803A	Client → Server
	Config Heartbeat Subscription Set	0x803B	Client → Server
	Config Heartbeat Subscription Status	0x803C	Server → Client
Network Transmit	Config Network Transmit Get	0x8023	Client → Server
	Config Network Transmit Set	0x8024	Client → Server
	Config Network Transmit Status	0x8025	Server → Client
PollTimeout List	Config Low Power Node PollTimeout Get	0x802D	Client → Server
	Config Low Power Node PollTimeout Status	0x802E	Server → Client

Memory region for Configuration states is allocated in Mesh Stack. When receiving Configuration message, Mesh Stack updates values of the state automatically. Therefore, application does not have to handle them. Also, application can access values of the Configuration states by using Mesh Stack API.

2.2.3.2 Health Model

Health model is the model to monitor the physical condition of a node. Health Server has Fault states for storing physical fault condition of node. These states are updated when fault occurs. In addition, self-testing of a node can be performed by messages from Health Server.

Also, Health Server has Attention Timer state to activate a mechanism (e.g., LED blinking or noise making) to attract human's attraction.

Table 2-3 States of Health Model

Model Name	SIG Model ID (16bits)	State
Health Server	0x0002	Current Fault Registered Fault Health Period Attention Timer
Health Client	0x0003	-

Table 2-4 Health Messages

State	Message Name	Opcode	Direction
Current Fault	Health Current Status	0x04	Server → Client
Registered Fault	Health Fault Get	0x8031	Client → Server
	Health Fault Clear	0x802F	Client → Server
	Health Fault Clear Unacknowledged	0x8030	Client → Server
	Health Fault Status	0x05	Server → Client
	Health Fault Test	0x8032	Client → Server
	Health Fault Test Unacknowledged	0x8033	Client → Server
Health Period	Health Period Get	0x8034	Client → Server
	Health Period Set	0x8035	Client → Server
	Health Period Set Unacknowledged	0x8036	Client → Server
	Health Period Status	0x8037	Server → Client
Attention Timer	Health Attention Get	0x8004	Client → Server
	Health Attention Set	0x8005	Client → Server
	Health Attention Set Unacknowledged	0x8006	Client → Server
	Health Attention Status	0x8007	Server → Client

Memory region for Health states is allocated in Mesh Stack.

2.2.3.3 Generic OnOff Model

Generic OnOff Model is a model that is defined by Bluetooth SIG. Generic OnOff Server has a Generic OnOff state storing value of either On or Off. This state is operated by messages from Generic OnOff Client.

Table 2-5 State of Generic OnOff Model

Model Name	SIG Model ID (16bits)	State
Generic OnOff Server	0x1000	Generic OnOff (0x00: Off, 0x01: On)
Generic OnOff Client	0x1001	-

Table 2-6 Generic OnOff Messages

State	Message Name	Opcode	Direction
Generic OnOff	Generic OnOff Get	0x8201	Client → Server
	Generic OnOff Set	0x8202	Client → Server
	Generic OnOff Set Unacknowledged	0x8203	Client → Server
	Generic OnOff Status	0x8204	Server → Client

2.2.3.4 Vendor Model

User can define original Vendor Model. This page describes Vendor Model implemented in Mesh Sample Program.

Vendor Server has a Vendor state storing any variable-length data. This state is operated by messages from Vendor Client.

Table 2-7 State of Vendor Model

Model Name	Vendor Model ID (32bits)	State
Vendor Server	0x00010036 (default value)	Vendor state (any variable-length data)
Vendor Client	0x00020036 (default value)	-

Table 2-8 Vendor Messages

State	Message Name	Opcode	Direction
Vendor	Vendor Get	0xC10036 (default value)	Client → Server
	Vendor Set	0xC20036 (default value)	Client → Server
	Vendor Set Unacknowledged	0xC30036 (default value)	Client → Server
	Vendor OnOff Status	0xC40036 (default value)	Server → Client

2.3 Bluetooth Mesh Stack

Bluetooth Mesh Stack provides applications with many-to-many wireless communication features which is compliant with the Bluetooth Mesh Networking specifications. Library file of the Mesh Stack is included in the package, so you can use the Mesh features via Bluetooth Mesh Stack API.

Figure 2-3 shows the internal architecture of Bluetooth Mesh Stack.

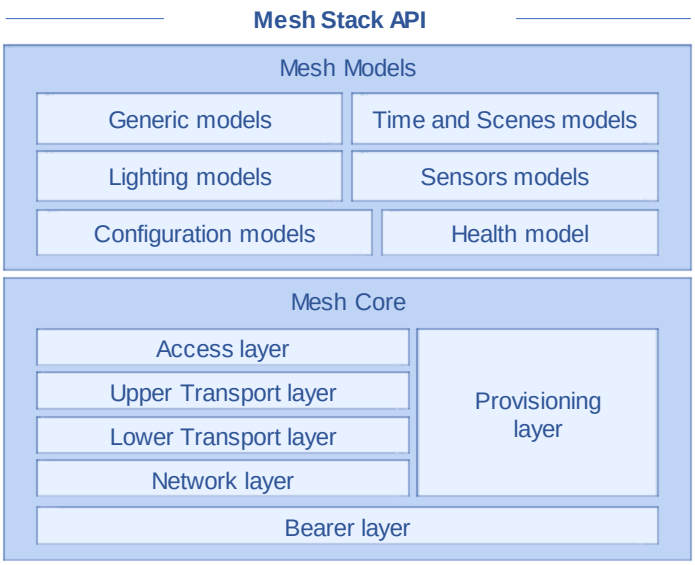

Figure 2-3 Internal Architecture of Bluetooth Mesh Stack

The Bluetooth Mesh Stack is composed of the following two blocks:

- **Mesh Core**

  Mesh Core block is composed of modules corresponding with each layer defined by Mesh Profile Specification and provides application with the features to perform Provisioning process and mesh networking operations. Regarding the Mesh Profile Specification, visit the Mesh Networking Specifications website of Bluetooth SIG and refer to Mesh Profile Specification document.

- **Mesh Models**

  Mesh Models block is composed of modules corresponding with each model defined by Mesh Model Specification and provides application with the features to support Mesh models that defines basic operations on a mesh network. Regarding the Mesh Model Specification, visit the Mesh Networking Specifications website of Bluetooth SIG and refer to Mesh Model Specification document.

Regarding the specification of Mesh Stack API, refer to the Bluetooth Mesh Stack API Manual "blemesh_api.chm" included in the Mesh FIT Module.

Mesh Stack consists of modules to implement protocol defined by Bluetooth Mesh Networking Specifications. Mesh Stack API has the following function prefixes corresponding to each module.

Mesh Application is required to call Mesh Stack API in accordance with scenario of application.

Table 2-9 Mesh Stack Functions

Module		Function Prefix
Mesh Model		
	Generic OnOff	MS_generic_onoff_*()
	Generic Level	MS_generic_level_*()
	Generic Default Transition Time	MS_generic_default_transition_time_*(),
	Generic Power OnOff	MS_generic_power_onoff_*()
	Generic Power Level	MS_generic_power_level_*()
	Generic Battery	MS_generic_battery_*()
	Generic Location	MS_generic_location_*()
	Generic Property	MS_generic_property_*()
	Sensor	MS_sensor_*()
	Time	MS_time_*()
	Scene	MS_scene_*()
	Scheduler	MS_scheduler_*()
	Light Lightness	MS_light_lightness_*()
	Light CTL	MS_light_ctl_*()
	Light HSL	MS_light_hsl_*()
	Light xyL	MS_light_xyl_*()
	Light LC	MS_light_lc_*()
	Configuration	MS_config_*()
	Health	MS_health_*()
Mesh Core		
	Access Layer	MS_access_*()
	Transport Layer	MS_trn_*()
	Lower Transport Layer	MS_ltrn_*()
	Network Layer	MS_net_*()
	Bearer Layer	MS_brr_*()
	Provisioning Layer	MS_prov_*()

2.4 Bluetooth Bearer

Bluetooth Bearer provides Mesh Stack and applications with wrapper functions of Bluetooth LE Stack. Source code files of Bluetooth Bearer are included in the package. Regarding the specification of Bluetooth Bearer API, refer to the Bluetooth Mesh Stack API Manual "blemesh_api.chm" included in the Mesh FIT Module.

Bluetooth LE Stack provides upper layers with wireless communication features which is compliant with the Bluetooth Low Energy specifications. Library file of Bluetooth LE Stack is included in the package. Regarding the specification of R_BLE API, refer to the R_BLE API Specification "r_ble_api_spec.chm" in the BLE FIT module.

Figure 2-4 shows the internal architecture of Bluetooth Bearer. Bearer functions for message transmission and reception are used by Mesh Stack. Bearer functions for connection control must be used by Mesh Application as necessary.

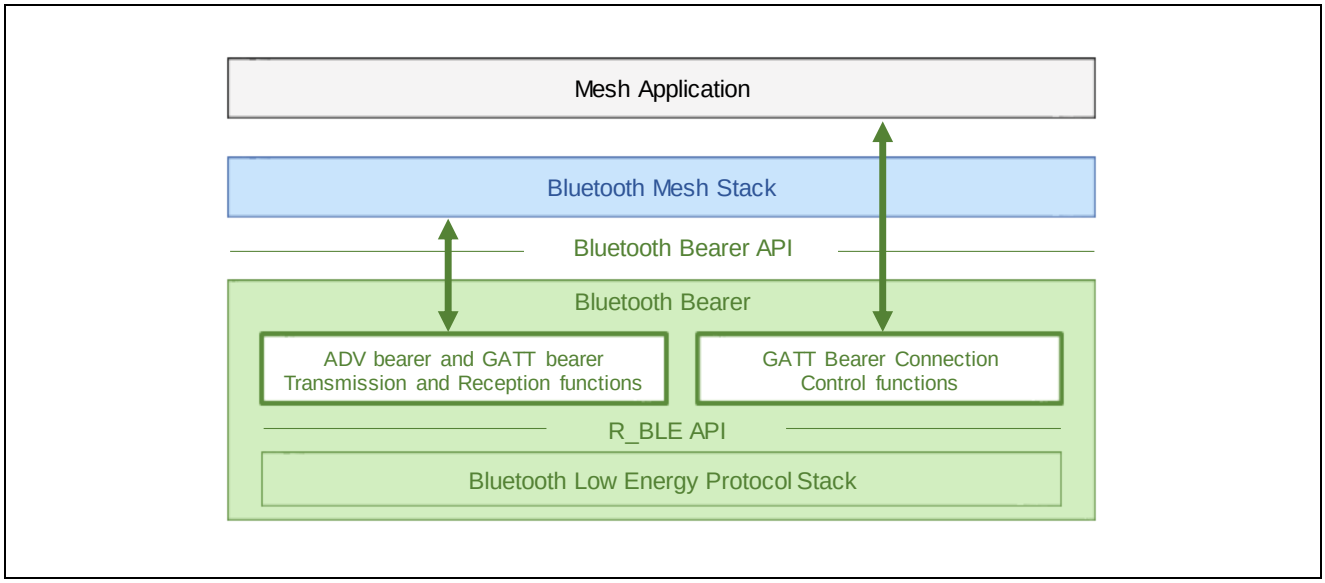

Figure 2-4 Bluetooth Bearer Operations

2.4.1 Bearer Functions for Message Transmission and Reception (blebrr.c)

Table 2-10 shows the bearer functions for message transmission and reception. The bearer functions provide the functionalities for ADV bearer mode control as well as message transmission and reception. The bearer functions are registered with Mesh Stack. Mesh Stack sends and receives Provisioning PDUs as well as mesh messages and controls ADV bearer mode by using bearer functions.

Table 2-10 Bearer Functions for Message Transmission and Reception

Function	Routine
blebrr_adv_send()	Send Data
(no registration required) NOTE	Handle Received Data
blebrr_adv_sleep()	Bearer Sleep
blebrr_adv_wakeup()	Bearer Wakeup

NOTE: A function to handle received data is registered by Mesh Stack automatically by MS_brr_add_bearer() called in R_MS_BRR_Setup().

2.4.2 Bearer Functions for Connection Control (blebrr_pl.c, blebrr_gatt.c)

Mesh Stack manages neither connection status nor GATT service. Therefore, to use GATT bearer, Mesh Application must control a connection and GATT services by using the bearer functions for connection control directly.

Table 2-11 shows the bearer functions for connection control. Those functions provide the functionalities for service discovery and notification permission as well as connection establishment and disconnection.

Table 2-11 Bearer Functions for Connection Control

Function	Routine	GATT Server (Peripheral)	GATT Client (Central)
R_MS_BRR_Register_GattIfaceCallback()	Register GATT Interface Callback	used	used
R_MS_BRR_Set_GattMode()	Set GATT Bearer Mode NOTE1	used	used
R_MS_BRR_Get_GattMode()	Get GATT Bearer Mode NOTE1	used	used
R_MS_BRR_Disconnect()	Disconnect	used	used
R_MS_BRR_Set_ScanRspData()	Set Scan Response Data	used	not used
R_MS_BRR_Scan_GattBearer()	Scan Connectable Device	not used	used
R_MS_BRR_Create_Connection()	Create Connection	not used	used
R_MS_BRR_Cancel_CreateConnection()	Cancel to Create Connection	not used	used
R_MS_BRR_Discover_Service()	Perform Service Discovery	not used	used
R_MS_BRR_Config_Notification()	Configure Mesh GATT Services Notification Permission NOTE2	not used	used
R_MS_BRR_Config_ServChanged()	Configure GATT Service Changed Indication Permission	not used	used

NOTE1: GATT Bearer Mode is either Provisioning Mode or Proxy Mode.

NOTE2: GATT Server configures MTU size to Mesh Stack when Notification is enabled. When changing MTU size, GATT Client must perform MTU Exchange procedure before enabling Notification.

Regarding how to change MTU size, refer to Section 8.4 in "RX23W Group Bluetooth Low Energy Application Developer's Guide"(R01AN5504).

2.4.3 Mesh GATT Services (gatt_db.c)

Mesh GATT Services are used for mesh message transmission and reception over GATT bearer. Composition of the Mesh GATT Services are listed in Table 2-12. Mesh Provisioning Service is used for Provisioning over PB-GATT bearer, and Mesh Proxy Service is used for Proxy connection after Provisioning. Which one of Mesh GATT Services are exposed is switched by R_MS_BRR_Set_GattMode().

Table 2-12 Composition of the Mesh GATT Services

Service (UUID)	Characteristic (UUID)	Property	Value
Mesh Provisioning Service (0x1827)	Mesh Provisioning Data In Characteristic (0x2ADB)	Write Without Response	Provisioning PDU from a Provisioning Client to a Provisioning Server
	Mesh Provisioning Data Out Characteristic (0x2ADC)	Notify	Provisioning PDU from a Provisioning Server to a Provisioning Client.
Mesh Proxy Service (0x1828)	Mesh Proxy Data In Characteristic (0x2ADD)	Write Without Response	Proxy PDU message containing Network PDU, mesh beacons, or proxy configuration from a Proxy Client to a Proxy Server
	Mesh Proxy Data Out Characteristic (0x2ADE)	Notify	Proxy PDU message containing Network PDU, mesh beacon, or proxy configuration from a Proxy Server to a Proxy Client.

GATT Database that defines Mesh Proxy Service and other services is implemented in "gatt_db.c" of Bluetooth Bearer.

2.4.4 ADV Bearer Operation

When Mesh Application calls R_MS_BRR_Setup(), Bluetooth Bearer registers message transmission and reception functions for ADV Bearer with Mesh Stack and starts Scan operation.

Advertising packets received by Bluetooth LE Stack are notified to Mesh Stack. Also, Bluetooth LE Stack transmits Advertising packets when Mesh Stack calls the message transmission function.

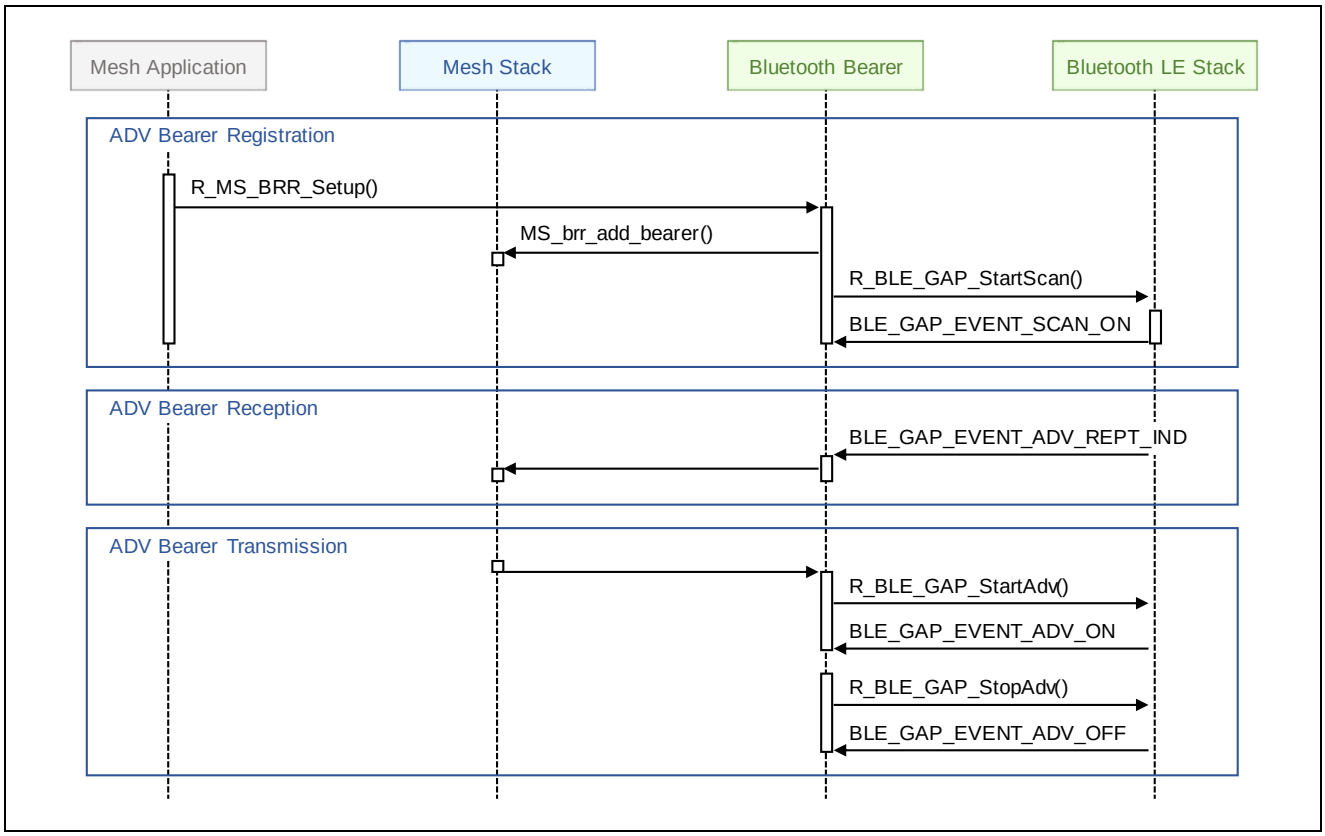

Figure 2-5 ADV Bearer Operation

2.4.5 GATT Bearer Operation

When a connection is established and enabling Notification completes, Bluetooth Bearer registers message transmission and reception functions for GATT Bearer with Mesh Stack.

In the case that node works as a GATT Server, Bluetooth LE Stack transmits message by Notification when Mesh Stack calls the message transmission function. Also, message transmitted by Write Without Response is notified to Mesh Stack.

In the case that node works as a GATT Client, Bluetooth LE Stack transmits message by Write Without Response when Mesh Stack calls the message transmission function. Also, message transmitted by Notification is notified to Mesh Stack.

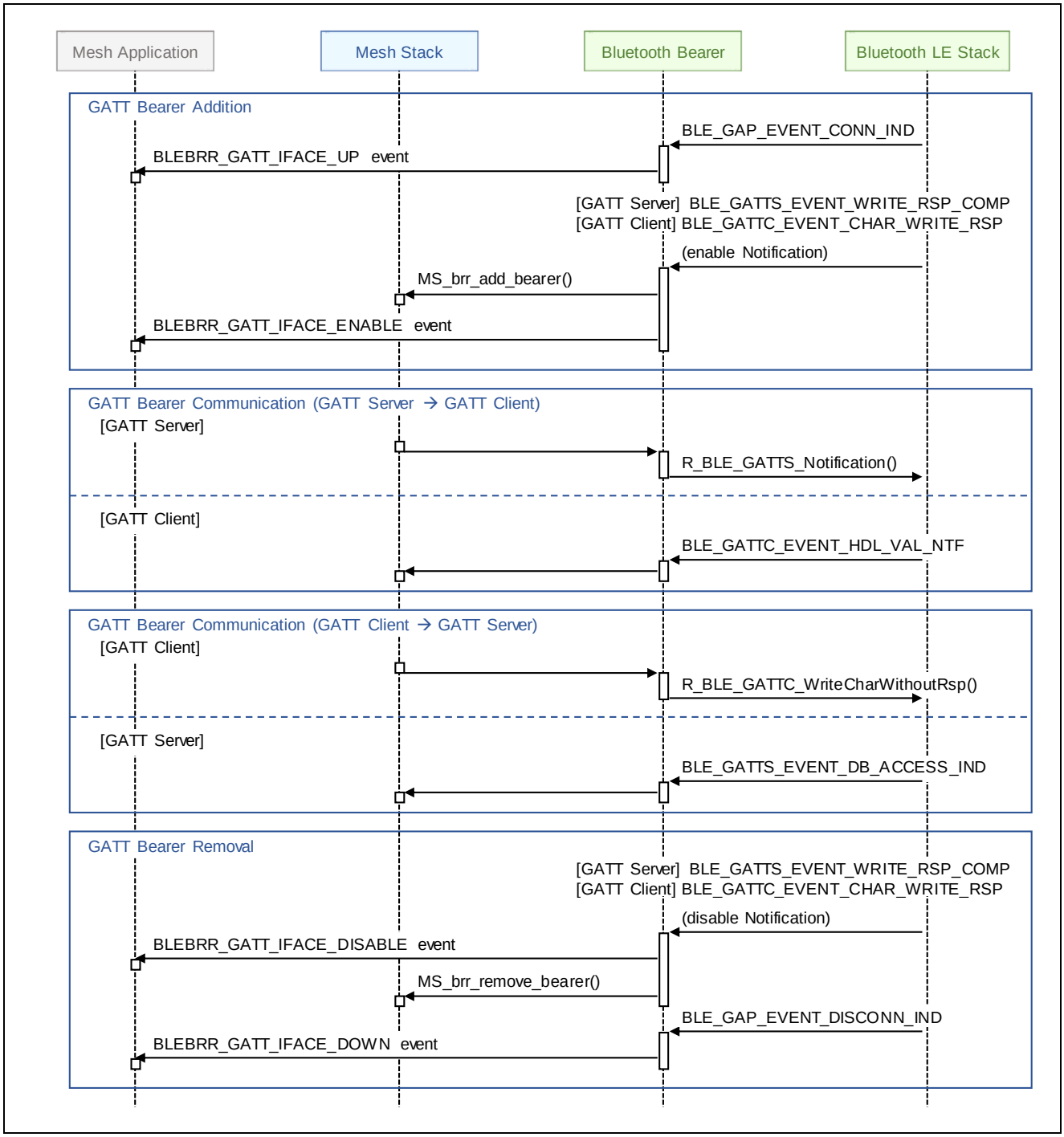

Figure 2-6 GATT Bearer Operation

2.5 MCU Peripheral Functions

Mesh Sample Program uses some RX23W peripheral functions listed in Table 2-13.

Table 2-13 RX23W Peripheral Functions used

RX23W Peripherals	Peripheral Driver	Software using Peripherals
I/O Ports – P15, PB0, and PC0: when Target Board is used – P30, P31, P42, and P43: when RSSK is used	GPIO FIT Module (R01AN1721)	Mesh Sample Program
Serial Communication Interface (SCI) – SCI8	SCI FIT Module (R01AN1815)	Mesh Sample Program
Compare Match Timer (CMT) – CMT2 and CMT3: Bluetooth LE Stack use exclusively – CMT0 or CMT1: Mesh Sample Program and Bluetooth Bearer share	CMT FIT Module (R01AN1856)	Mesh Sample Program Mesh Stack Bluetooth Bearer Bluetooth LE Stack
Low Power Consumption Function (LPC)	LPC FIT Module (R01AN2769)	Mesh Sample Program Bluetooth LE Stack
E2 Data Flash memory (FLASH) – Block 1 to 5	FLASH FIT Module (R01AN2184)	Mesh Stack
8-bit Timer (TMR) – TMR2 and TMR3	driver is included in Mesh FIT Module (R01AN4930)	Mesh Stack

- **I/O Ports**

Mesh Sample Program uses GPIO FIT Module to use General Purpose I/O Port (GPIO) for the following processing.

- LED Control on development board
- Switch Pushing Detection on development board

- **Serial Communication Interface (SCI)**

Mesh Sample Program uses SCI FIT Module to output and input console over UART.

- **Compare Match Timer (CMT)**

Software Timer (R_BLE_TIMER) to share one channel of CMT for multiple processing is included in BLE FIT Module. R_BLE_TIMER uses one channel of CMT exclusively by using CMT FIT Module. Also, Bluetooth LE Stack of BLE FIT Module uses CMT2 and CMT exclusively.

Bluetooth Bearer uses R_BLE_TIMER for the following processing.

- Advertising Transmission Control for ADV Bearer

Mesh Sample Program uses R_BLE_TIMER for the following processing.

- LED Blinking on development board
- Avoiding Chattering of Switch on development board
- MCU Reset Delay after receiving Config Node Reset
- Completion of IV Update Procedure

- **Low Power Consumption (LPC)**

Mesh Sample Program uses LPC FIT Module to enable Low Power Consumption function of MCU.

- **Flash memory (FLASH)**

Data Flash driver to use Data Flash memory is implemented in "mesh_dataflash.c". This driver accesses Data Flash memory by using FLASH FIT Module. Flash memory region used by this driver can be configured by the MESH_CFG_DATA_FLASH_BLOCK_ID macro and the MESH_CFG_DATA_FLASH_BLOCK_NUM macro.

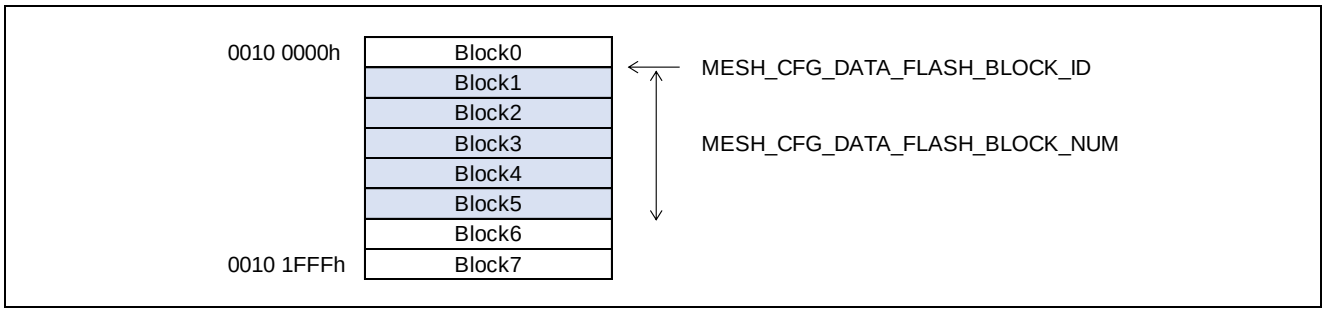

Figure 2-7 Data Flash memory region used

Mesh Stack stores the following information to Data Flash memory.

- Information exchanged during Provisioning
  - mesh addresses
  - encryption keys
- Information exchanged during Configuration
  - model composition
  - model configuration
- IV index and associated state
- Sequence Number

This information will be changed very rarely except for Sequence Number. The sequence number is incremented for each new network message transmission. If it is written for each increment, the flash memory reaches the write cycle limit in a short span of time.

Thus, to reduce frequency of writing into the flash memory, sequence numbers are handled as block and written only when the sequence number reaches next block. The block size that means distance between the sequence numbers can be configured by the MESH_CFG_NET_SEQ_NUMBER_BLOCK_SIZE macro and default block size is 2048.

- **8-bit Timer (TMR)**

System Time driver to use 8-bit Timer is implemented in "mesh_systemtime.c". This driver uses two channels of 8-bit Timer and generates 32bit length system time in units of 1msec.

Mesh Stack monitors 96 hours that is minimum duration of IV Update Procedure by using the system time.

2.6 Mesh Sample Program Configuration

Mesh Sample Program has multiple compilation switches to configure its operation. Compilation switches are implemented in "mesh_appl.h".

mesh_appl.h

```
/**
 * Monitor SEQ of Incoming and Outgoing message.
 * If SEQ is greater than or equal to threshold, initiating IV Update procedure.
 */
#define IV_UPDATE_INITIATION_EN         (1)

/**
 * Low Power Feature
 */
#define LOW_POWER_FEATURE_EN            (0)

/**
 * Monitoring Mesh Layer
 * Combination of the following macros can be set like "(MS_MONITOR_ACCESS_PDU |
MS_MONITOR_NET_PDU)".
 *   - MS_MONITOR_ACCESS_PDU
 *   - MS_MONITOR_TRANS_PDU
 *   - MS_MONITOR_LTRANS_PDU
 *   - MS_MONITOR_NET_PDU
 *   - MS_MONITOR_NET_SNB
 * To specify all layer, MS_MONITOR_ALL macro can be set.
 */
#define CONSOLE_MONITOR_CFG             (MS_MONITOR_NONE)

/** Logging Console using SCI (Serial Communication Interface) */
#define CONSOLE_OUT_EN                  (1)

/** ANSI escape sequence - CSI (Control Sequence Introducer) */
#define ANSI_CSI_EN                     (1)

/** Monitoring CPU Usage */
#define CPU_USAGE_EN                    (0)
```

- **Enabling IV Update Initiation Processing**

IV Update Initiation processing is enabled by setting the IV_UPDATE_INITIATION_EN macro to (1). This processing monitors sequence number of incoming and outgoing message and initiates IV Update procedure when the sequence number is greater than or equal to threshold. It prevents sequence number of own node or other nodes from exhausting.

Configuration Macro	Configuration Value	Description
IV_UPDATE_INITIATION_EN	0	Disable IV Update Initiation processing
	1	Enable IV Update Initiation processing

- **Enabling Low Power Feature**

Low Power feature is enabled by setting the LOW_POWER_FEATURE_EN macro to (1). After Provisioning, Mesh Sample Program establishes a Friendship with Friend node and works as a Low Power node.

Configuration Macro	Configuration Value	Description
LOW_POWER_FEATURE_EN	0	Disable Transition to Low Power Node
	1	Enable Transition to Low Power Node

- **Mesh Monitoring Configuration**

Mesh Monitoring functionality is enabled by setting Monitor Configuration macro into the CONSOLE_MONITOR_CFG macro. Logs of messages and beacons that are transmitted or received by each layer of Mesh Stack, so it is possible to analyze mesh network communication in Mesh Application development.

Configuration Macro	Configuration Value	Description
CONSOLE_MONITOR_CFG	MS_MONITOR_ACCESS_PDU	Access PDU
	MS_MONITOR_TRANS_PDU	Transport PDU
	MS_MONITOR_LTRANS_PDU	Lower Transport PDU
	MS_MONITOR_NET_PDU	Network PDU and Secure Network Beacon
	MS_MONITOR_GENERIC_LOG	Mesh Stack Internal Miscellaneous Event

- **Console Output Configuration**

Console Output is enabled by setting the CONSOLE_OUT_EN macro to (1). It is possible to trace API called by Mesh Sample Program and events returned by Mesh Stack.

Configuration Macro	Configuration Value	Description
CONSOLE_OUT_EN	0	Disable Console Log Output
	1	Enable Console Log Output

- **ANSI CSI Console Output Configuration**

Output ANSI CSI (Control Sequence Introducer) to console is enabled by setting the ANSI_CSI_EN macro to (1). Mesh Sample Program colors some log. In the case that serial terminal emulator you use does not support ANSI CSI, set the ANSI_CSI_EN macro to (0).

Configuration Macro	Configuration Value	Description
ANSI_CSI_EN	0	Disable ANSI CSI Output to Console Log
	1	Enable ANSI CSI Output to Console Log

- **CPU Usage Measurement Configuration**

CPU Usage Measurement is enabled by setting the CPU_USAGE_EN macro to (1). Mesh Sample Program measures the time of CPU RUN state and CPU SLEEP state and outputs CPU usage log to console.

Configuration Macro	Configuration Value	Description
CPU_USAGE_EN	0	Disable CPU Usage Measurement
	1	Enable CPU Usage Measurement

2.7 Bluetooth Bearer Configuration

Configurations of Bluetooth Bearer included in Mesh FIT Module are shown below.

blebrr.h

```
/** Enable GATT Bearer Client Role */
/*  ROM/RAM used can be reduced by disabling GATT Client functionalities */
#define BLEBRR_GATT_CLIENT                      (1)

/** Specify Device Address Type
  * Either Public Address or Static Random Address can be set by the macro below.
  *  - BLE_GAP_ADDR_PUBLIC
  *  - BLE_GAP_ADDR_RAND
  * Device Address is obtained from BLE Protocol Stack via Vendor Specific API.
  */
#define BLEBRR_VS_ADDR_TYPE                     (BLE_GAP_ADDR_RAND)
```

- **Enabling GATT Client**

GATT Client functionality for GATT bearer is enabled by setting the BLEBRR_GATT_CLIENT macro to (1).

Configuration Macro	Configuration Value	Description
BLEBRR_GATT_CLIENT	0	Disable GATT Client Operation of GATT Bearer
	1	Enable GATT Client Operation of GATT Bearer

- **Device Address Type Configuration**

Device Address Type used by Bluetooth Bearer can be configured by setting Device Address Type macro into the BLEBRR_VS_ADDR_TYPE macro.

Configuration Macro	Configuration Value	Description
BLEBRR_VS_ADDR_TYPE	BLE_GAP_ADDR_PUBLIC	Public Device Address
	BLE_GAP_ADDR_RAND	Random Device Address

blebrr.c

```
#define BLEBRR_QUEUE_SIZE               64
#define BLEBRR_ADV_TIMEOUT              4
#define BLEBRR_ADVREPEAT_COUNT          3
#define BLEBRR_ADVREPEAT_RAND_DELAY     10
```

- **ADV Bearer Transmission Configuration**

Configuration macros for ADV bearer transmission are defined as follows:

Configuration Macro	Configuration Value	Description
BLEBRR_QUEUE_SIZE	4 or more	Transmission Queue Size
BLEBRR_ADVREPEAT_RAND_DELAY	1 or more	Transmission Randomized Delay in units of 1msec

blebrr_pl.c

```
#define BLEBRR_CON_ADVINTMIN                0x20
#define BLEBRR_CON_ADVINTMAX                0x20
#define BLEBRR_CON_ADVTYPE                  BLE_GAP_LEGACY_PROP_ADV_IND
#define BLEBRR_CON_ADVCHMAP                 BLE_GAP_ADV_CH_ALL
#define BLEBRR_CON_ADVFILTERPOLICY          BLE_GAP_ADV_ALLOW_SCAN_ANY_CONN_ANY

#define BLEBRR_SCAN_INTERVAL                0x0060
#define BLEBRR_SCAN_WINDOW                  0x0060

#define BLEBRR_INIT_SCANINTERVAL            0x0060
#define BLEBRR_INIT_SCANWINDOW              0x0060

#define BLEBRR_CONN_INTERVAL_MIN            0x0040
#define BLEBRR_CONN_INTERVAL_MAX            0x0040
#define BLEBRR_CONN_LATENCY                 0x0000
#define BLEBRR_CONN_SUPERVISION_TO          0x03BB
```

- **GATT Bearer Connectable Advertising Configuration**

Configuration macros for Connectable Advertising for GATT bearer are defined as follows:

Configuration Macro	Configuration Value	Description
BLEBRR_CON_ADVINTMIN	0x20 to 0xFFFFFF	Minimum Advertising Interval (in units of 0.625msec)
BLEBRR_CON_ADVINTMAX	0x20 to 0xFFFFFF	Maximum Advertising Interval (in units of 0.625msec)
BLEBRR_CON_ADVTYPE	BLE_GAP_LEGACY_PROP_ADV_IND	Advertising Type: Connectable and Scannable Undirected Legacy Advertising
BLEBRR_CON_ADVCHMAP	BLE_GAP_ADV_CH_37	Advertising Channel 37ch
	BLE_GAP_ADV_CH_38	Advertising Channel 38ch
	BLE_GAP_ADV_CH_39	Advertising Channel 39ch
	BLE_GAP_ADV_CH_ALL	All Advertising Channels
BLEBRR_CON_ADVFILTERPOLICY	BLE_GAP_ADV_ALLOW_SCAN_ANY_CONN_ANY	Advertising Filter Policy: Process Scan Requests and Connection Requests from All Devices

- **ADV Bearer Scan Configuration**

Configuration macros for ADV bearer Scan are defined as follows:

NOTE: Scan Window Size should be equal to Scan Interval to avoid missing incoming Advertising packets.

- BLEBRR_SCAN_INTERVAL : Scan Interval (in units of 0.625msec)
- BLEBRR_SCAN_WINDOW : Scan Window Size (in units of 0.625msec)

Configuration Macro	Configuration Value	Description
BLEBRR_SCAN_INTERVAL	0x0004 to 0xFFFF	Scan Interval (in units of 0.625msec)
BLEBRR_SCAN_WINDOW	0x0004 to 0xFFFF	Scan Window Size (in units of 0.625msec)

- **GATT Bearer GATT Client Connection Configuration**

Configuration macros for GATT Client Connection of GATT Bearer are defined as follows:

Configuration Macro	Configuration Value	Description
BLEBRR_INIT_SCANINTERVAL	0x0004 to 0xFFFF	Scan Interval for Initiating (in units of 0.625msec)
BLEBRR_INIT_SCANWINDOW	0x0004 to 0xFFFF	Scan Windows Size for Initiating (in units of 0.625msec)
BLEBRR_CONN_INTERVAL_MIN	0x0006 to 0x0C80	Minimum Connection Interval (in units of 1.25msec)
BLEBRR_CONN_INTERVAL_MAX	0x0006 to 0x0C80	Maximum Connection Interval (in units of 1.25msec)
BLEBRR_CONN_LATENCY	0x0000 to 0x01F3	Slave Latency
BLEBRR_CONN_SUPERVISION_TO	0x000A to 0x0C80	Supervision Timeout (in units of 10msec)

2.8 Mesh Driver Configuration

Configurations of Mesh Driver included in Mesh FIT Module are shown below.

mesh_dataflash.h

```
#define DATAFLASH_EN    (1)
```

- **Enabling Data Flash Driver**

Data Flash driver is enabled and then driver functions are registered with Mesh Stack by setting the DATAFLASH_EN macro to (1).

Configuration Macro	Configuration Value	Description
DATAFLASH_EN	0	Disable Data Flash Access
	1	Enable Data Flash Access

mesh_systemtime.h

```
#define SYSTEMTIME_EN                   (1)
#define SYSTEMTIME_STRING_EN            (1)
```

- **Enabling System Time Driver**

System Time Driver for generating 32bit system time is enabled and then driver function is registered with Mesh Stack by setting the SYSTEMTIME_EN macro to (1).

Configuration Macro	Configuration Value	Description
SYSTEMTIME_EN	0	Disable System Time Generation
	1	Enable System Time Generation

- **System Time String Configuration**

Function to generate string of 32bit system time is enabled by setting the SYSTEMTIME_STRING_EN macro to (1).

Configuration Macro	Configuration Value	Description
SYSTEMTIME_STRING_EN	0	Disable String Output of System Time
	1	Enable String Output of System Time

3. Application Development

This chapter describes how to develop an application using Bluetooth Mesh Stack while referring to the implementation of Mesh Sample Program. Figure 3-1 shows the sequence chart of Mesh Sample Program.

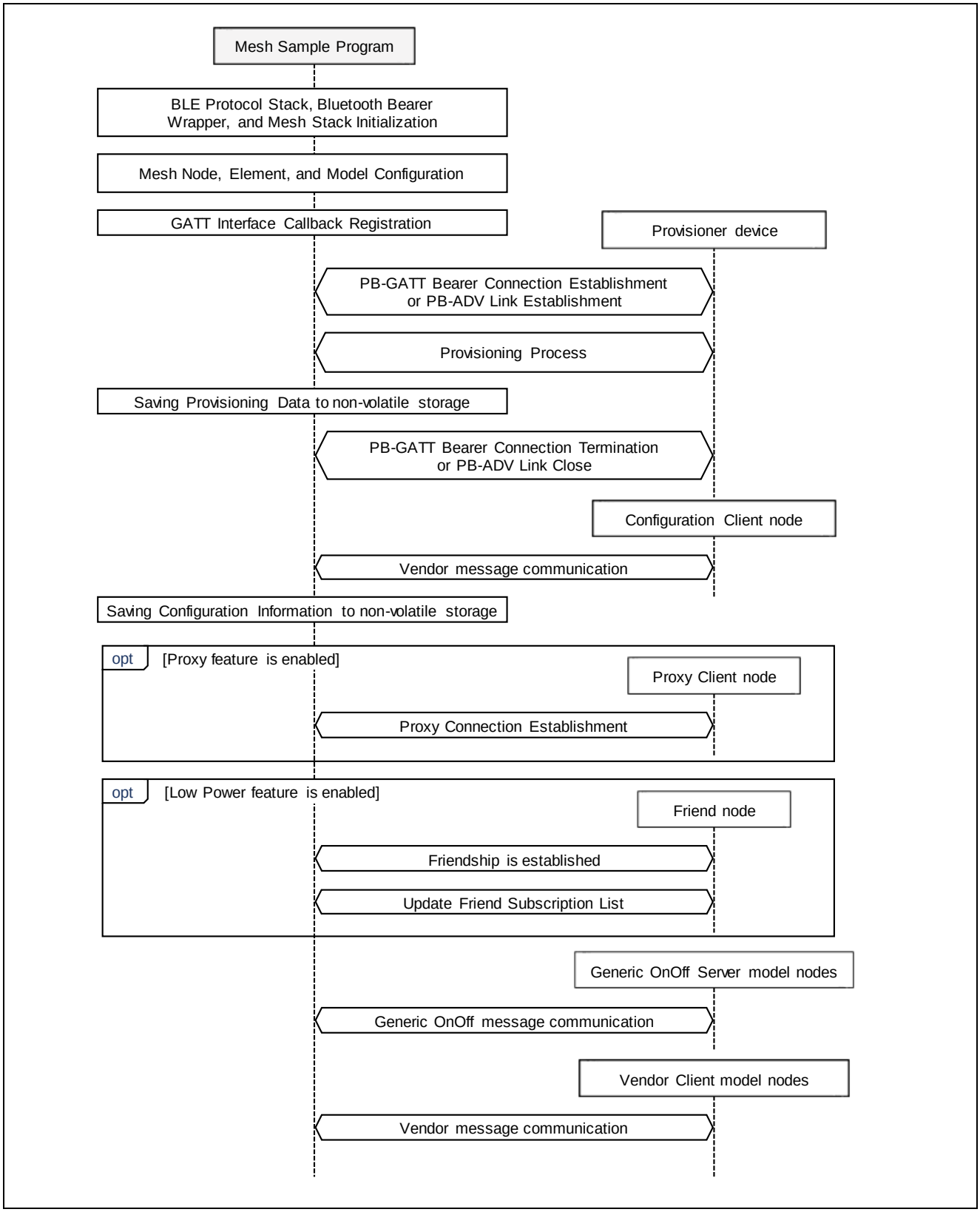

Figure 3-1 Sequence Chart of Mesh Sample Program

3.1 Main Routine

Mesh Stack works on Bluetooth LE Stack. Therefore, application must initialize Bluetooth LE Stack and Bluetooth Bearer before initializing Mesh Stack.

Each processing of Bluetooth LE Stack is performed by Bluetooth LE Stack Scheduler. Therefore, application must continue to execute R_BLE_Execute() that is scheduler function at main loop after initializing Bluetooth LE Stack.

Main routine of Mesh Sample Program is shown as below.

- **Main Routine (main.c)**

Initialize Bluetooth LE Stack and Bluetooth Bearer by R_BLE_Open() and R_MS_BRR_Init(). Initialize other FIT modules too as necessary. Execute Bluetooth LE Stack Scheduler, R_BLE_Execute(), iteratively in a main loop.

Initialization processing of Bluetooth Bearer is performed by Bluetooth LE Stack Scheduler and completion of the initialization is notified by a callback function.

```
int main(void)
{
    /* Initialize BLE Protocol Stack */
    R_BLE_Open();

    /* Initialize the Low Power Control function */
    R_BLE_LPC_Init();

    /* Initialize Timer */
    R_BLE_TIMER_Init();

    /* Initialize underlying BLE Protocol Stack to use as a Mesh Bearer */
    R_MS_BRR_Init(blebrr_init_cb);

    /* main loop */
    while (1)
    {
        /* Process Event */
        R_BLE_Execute();
    }
}
```

- **Callback Function of Bluetooth Bearer Initialization Completion**

Implement a callback function to receive a notification of completion of Bluetooth Bearer initialization. In this callback function, initialize resources for Mesh Stack and initialize Mesh Stack by MS_init() and register Bluetooth Bearer with Mesh Stack by R_MS_BRR_Setup(). After initializing these, start mesh application.

```
static void blebrr_init_cb(st_ble_dev_addr_t * own_addr)
{
    MS_CONFIG config;

    /* Initialize Mesh Resources */
    mesh_section_init();
    mesh_mempool_init();
    mesh_storage_init();
    #if SYSTEMTIME_EN
    mesh_systemtime_init();
```

```
    #endif /* SYSTEMTIME_EN */

    /* Initialize Mesh Stack */
    MESH_MS_CONFIG(config);
    MS_init(&config);

    /* Registers ADV Bearer with Mesh Stack and Start Scan */
    R_MS_BRR_Setup();

    /* Start Mesh Application */
    mesh_model_config(&gs_mesh_model_callbacks);
    mesh_core_setup();
}
```

3.2 Node Composition

Application must configure node composition such as elements and models. Its composition depends on each scenario that application should carry out; it means what and how application will control.

Configuring Node composition of Mesh Sample Program is shown as below.

- **Node and Element (mesh_model.c)**

Create a Node and Register Elements by MS_access_create_node() and MS_access_register_element(). The necessary number of elements differs from each application scenario. Element handle returned by Mesh Stack after registering an Element is used for adding Mesh Models.

```
API_RESULT mesh_model_config(const mesh_model_callbacks_t * callbacks)
{
    API_RESULT retval;
    MS_ACCESS_NODE_ID      node_id;
    MS_ACCESS_ELEMENT_DESC element_desc;

    /* Create Node */
    retval = MS_access_create_node(&node_id);

    /* Register Element */
    if (API_SUCCESS == retval)
    {
        element_desc.loc = ELEMENT_DESC_LOCATION;
        retval = MS_access_register_element
                    (
                        node_id,
                        &element_desc,
                        &gs_element_handle
                    );
    }

    return retval;
}
```

3.3 Provisioning

This section shows how to implement for working as a Provisioning Server.

3.3.1 Provisioning Server

To join a network and communicate with other nodes, application must perform Provisioning as a Provisioning Server and must receive a Provisioning Data from Provisioning Client.

Mesh Sample Program works as Provisioning Server. Provisioning processing of Mesh Sample Program is shown as below.

- **Registration of Provisioning Capabilities and Provisioning Callback Function (mesh_core.c)**

Register Provisioning Capabilities such as Authentication method as well as Provisioning Callback Function with Mesh Stack by MS_prov_register().

```
API_RESULT mesh_core_setup(void)
{
    /* Register Provisioning capabilities */
    retval = MS_prov_register
                (
                    &gs_prov_capabilities, mesh_core_prov_cb
                );

    return retval;
}
```

- **Start of Provisioning (mesh_core.c)**

Start transmission of Unprovisioned Device Beacon and connectable advertising by MS_prov_setup() and MS_prov_bind().

```
static API_RESULT mesh_core_prov_setup(UCHAR brr)
{
    if (PROV_BRR_GATT & brr)
    {
        R_MS_BRR_Set_GattMode(BLEBRR_GATT_PROV_MODE);
    }

    gs_prov_role = PROV_ROLE_DEVICE;
    retval = MS_prov_setup
                (
                    brr,
                    PROV_ROLE_DEVICE,
                    &gs_prov_device,
                    CORE_PROV_SETUP_TIMEOUT_SECS
                );

    if (API_SUCCESS == retval)
    {
        if (PROV_BRR_ADV & brr)
        {
            retval = MS_prov_bind
                        (
                            PROV_BRR_ADV,
                            &gs_prov_device,
                            CORE_PROV_DEVICE_ATTENTION_TIMEOUT,
```

```
                            &gs_prov_handle
                        );
        }
    }

    return retval;
}
```

- **Provisioning Callback Function (mesh_core.c)**

Implement a callback function to receive Provisioning events. Provisioning Data provided by a Provisioning Client is required to be registered with Mesh Stack by MS_access_cm_set_prov_data().

```
static API_RESULT mesh_core_prov_cb
                (
                    PROV_HANDLE * phandle,
                    UCHAR         event_type,
                    API_RESULT    event_result,
                    void        * event_data,
                    UINT16        event_datalen
                )
{
    PROV_DATA_S * rdata;

    switch (event_type)
    {
        case PROV_EVT_PROVISIONING_SETUP:
            /**
             * This event indicates that Provisioning Invite is received.
             */
            break;

        case PROV_EVT_PROVDATA_INFO:
            rdata = (PROV_DATA_S *)event_data;

            /* Provide Provisioning Data to Access Layer */
            retval = MS_access_cm_set_prov_data(rdata);
            break;

        case PROV_EVT_PROVISIONING_COMPLETE:
            /**
             * This event indicates Provisioning is complete.
             */
            break;
    }

    return API_SUCCESS;
}
```

3.3.2 Provisioning Sequence

(1) Provisioning Setup

This sample program supports both PB-ADV bearer and PB-GATT bearer and transmits Unprovisioned Device beacon by PB-ADV bearer and connectable advertising for PB-GATT bearer alternately.

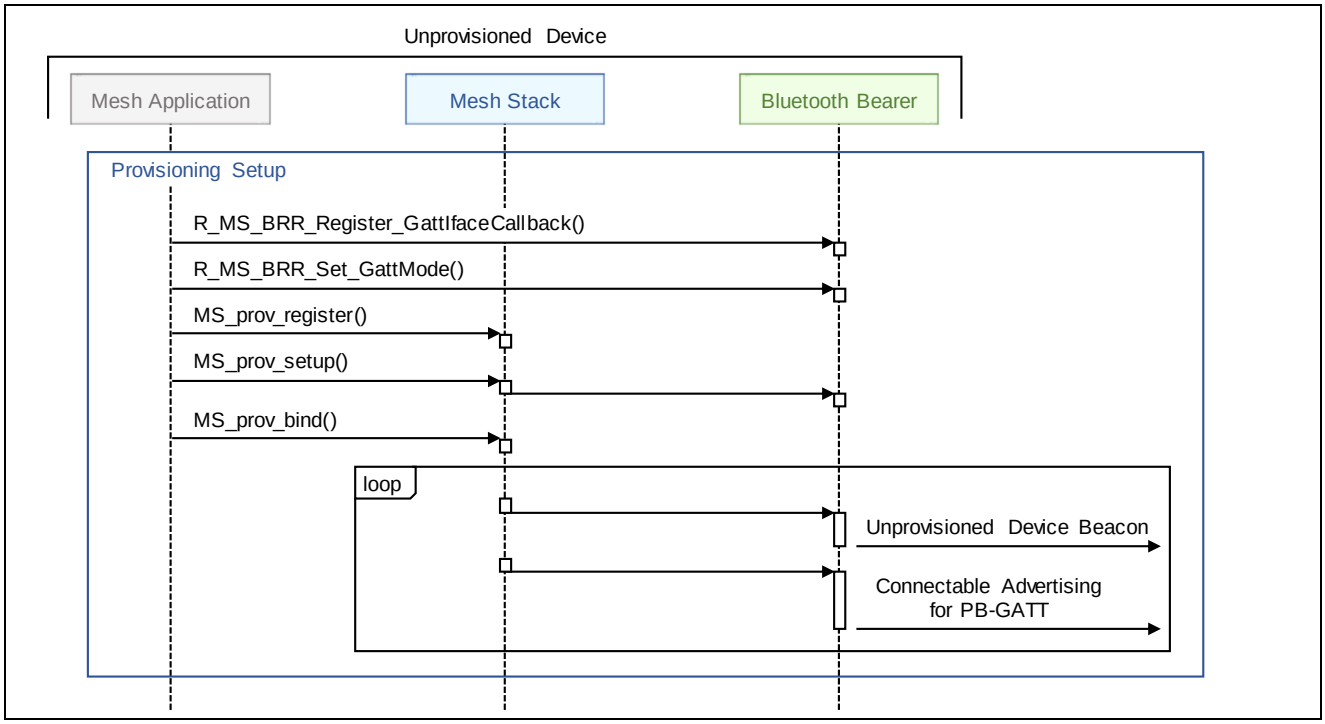

Figure 3-2 Provisioning Setup

(2) Session Establishment over PB-ADV

To perform Provisioning Process over PB-ADV, Provisioning Server establishes a session with Provisioning Client. Also, Provisioning Server closes a session after Provisioning Process.

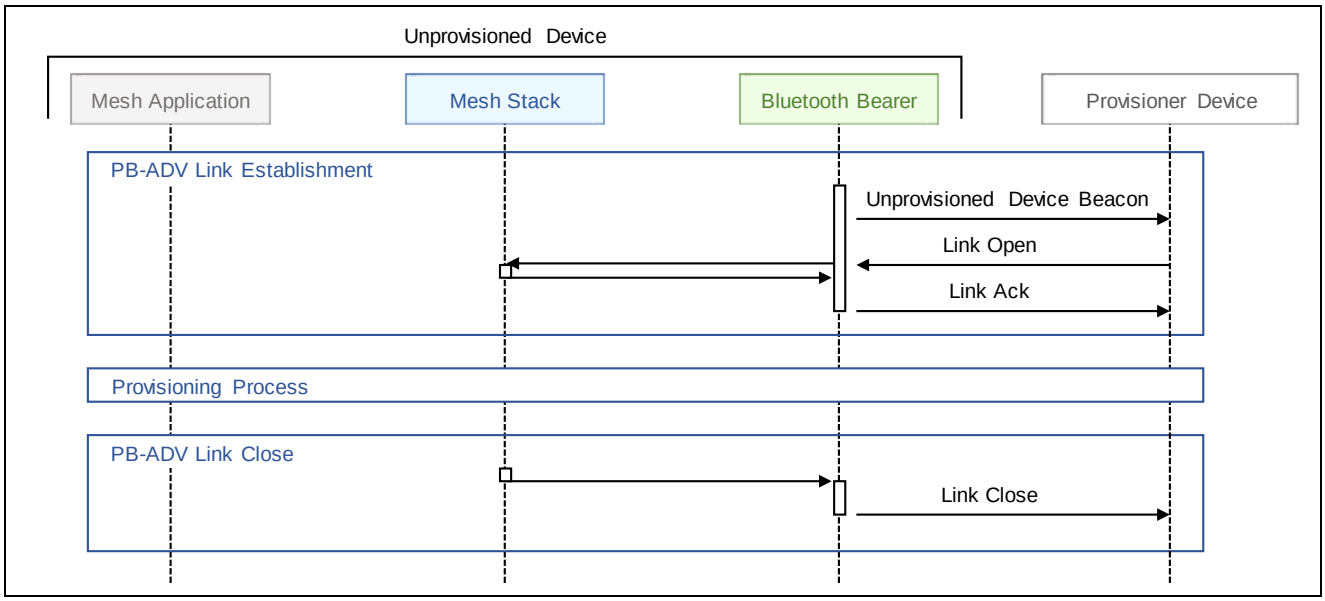

Figure 3-3 Session Establishment over PB-ADV

(3) Connection Establishment over PB-GATT

To perform Provisioning Process over PB-GATT, Provisioning Client establishes a connection with Provisioning Server and enables notification of the Mesh Provisioning Service. Also, Provisioning Client terminates the connection after Provisioning Process.

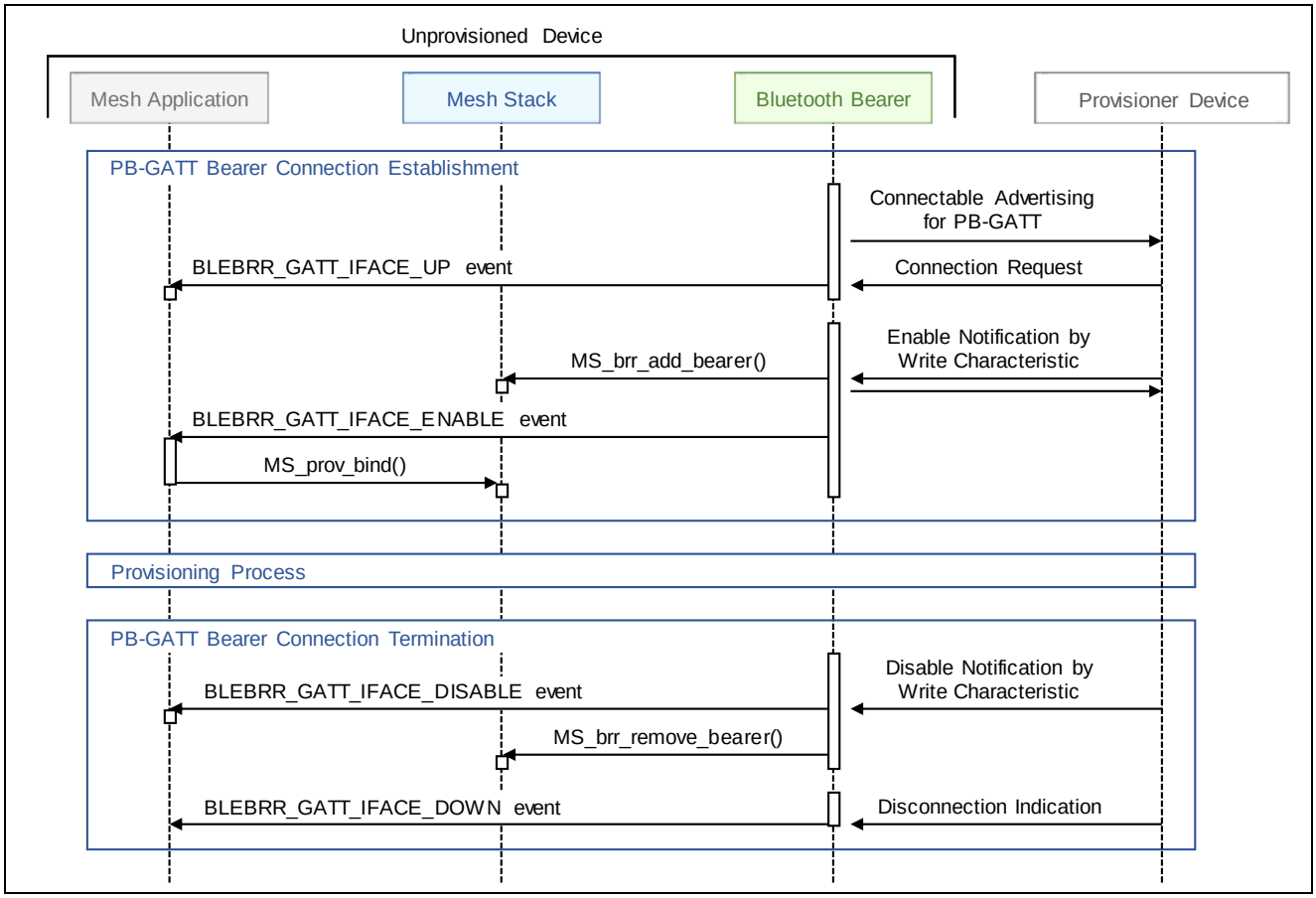

Figure 3-4 Connection Establishment over PB-GATT

(4) Provisioning Process

After establishing a session or a connection over Provisioning Bearer, Provisioning Process from Invitation to Distribution of provisioning data is performed and Provisioning PDUs are exchanged. The same process is performed over either PB-ADV or PB-GATT during Provisioning Process.

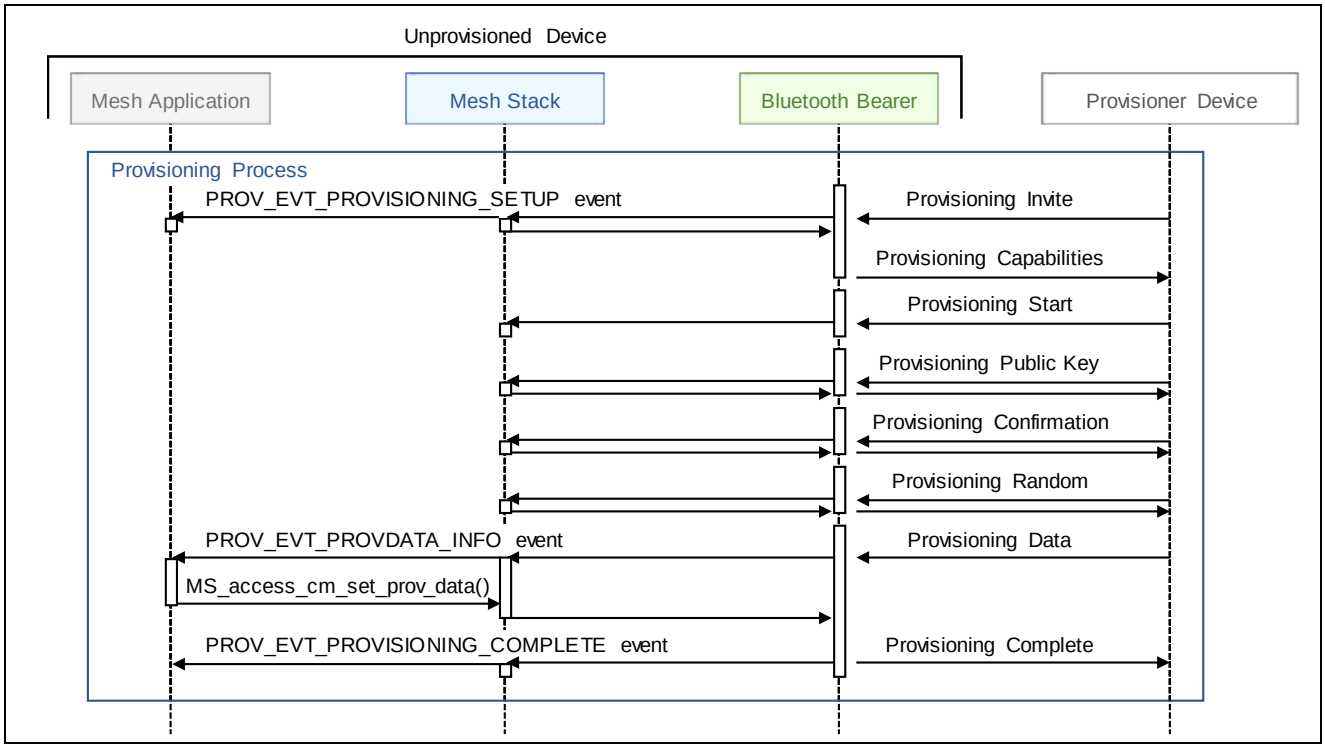

Figure 3-5 Provisioning Process

To reduce security risk of Provisioning Procedure, the followings are recommended:

- Using OOB (Out Of Band) in Public Key Exchange step
- Selecting a cryptographically secure random value or a pseudorandom number having the maximum permitted 128bit entropy as AuthValue in Authentication step

To use OOB in Public Key Exchange step, get a Public Key from Mesh Stack by using MS_prov_get_local_public_key() and deliver the Public Key to Provisioner via OOB.

Random number, generated using seed given from hardware, can be got by using R_BLE_VS_GetRand() of R_BLE API. When Unprovisioned Device uses Output OOB in Authentication step, set a 128bit random number, got by R_BLE_VS_GetRand(), to MS_prov_set_authval() as AuthValue. Also, deliver the AuthValue to Provisioner via OOB.

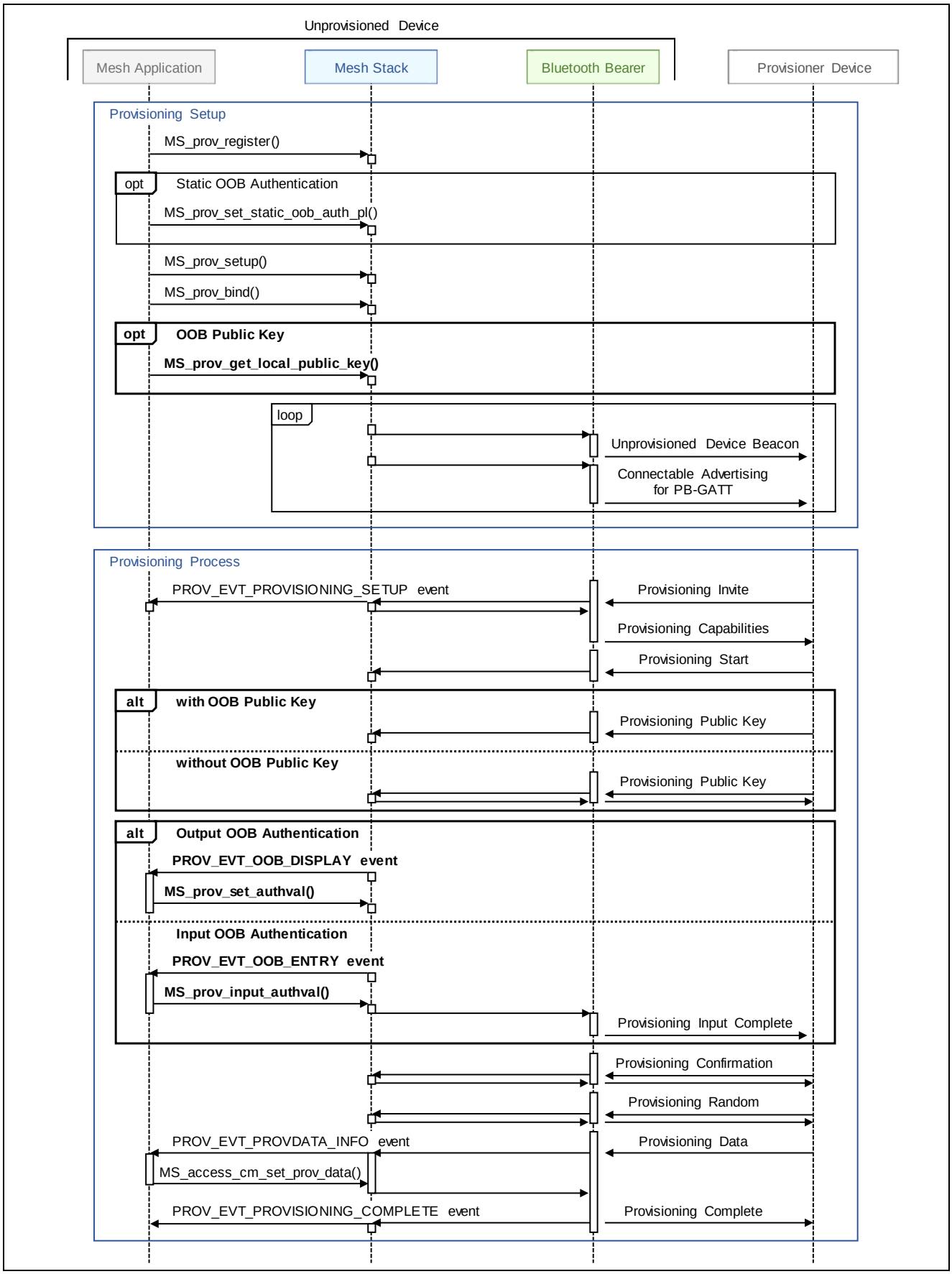

Figure 3-6 Provisioning Process with OOB

3.4 Proxy

This section shows how to implement for working as a Proxy Server or Proxy Client.

3.4.1 Proxy Server

Mesh Sample Program can work as a Proxy Server. Processing for working as a Proxy Server is shown as below.

- **Registration of Proxy Callback Function (mesh_core.c)**

Set Bluetooth Bearer Mode to BLEBRR_GATT_PROXY_MODE. Register a Proxy callback function with Mesh Stack by MS_proxy_register().

```
    R_MS_BRR_Set_GattMode(BLEBRR_GATT_PROXY_MODE);

    /* Register Proxy Callback */
    retval = MS_proxy_register(mesh_core_proxy_cb);
```

- **Starting Connectable Advertising (mesh_core.c)**

To establish a Proxy connection with Proxy Client, start Connection Advertising by MS_proxy_server_adv_start().

```
    retval = MS_proxy_server_adv_start(MS_PRIMARY_SUBNET, proxy_adv_mode);
```

- **Proxy Callback Function (mesh_core.c)**

Implement a callback function to receive Proxy events.

When a connection is established and GATT Proxy Service is enabled, MS_PROXY_UP_EVENT is notified. To deliver Key Refresh Flag, IV Update Flag, and current IV Index to Proxy Client. send Secure Network Beacon by MS_net_broadcast_secure_beacon().

```
static void mesh_core_proxy_cb
                (
                    NETIF_HANDLE * handle,
                    UCHAR          event,
                    UCHAR        * data_param,
                    UINT16         data_len
                )
{
    MS_SUBNET_HANDLE subnet_handle;
    UINT8 netkey[MS_ACCESS_NETKEY_SIZE];

    switch (event)
    {
        case MS_PROXY_UP_EVENT:
            /* Send Secure Network Beacons */
            for (subnet_handle = 0; subnet_handle < MESH_CFG_MAX_SUBNETS; subnet_handle++)
            {
                /* Check if each subnet is present */
                if (API_SUCCESS == MS_access_cm_get_netkey(subnet_handle, netkey))
                {
                    retval = MS_net_broadcast_secure_beacon(subnet_handle);
                }
            }
        break;
```

```
        case MS_PROXY_DOWN_EVENT:
            break;
    }
}
```

- **Terminating Proxy Connection (mesh_core.c)**

To terminate a connection, call R_MS_BRR_Disconnect().

```
    retval = R_MS_BRR_Disconnect(gs_proxy_client_conn_hdl);
```

3.4.2 Proxy Client

Mesh Sample Program can work as a Proxy Server. Processing for working as a Proxy Server is shown as below.

- **Registration of Proxy Callback Function (mesh_core.c)**

Set Bluetooth Bearer Mode to BLEBRR_GATT_PROXY_MODE. Register a Proxy callback function with Mesh Stack by MS_proxy_register().

```
    R_MS_BRR_Set_GattMode(BLEBRR_GATT_PROXY_MODE);

    /* Register Proxy Callback */
    retval = MS_proxy_register(mesh_core_proxy_cb);
```

- **Establishing Proxy Connection (mesh_core.c)**

To establish a Proxy connection with Proxy Server, call R_BRR_Create_Connection(). To scan Proxy Servers, call R_MS_BRR_Scan_GattBearer(). Device Address of each Proxy Server is notified by BLEBRR_GATT_IFACE_SCAN event.

```
    st_ble_dev_addr_t remote_addr;
    retval = R_MS_BRR_Create_Connection(&remote_addr, BLEBRR_GATT_PROXY_MODE);
```

- **Proxy Callback Function (mesh_core.c)**

Implement a callback function to receive Proxy events.

When a connection is established and GATT Proxy Service is enabled, MS_PROXY_UP_EVENT is notified. Configure Proxy Filter Type of Proxy Server with either MS_proxy_set_whitelist_filter() or MS_proxy_set_blacklist_filter(). Add Subscription Address to Proxy Filter List of Proxy Server by MS_access_cm_get_all_model_subscription_list().

```
static void mesh_core_proxy_cb
                (
                    NETIF_HANDLE * handle,
                    UCHAR          event,
                    UCHAR        * data_param,
                    UINT16         data_len
                )
{
    switch (event)
    {
        case MS_PROXY_UP_EVENT:
```

```
            retval = MS_proxy_set_whitelist_filter(handle, MS_PRIMARY_SUBNET);
            gs_proxy_opcode = MS_PROXY_SET_FILTER_OPCODE;
            break;

        case MS_PROXY_STATUS_EVENT:
            switch (gs_proxy_opcode)
            {
                case MS_PROXY_SET_FILTER_OPCODE:
                    /* Add Subscription Addresses to Proxy filter list */
                    retval = mesh_core_proxy_add_addresses(handle, MS_PRIMARY_SUBNET);
                    gs_proxy_opcode = MS_PROXY_ADD_TO_FILTER_OPCODE;
                    break;
                }

                default:
                    gs_proxy_opcode = 0xFF;
                    break;
            }
            break;

        case MS_PROXY_DOWN_EVENT:
            break;
    }
}

static API_RESULT mesh_core_proxy_add_addresses(NETIF_HANDLE * netif_hdl, MS_SUBNET_HANDLE
subnet_hdl)
{
    PROXY_ADDR addr_list[MESH_CFG_MAX_VIRTUAL_ADDRS + MESH_CFG_MAX_NON_VIRTUAL_ADDRS];
    UINT16     addr_count = ARRAY_SIZE(addr_list);

    retval = MS_access_cm_get_all_model_subscription_list(&addr_count, addr_list);
    if (0 != addr_count)
    {
        retval = MS_proxy_add_to_list(netif_hdl, subnet_hdl, addr_list, addr_count);
    }

    return retval;
}
```

- **Terminating Proxy Connection (mesh_core.c)**

To terminate Proxy connection, call R_MS_BRR_Disconnect().

```
    retval = (API_SUCCESS == R_MS_BRR_Disconnect(gs_proxy_server_conn_hdl);
```

3.4.3 Proxy Sequence

(1) Proxy Setup

This sample program supports Proxy feature so Configuration Client that supports only GATT bearer can configure the sample program over GATT bearer. Moreover, this sample program can forward messages between GATT bearer and ADV bearer for a node that supports only GATT bearer.

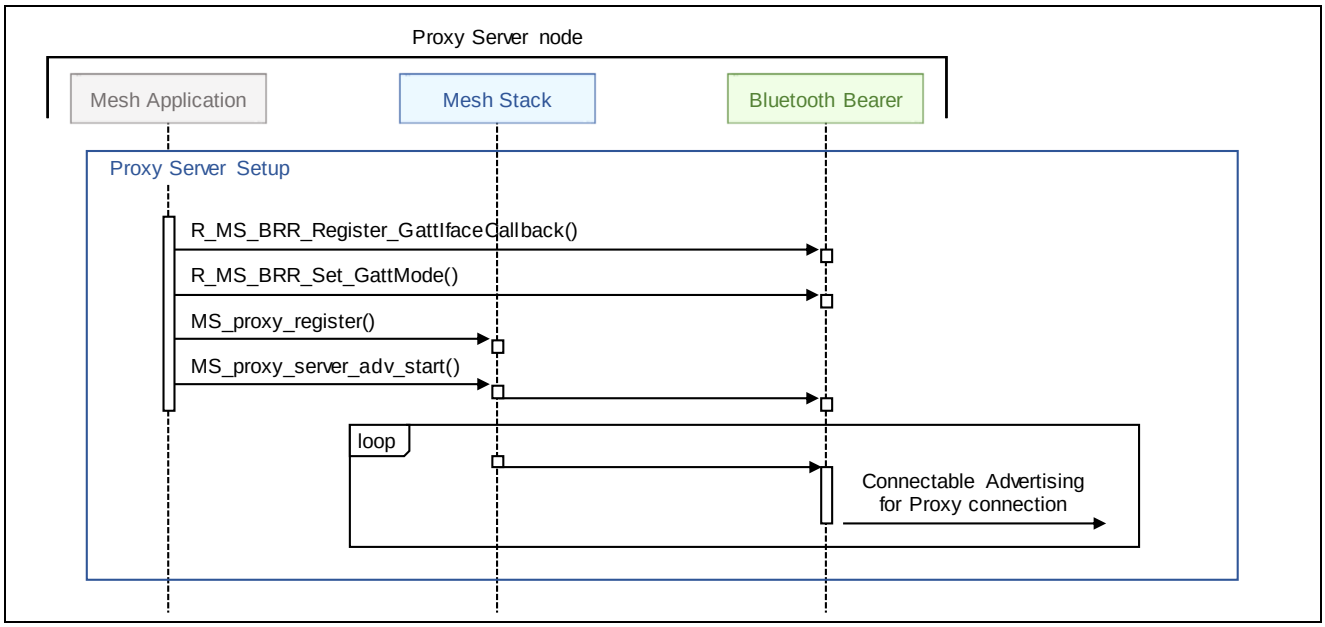

Figure 3-7 Proxy Setup

(2) Proxy Connection Establishment

Proxy Client establishes a connection with Proxy Server and enables notification of the Mesh Proxy Service. After enabling Notification, Proxy Client becomes able to perform Message Communication over GATT bearer.

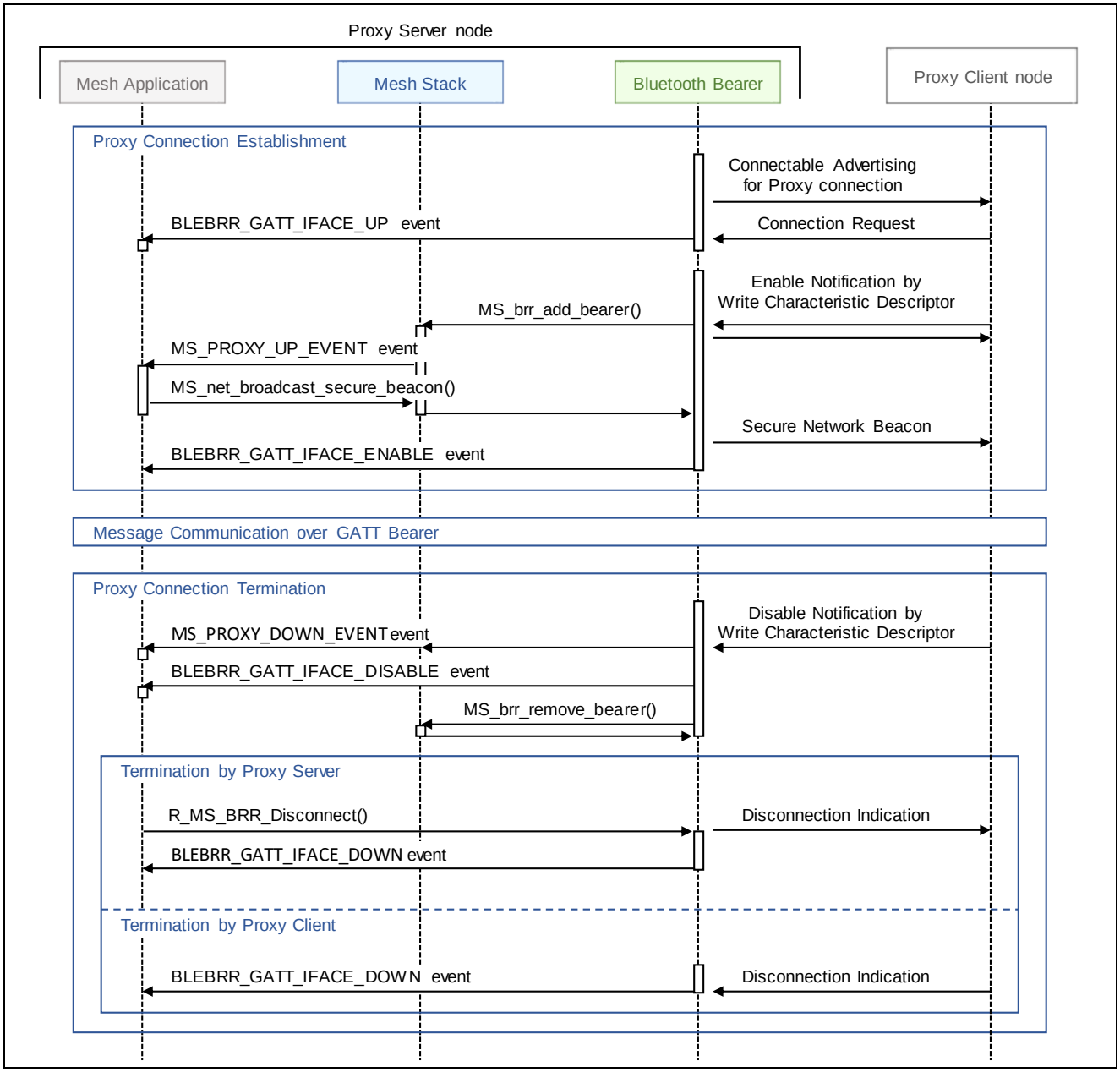

Figure 3-8 Proxy Connection Establishment and Termination

3.5 Friendship

This section shows how to implement for working as a Friend node or Low Power node.

3.5.1 Friend Node

To work as a Friend node, Friend feature must be enabled. Friend feature is enabled by the following way:

- Configuration Client sends Config Friend Set message.
- Mesh Application calls MS_ENABLE_FRIEND_FEATURE().

After enabling Friend feature, Friend-related-processing such as Friendship establishment, Friend Queue management, and response for Low Power node is handled automatically by Mesh Stack, so application does not have to handle it.

3.5.2 Low Power Node

To work as a Low Power node, application must enable Low Power feature and request a Friend node to establish a Friendship. After establishing a Friendship, Mesh Stack polls the Friend node if any messages are stored and suspends and resumes Scan automatically.

Mesh Sample Program can work as a Low Power Node. Processing for establishing a Friendship as a Low Power node is shown as below.

NOTE: This feature is disabled by default. To enable this feature, change the value of LOW_POWER_FEATURE_EN macro by referring to Section 2.6.

- **Enabling Low Power feature (mesh_core.c)**

Enable Low Power feature by MS_ENABLE_LPN_FEATURE().

```
#if LOW_POWER_FEATURE_EN
MS_ENABLE_LPN_FEATURE();
#endif /* LOW_POWER_FEATURE_EN */
```

- **Requirement for Friendship Establishment (mesh_core.c)**

To establish a friendship as a Low Power node, send Friend Request message by MS_trn_lpn_setup_friendship(). As arguments for this function, set parameters related to timing of polling Friend node as well as Friendship callback function.

```
#if LOW_POWER_FEATURE_EN
API_RESULT mesh_core_lpn_setup(void)
{
    API_RESULT retval;

    retval = MS_trn_lpn_setup_friendship
                (
                    0x00,
                    CORE_FRIEND_CRITERIA,
                    CORE_FRIEND_RECEIVE_DELAY,
                    CORE_FRIEND_POLLTIMEOUT,
                    CORE_FRIEND_SETUPTIMEOUT,
                    mesh_core_lpn_cb
                );

    return retval;
```

```
}
#endif /* LOW_POWER_FEATURE_EN */
```

- **Friendship Callback Function (mesh_core.c)**

Implement a callback function to receive Friendship events notified by Mesh Stack. When a friendship is established, MS_TRN_FRIEND_SETUP_CNF is notified. Also, when a friendship is terminated, MS_TRN_FRIEND_TERMINATE_IND is notified.

Low Power node can add and remove Subscription Addresses to/from Friend Subscription List of Friend node. After establishing a Friendship, Mesh Sample Program gets all Subscription Addresses from Subscription List by MS_access_cm_get_all_model_subscription_list() and adds them to Friend Subscription List of Friend node by MS_trn_lpn_subscrn_list_add().

```
static void mesh_core_lpn_cb
                (
                    MS_SUBNET_HANDLE    subnet_handle,
                    UCHAR               event_type,
                    UINT16              status
                )
{
    UCHAR  seek_req = MS_FALSE;
    UINT16 subscrn_list[MESH_CFG_FRIEND_SUBSCRIPTION_LIST_SIZE];
    UINT16 subscrn_count = MESH_CFG_FRIEND_SUBSCRIPTION_LIST_SIZE;

    switch(event_type)
    {
        case MS_TRN_FRIEND_SETUP_CNF:
        {
            /* Friendship is established. */
            if (API_SUCCESS == status)
            {
                retval = MS_access_cm_get_all_model_subscription_list
                            (
                                &subscrn_count, subscrn_list
                            );

                if (0 != subscrn_count)
                {
                    retval = MS_trn_lpn_subscrn_list_add(subscrn_list, subscrn_count);
                }
            }
            /* Setup timeout is expired, and Friendship is not established. */
            else
            {
                seek_req = MS_TRUE;
            }
        }
        break;

        case MS_TRN_FRIEND_TERMINATE_IND:
        {
            /* Friendship is terminated. */
            seek_req = MS_TRUE;
        }
        break;
    }
```

```
    if (MS_TRUE == seek_req)
    {
        /* Seek a Friend Node again */
        mesh_core_lpn_setup();
    }
}
```

3.5.3 Low Power Node Sequence

(1) Enabling Low Power Feature and Friendship Request

This sample program supports Low Power feature and transmits Friend Request to establish a Friendship.

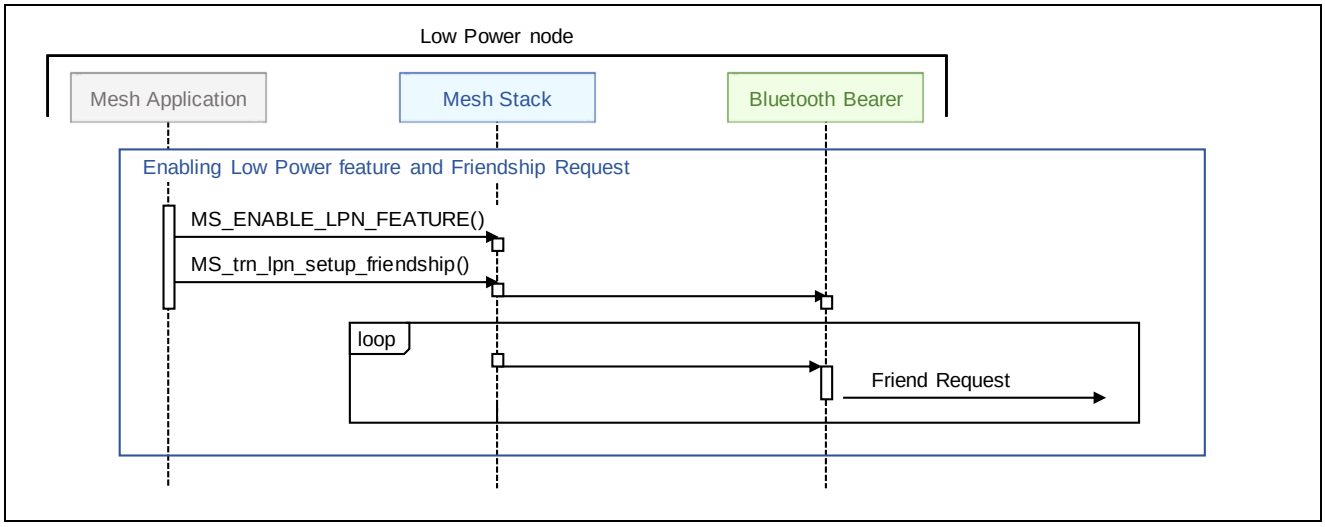

Figure 3-9 Enabling Low Power Feature and Friendship Request

(2) Friendship Establishment and Termination

Friendship is established by receiving Friend Offer. After Friendship establishment, this sample program registers all Subscription Addresses with Friend Subscription List of Friend node. After registration, the Friend node stores messages addressed to the Subscription Addresses. Also, Low Power node performs Message Polling periodically and receives messages from the Friend node.

When Message Polling fails continuously and then Friendship is terminated by arising Friend Poll Timeout, this sample program transmits Friend Request to establish a Friendship again.

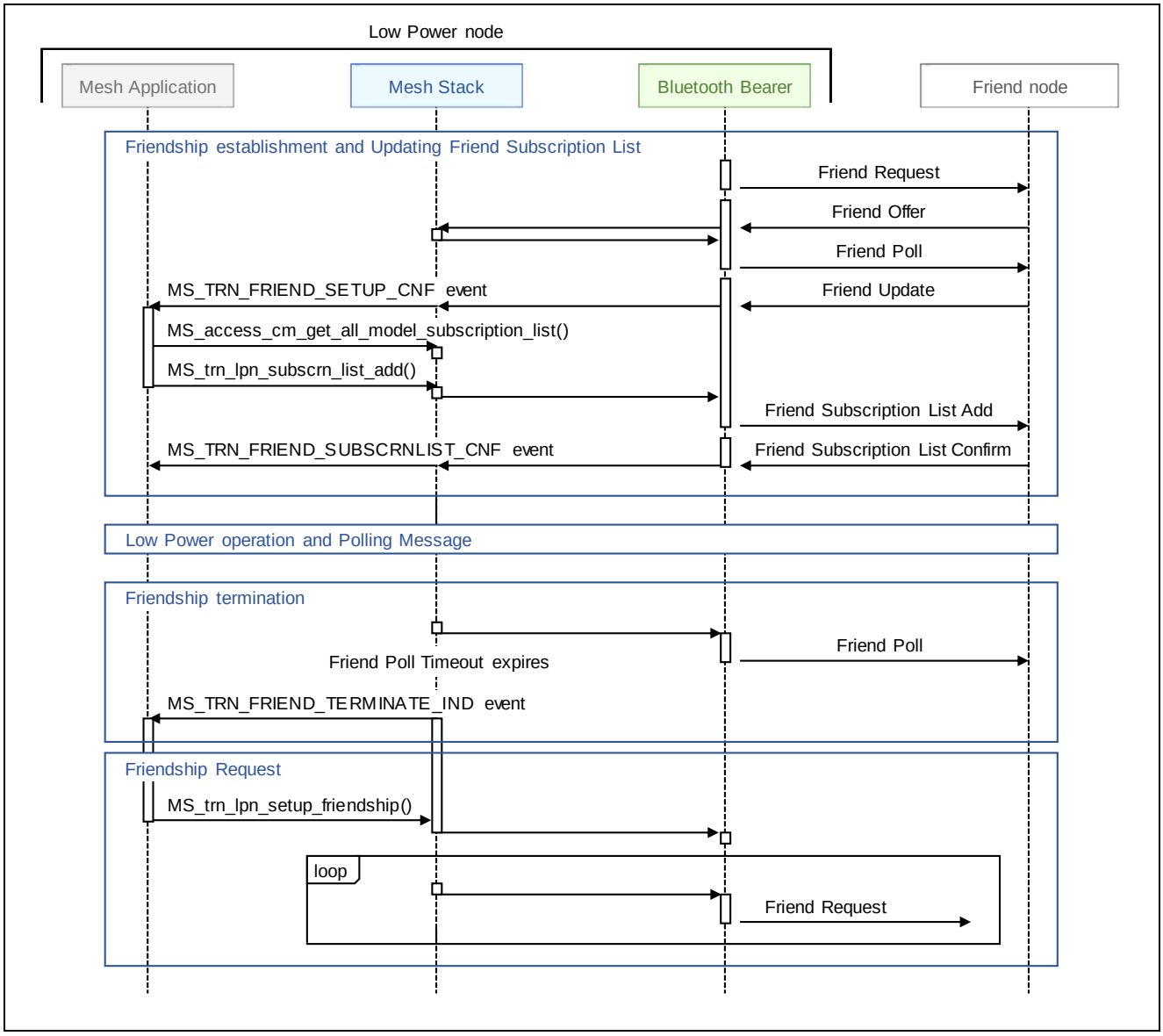

Figure 3-10 Friendship Establishment and Termination

If Low Power Node terminates a Friendship spontaneously, send Friend Clear message by MS_trn_lpn_clear_friendship(). Completion of termination is notified by MS_TRN_FRIEND_CLEAR_CNF event.

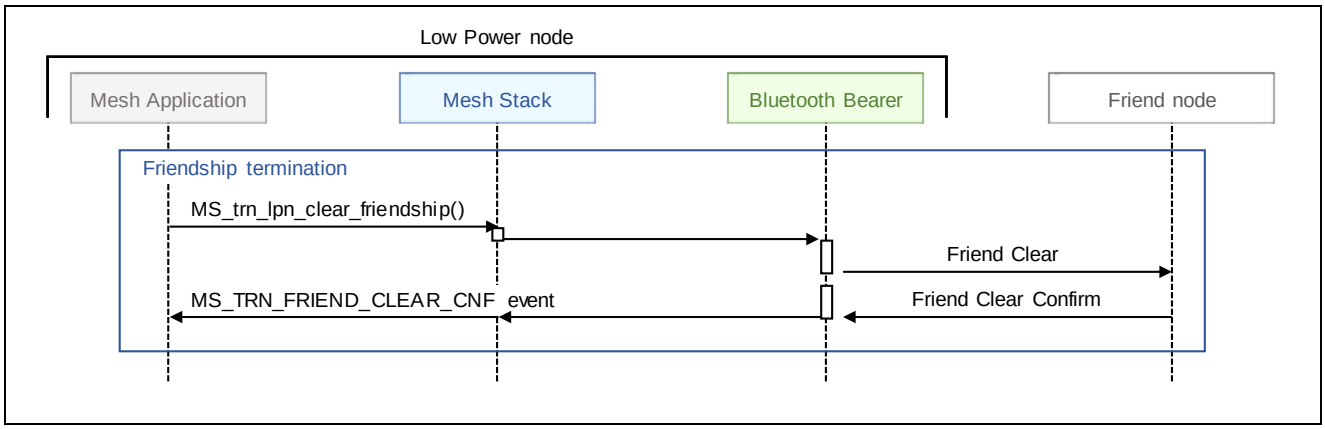

Figure 3-11 Friendship Termination

3.5.4 Friend Node Sequence

(1) Enabling Friend Feature

By being enabled Friend feature by Configuration Client, this sample program can work as a Friend Node. Mesh Stack does not notify any events regarding Friend Node operation such as friendship establishment and termination.

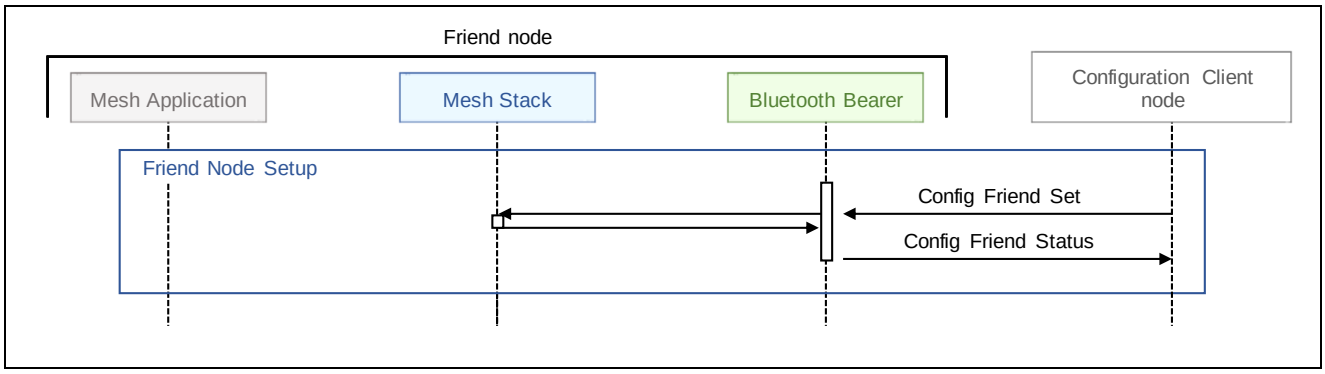

Figure 3-12 Enabling Friend Feature

(2) Friendship Establishment and Termination

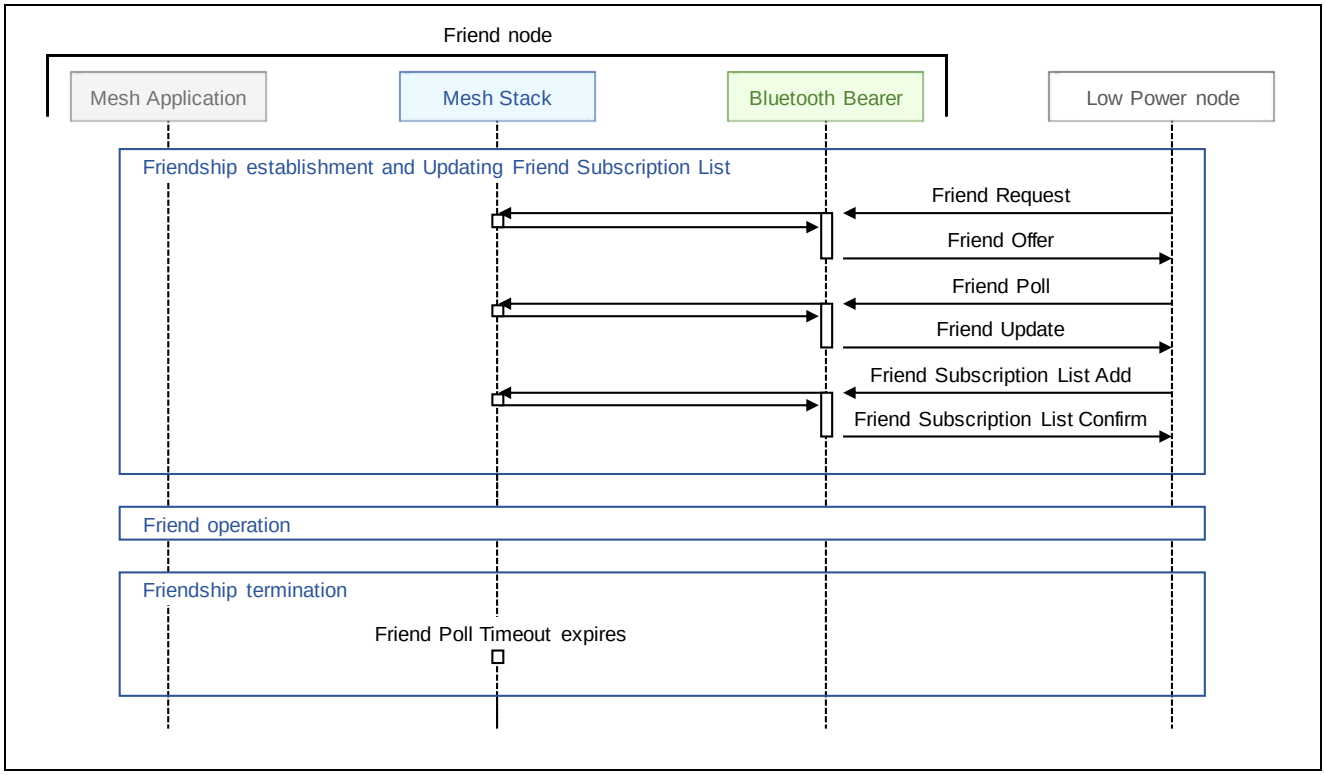

Figure 3-13 Friendship Establishment and Termination

3.6 Configuration

This section shows how to implement for working as a Configuration Server.

3.6.1 Configuration Server

After Provisioning, a node must receive configuration information such as Application Key, Publish Address, and Subscription Address from Configuration Client so that a node communicates by each model. There configuration information is handled as Configuration States by Configuration model.

When application registers Configuration Server model, memory region for Configuration states are allocated in Mesh Stack. By receiving Configuration messages from Configuration Client, Mesh Stack updates the Configuration states automatically. Therefore, application does not have to manage the Configuration states.

Mesh Stack provides application with API to access local Configuration states. Application can access the Configuration states directly by using Mesh Stack API. Regarding API to access local Configuration states, refer to refer to [Modules]→[Mesh Core]→[ACCESS (Access Layer)]→[API Definitions] in the Bluetooth Mesh Stack API Manual "blemesh_api.chm".

Mesh Sample Program works as a Configuration Server. Implementation for Configuration Server Model of Mesh Sample Program is shown as below.

- **Registration of Configuration Server Model (mesh_model.c)**

Register Configuration Server model with element by MS_config_server_init().

```
    retval = MS_config_server_init
                (
                    gs_element_handle,
                    &gs_config_server_model_handle,
                    mesh_model_config_server_cb
                );
```

- **Configuration Server Callback Function (mesh_model.c)**

Implement a callback function to receive a message from Configuration Client.

```
static API_RESULT mesh_model_config_server_cb
                (
                    MS_ACCESS_MODEL_REQ_MSG_CONTEXT * ctx,
                    UINT32                    opcode,
                    UCHAR                   * data_param,
                    UINT16                    data_len
                )
{
    return API_SUCCESS;
}
```

3.6.2 Configuration Server Sequence

When receiving Config Node Reset message, Mesh Stack delete all Configuration states. Also, this sample program resets MCU and performs Provisioning again.

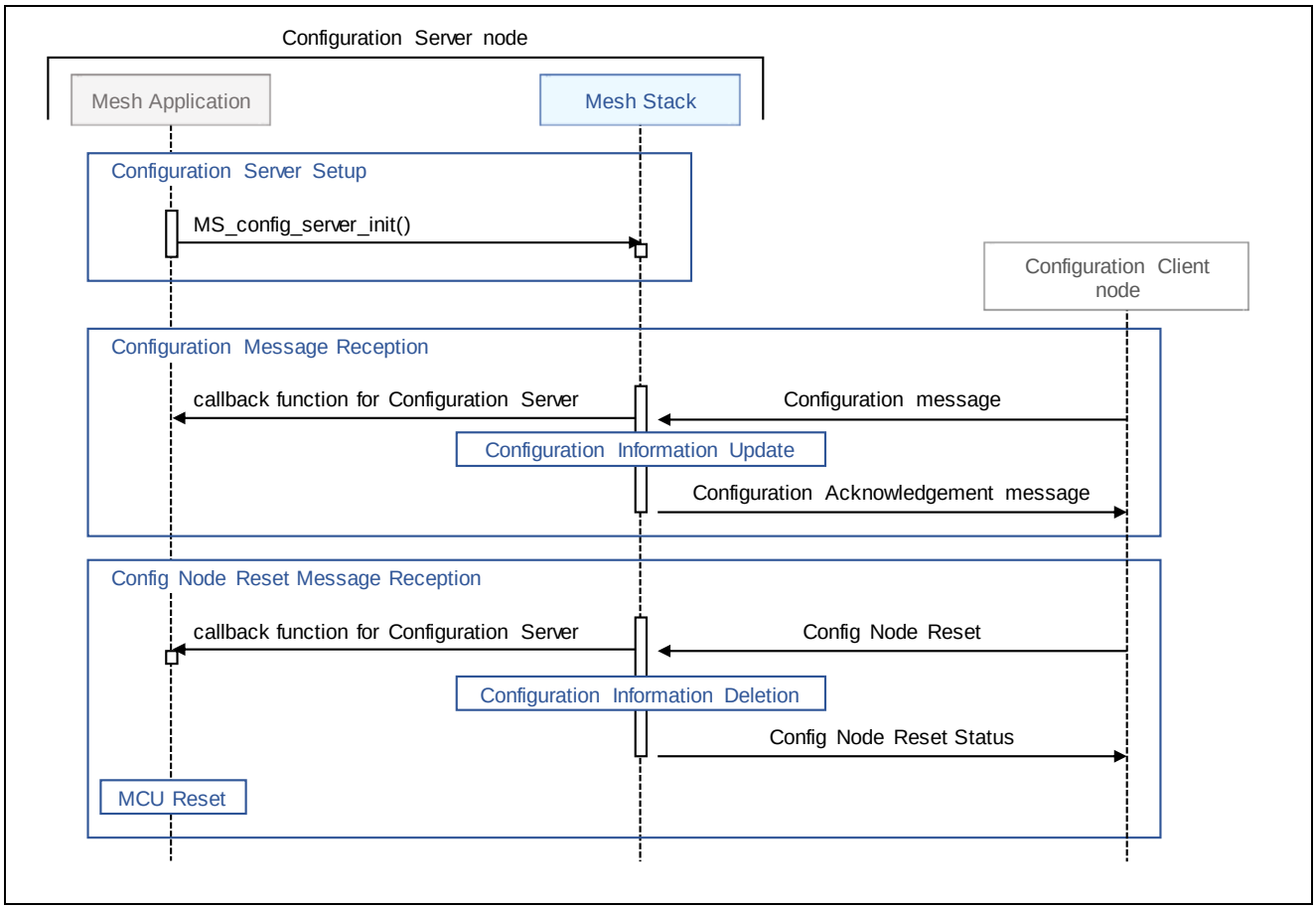

Figure 3-14 Configuration Server Model Operation of Mesh Sample Program

3.7 Health Model

This section shows how to implement for working as a Health Server.

3.7.1 Health Server

Health Server performs self-testing by receiving Health Fault Test message from Health Client. Also, Health Server starts Attention Timer by receiving Health Attention Set message from Health Client.

- **Registration of Health Server Model (mesh_model.c)**

Register Health Server model with element by MS_health_server_init().

```
    retval = MS_health_server_init
                (
                    gs_element_handle,
                    &gs_health_server_model_handle,
                    MESH_CFG_DEFAULT_COMPANY_ID,
                    gs_callbacks.self_tests,
                    gs_callbacks.num_self_tests,
                    gs_callbacks.health_cb
                );
```

- **Definition of Test ID (main.c)**

Define Test IDs of self-testing performed by receiving Health Fault Test message.

```
typedef enum
{
    MESH_HEALTH_TEST_ID_00 = 0x00,
    MESH_HEALTH_TEST_ID_01 = 0x01,
} e_mesh_health_test_id_t;
```

- **Definition of Test Function (main.c)**

Define Test Functions of self-testing performed by receiving Health Fault Test message.

```
static void mesh_health_self_test_00(UINT8 test_id, UINT16 company_id)
{
    if ((MESH_HEALTH_TEST_ID_00 == test_id) && (MESH_CFG_DEFAULT_COMPANY_ID == company_id))
    {
        CONSOLE_OUT("[HEALTH] A Self-Test Procedure(TestID: 0x00)\n");
        mesh_model_health_server_fault_status(MESH_HEALTH_TEST_ID_00, MS_HEALTH_FAULT_NO_FAULT);
    }
}
```

- **Registration of Pairs of Test ID and Test Function (main.c)**

Register pairs of defined Test IDs and defined Test functions.

```
static MS_HEALTH_SERVER_SELF_TEST gs_health_server_self_tests[] =
{
    { MESH_HEALTH_TEST_ID_00, mesh_health_self_test_00 },
    { MESH_HEALTH_TEST_ID_01, mesh_health_self_test_01 },
  };
```

- **Setting and Notification of Self Testing Result (mesh_model.c)**

Call MS_health_server_report_fault() to add self testing result to Fault state and to send Health Fault Status message.

```
API_RESULT mesh_model_health_server_fault_status(UINT8 test_id, UINT8 fault_code)
{
    API_RESULT retval;

    retval = MS_health_server_report_fault
                (
                    &gs_health_server_model_handle,
                    test_id,
                    MESH_CFG_DEFAULT_COMPANY_ID,
                    fault_code
                );

    return retval;
}
```

- **Attention Timer Callback Function (main.c)**

Implement an Attention Timer callback function performed by receiving Health Attention Set message.

MS_HEALTH_SERVER_ATTENTION_START event is notified when Attention Timer starts, and MS_HEALTH_SERVER_ATTENTION_RESTART event is notified when Attention Timer restarts. Start attention behavior such as LED blinking by these events.

MS_HEALTH_SERVER_ATTENTION_STOP event is notified when Attention Timer stops. Stop attention behavior by this event.

```
static API_RESULT mesh_health_server_cb
                (
                    MS_ACCESS_MODEL_HANDLE * handle,
                    UINT8                    event_type,
                    UINT8                  * event_param,
                    UINT16                   param_len
                )
{
    UINT8 attention_sec;

    switch (event_type)
    {
        case MS_HEALTH_SERVER_ATTENTION_START:
        case MS_HEALTH_SERVER_ATTENTION_RESTART:
            attention_sec = *event_param;
            if (0 != attention_sec)
            {
            }
            break;

        case MS_HEALTH_SERVER_ATTENTION_STOP:
            break;
    }

    return API_SUCCESS;
}
```

3.7.2 Health Server Sequence

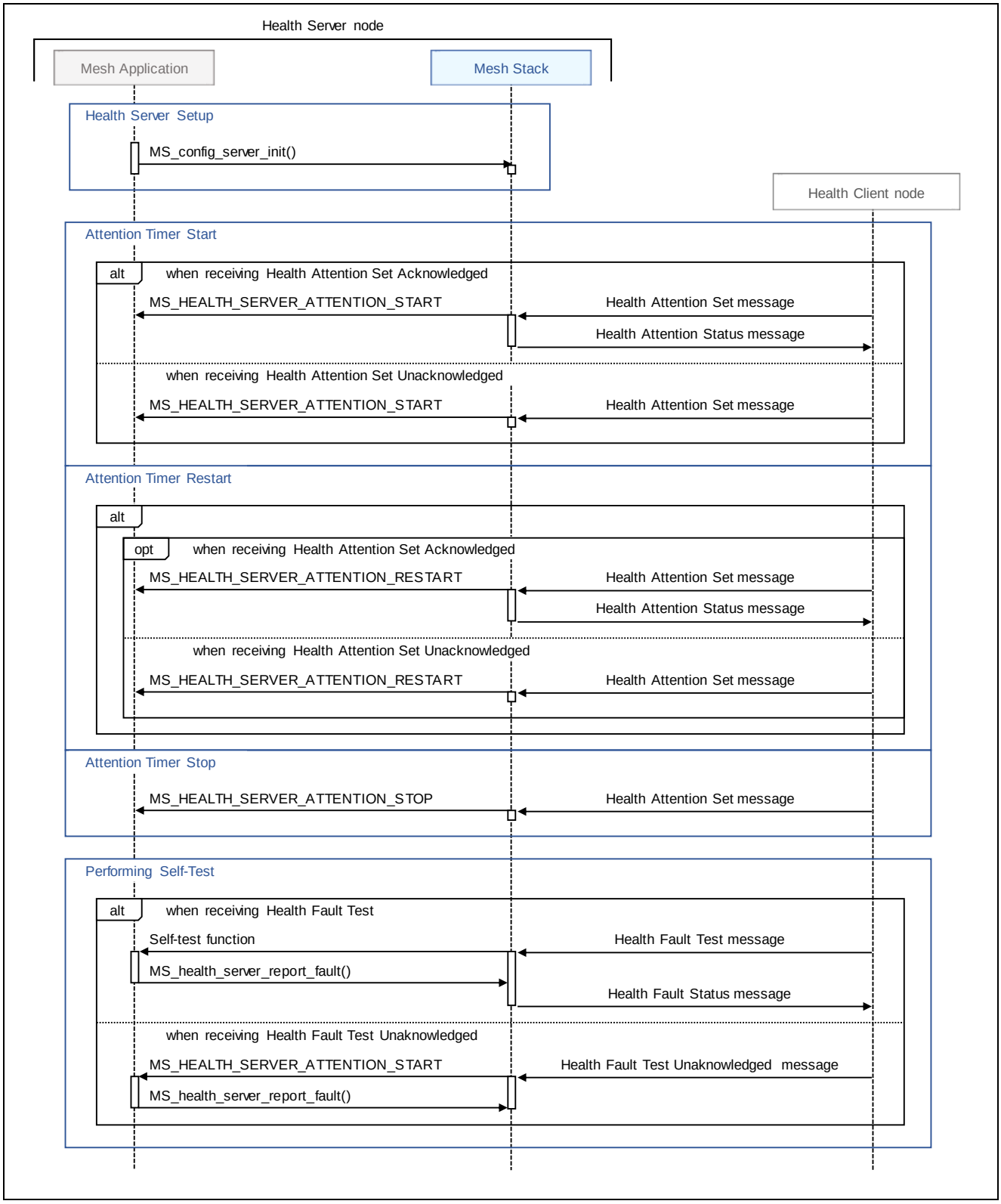

Figure 3-15 Health Server Model Operation of Mesh Sample Program

3.8 Application Model

Models that should be used by application differs depends on each application scenario. Application can use single model or multiple models. Mesh Stack provides application with API to use models defined by Bluetooth Mesh Model Specification.

This section shows how to implement Application Models while referring to the implementation of Generic OnOff model of Mesh Sample Program.

Mesh Sample Program works as a Configuration Client or Generic OnOff Server. Generic OnOff Client model can change the Generic OnOff state of Generic OnOff Server model into either ON or OFF. Implementation for Configuration Server Model of Mesh Sample Program is shown as below.

3.8.1 Server Model

- **Instance of state (mesh_model.c)**

Implement a global variable as instance of state by using the structure type for each state.

```
#ifdef ONOFF_SERVER_MODEL
static MS_STATE_GENERIC_ONOFF_STRUCT gs_onoff_state;
#endif /* ONOFF_SERVER_MODEL */
```

- **Initialization of the state (mesh_model.c)**

Initialize the state defined as the global variable.

```
    #ifdef ONOFF_SERVER_MODEL
    memset(&gs_onoff_state, 0, sizeof(gs_onoff_state));
    #endif /* ONOFF_SERVER_MODEL */
```

- **Registration of Server Model (mesh_model.c)**

Register Server Model to register its element handle and the callback function.

```
    #ifdef ONOFF_SERVER_MODEL
    retval = MS_generic_onoff_server_init
                (
                    gs_element_handle,
                    &gs_onoff_server_model_handle,
                    mesh_model_onoff_server_cb
                );
    #endif /* ONOFF_SERVER_MODEL */
```

- **Server Model Callback function (mesh_model.c)**

Implement a callback function to receive messages from Client and handle the state defined as the global variable.

```
#ifdef ONOFF_SERVER_MODEL
static API_RESULT mesh_model_onoff_server_state_get(UCHAR type, void * state)
{
    MS_STATE_GENERIC_ONOFF_STRUCT * param_p;
    API_RESULT retval = API_SUCCESS;
```

```
    param_p = (MS_STATE_GENERIC_ONOFF_STRUCT *)state;

    switch(type)
    {
        case MS_STATE_GENERIC_ONOFF_T:
        {
            memcpy(param_p, &gs_onoff_state, sizeof(MS_STATE_GENERIC_ONOFF_STRUCT));
        }
        break;
    }

    return retval;
}

static API_RESULT mesh_model_onoff_server_state_set(UCHAR type, void * state)
{
    MS_STATE_GENERIC_ONOFF_STRUCT * param_p;
    API_RESULT retval = API_SUCCESS;

    param_p = (MS_STATE_GENERIC_ONOFF_STRUCT *)state;

    switch (type)
    {
        case MS_STATE_GENERIC_ONOFF_T:
        {
            memcpy(&gs_onoff_state, param_p, sizeof(MS_STATE_GENERIC_ONOFF_STRUCT));
        }
        break;
    }

    return retval;
}

static API_RESULT mesh_model_onoff_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
                )
{
    MS_IGNORE_UNUSED_PARAM(ext_params);
    MS_IGNORE_UNUSED_PARAM(msg_raw);
    API_RESULT  retval;
    MS_ACCESS_ELEMENT_HANDLE      elem_handle;
    MS_ACCESS_MODEL_STATE_PARAMS  current_state_params;
    MS_STATE_GENERIC_ONOFF_STRUCT param;
    UCHAR                         reply, publish;

    retval = MS_access_get_element_handle_for_model_handle(ctx->handle, &elem_handle);
    if (API_SUCCESS == retval)
    {
        /* Check Message Type */
        switch (req_type->type)
        {
            case MS_ACCESS_MODEL_REQ_MSG_T_GET:
            {
                retval = mesh_model_onoff_server_state_get(state_params->state_type, &param);
                current_state_params.state_type = state_params->state_type;
                current_state_params.state = &param;
                reply = ((MS_NET_ADDR_UNASSIGNED != ctx->daddr) &&
```

```
                        (MS_NET_ADDR_UNASSIGNED != ctx->saddr)) ? MS_TRUE : MS_FALSE;
                publish = MS_FALSE;
            }
            break;

            case MS_ACCESS_MODEL_REQ_MSG_T_SET:
            {
                retval = mesh_model_onoff_server_state_set
                            (
                                state_params->state_type, state_params->state
                            );
                if (API_SUCCESS == retval)
                {
                    USER_CALLBACK(onoff_server_set_cb, ctx, state_params->state);
                }
                current_state_params.state_type = state_params->state_type;
                current_state_params.state = state_params->state;
                reply = (0x01 == req_type->to_be_acked) ? MS_TRUE : MS_FALSE;
                publish = MS_TRUE;
            }
            break;

            default:
            {
                retval = API_FAILURE;
            }
            break;
        }

        if (API_SUCCESS == retval)
        {
            retval = MS_generic_onoff_server_state_update
                     (
                        ctx, &current_state_params, NULL, 0, NULL, reply, publish
                     );
            CONSOLE_STATUS("[GENERIC_ONOFF] MS_generic_onoff_server_state_update()", retval);
        }
    }

    return retval;
}
#endif /* ONOFF_SERVER_MODEL */
```

3.8.2 Client Model

- **Registration of Client Model (mesh_model.c)**

Register Client Model to register its element handle and the callback function.

```
    #ifdef ONOFF_CLIENT_MODEL
    retval = MS_generic_onoff_client_init
                (
                    gs_element_handle,
                    &gs_onoff_client_model_handle,
                    mesh_model_onoff_client_cb
                );
    #endif /* ONOFF_CLIENT_MODEL */
```

- **Callback function to receive messages (mesh_model.c)**

Implement a callback function to receive messages from Server.

```
#ifdef ONOFF_CLIENT_MODEL
static API_RESULT mesh_model_onoff_client_cb
                (
                    MS_ACCESS_MODEL_REQ_MSG_CONTEXT * ctx,
                    UINT32                    opcode,
                    UCHAR                   * data_param,
                    UINT16                    data_len
                )
{
    API_RESULT retval = API_SUCCESS;
    MS_GENERIC_ONOFF_STATUS_STRUCT status;

    switch (opcode)
    {
        case MS_ACCESS_GENERIC_ONOFF_STATUS_OPCODE:
        {
            memcpy(&status, data_param, data_len);
            status.optional_fields_present =
                    (data_len > sizeof(status.present_onoff)) ? MS_TRUE : MS_FALSE;
        }
        break;
    }

    return retval;
}
#endif /* ONOFF_CLIENT_MODEL */
```

- **Functions to transmit messages (mesh_model.c)**

Implement functions to transmit message such as GET and SET. In addition to this, application must execute these functions in accordance with a trigger such as push-switch.

```
#ifdef ONOFF_CLIENT_MODEL
API_RESULT mesh_model_onoff_client_get(void)
{
    API_RESULT retval;

    retval = MS_generic_onoff_get();
```

```
    return retval;
}

API_RESULT mesh_model_onoff_client_set(UCHAR tid, UINT8 state)
{
    API_RESULT retval;
    MS_GENERIC_ONOFF_SET_STRUCT  param;

    memset(&param, 0, sizeof(param));
    param.onoff = state;
    param.tid = tid;

    retval = MS_generic_onoff_set(&param);

    return retval;
}

API_RESULT mesh_model_onoff_client_set_unack(UCHAR tid, UINT8 state)
{
    API_RESULT retval;
    MS_GENERIC_ONOFF_SET_STRUCT  param;

    memset(&param, 0, sizeof(param));
    param.onoff = state;
    param.tid = tid;

    retval = MS_generic_onoff_set_unacknowledged(&param);

    return retval;
}
#endif /* ONOFF_CLIENT_MODEL */
```

3.8.3 Generic OnOff Model Sequence

Mesh Sample Program which works as a Generic OnOff Client node sends Generic OnOff Set Unacknowledged message when a switch on board is pushed. On the other hand, Mesh Sample Program which works as a Generic OnOff Server node turns LED on board either on or off when receiving Generic OnOff Set message or Generic OnOff Set Unacknowledged message.

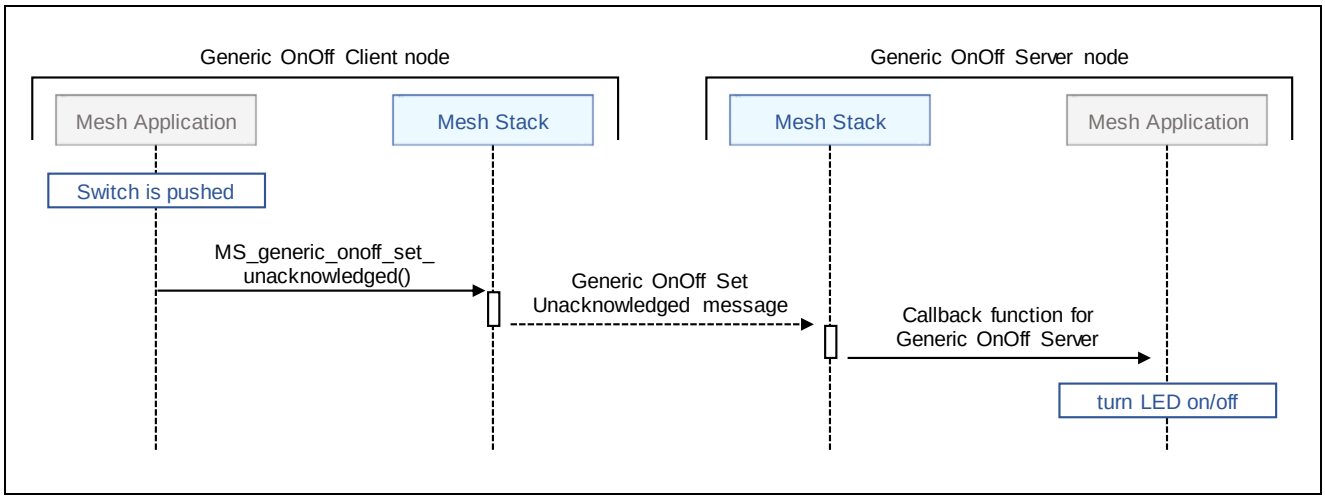

Figure 3-16 Generic OnOff Model Operation of Mesh Sample Program

3.8.4 Vendor Model Sequence

Mesh Sample Program which works as a Vendor Server node sends Vendor Set Unacknowledged message when character string is input from console. On the other hand, Mesh Sample Program which works as a Vendor Server output character string to console when receiving Vendor Set message or Vendor Set Unacknowledged message.

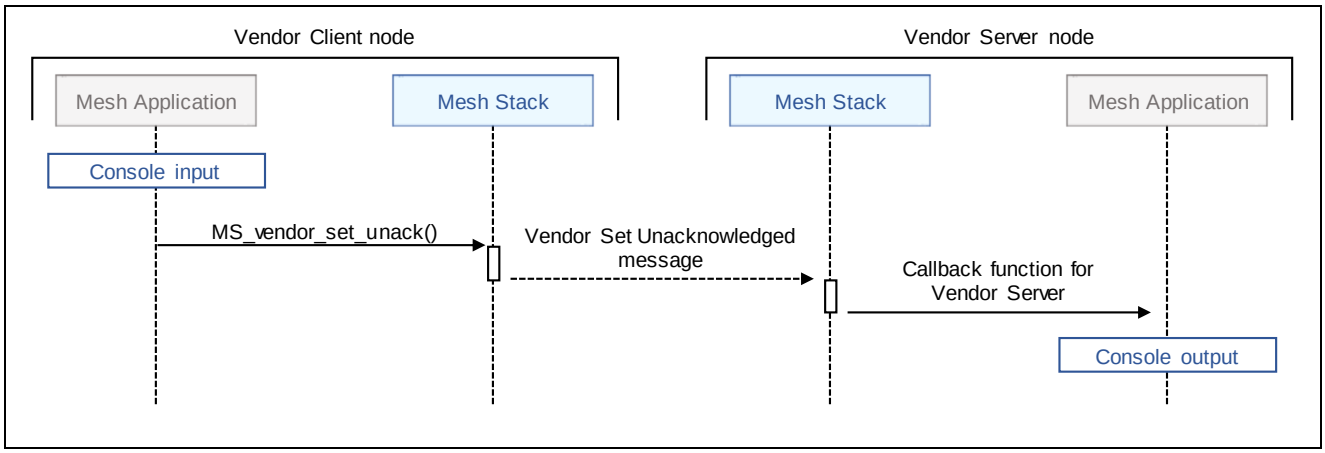

Figure 3-17 Vendor Model Operation of Mesh Sample Program

3.8.5 Mesh Monitoring

Mesh Sample Program can output log of Protocol Data Unit (PDU) and Secure Network Beacon (SNB) that are transmitted and received by each layer of Mesh Stack. This will make it easier to understand communication state when you develop a Mesh application.

Mesh Monitoring of Mesh Sample Program is shown as below.

NOTE: This feature is disabled by default. To enable this feature, change the value of CONSOLE_MONITOR_CFG macro by referring to Section 2.6.

- **Enabling Mesh Monitoring (mesh_model.c)**

Enable Mesh Monitoring by MS_monitor_register_pl().

```
    MS_MONITOR_PL ms_monitor;

    /* Set Monitoring Configuration */
    ms_monitor.cfg = CONSOLE_MONITOR_CFG;

    /* Set Monitoring Callbacks */
    ms_monitor.access_pdu = mesh_core_monitor_access_pdu;
    ms_monitor.trans_pdu = mesh_core_monitor_trans_pdu;
    ms_monitor.ltrans_pdu = mesh_core_monitor_ltrans_pdu;
    ms_monitor.net_pdu = mesh_core_monitor_net_pdu;
    ms_monitor.generic_log = mesh_core_monitor_generic_log;

    /* Register Monitoring Callbacks with Mesh Stack */
    MS_monitor_register_pl(&ms_monitor);

    return API_SUCCESS;
```

- **Mesh Monitoring Callback Function (mesh_model.c)**

Implement console output processing to Mesh monitoring callback functions.

```
static void mesh_core_monitor_access_pdu(UCHAR dir, UINT16 log_msg_id, void * data)
{
    MS_LOGGER_ACCESS_LOG_PKT_INFO * access = (MS_LOGGER_ACCESS_LOG_PKT_INFO *)data;

}

static void mesh_core_monitor_trans_pdu(UCHAR dir, UINT16 log_msg_id, void * data)
{
    char * str_dir = (MS_MONITOR_DIR_TX == dir) ? "TX" : "RX";

    switch (log_msg_id)
    {
        case MS_TRN_LOG_ID_TX_PKT:
        case MS_TRN_LOG_ID_RX_PKT:
        {
            MS_LOGGER_TRN_LOG_PKT_INFO * trn = (MS_LOGGER_TRN_LOG_PKT_INFO *)data;

        }
        break;

        case MS_TRN_LOG_ID_HEART_BEAT_PKT:
        {
```

```
            MS_LOGGER_TRN_LOG_HEART_BEAT_INFO * hb = (MS_LOGGER_TRN_LOG_HEART_BEAT_INFO *)data;

        }
        break;
    }
}

static void mesh_core_monitor_ltrans_pdu(UCHAR dir, UINT16 log_msg_id, void * data)
{
    switch (log_msg_id)
    {
        case MS_LTRN_LOG_ID_TX_PKT:
        {
            MS_LOGGER_LTR_LOG_TX_PKT_INFO * tx = (MS_LOGGER_LTR_LOG_TX_PKT_INFO *)data;

        }
        break;

        case MS_LTRN_LOG_ID_RX_SEG_PKT:
        {
            MS_LOGGER_LTR_LOG_RX_SEG_PKT_INFO * seg = (MS_LOGGER_LTR_LOG_RX_SEG_PKT_INFO *)data;

        }
        break;

        case MS_LTRN_LOG_ID_RX_UNSEG_PKT:
        {
            MS_LOGGER_LTR_LOG_RX_UNSEG_PKT_INFO * unseg = (MS_LOGGER_LTR_LOG_RX_UNSEG_PKT_INFO
*)data;

        }
        break;
    }
}

static void mesh_core_monitor_net_pdu(UCHAR dir, UINT16 log_msg_id, void * data)
{
    switch (log_msg_id)
    {
        case MS_NET_LOG_ID_TX_PKT:
        case MS_NET_LOG_ID_RX_PKT:
        {
            MS_LOGGER_NET_LOG_PKT_INFO * pdu = (MS_LOGGER_NET_LOG_PKT_INFO *)data;

        }
        break;

        case MS_NET_LOG_ID_SNB_PKT:
        {
            MS_LOGGER_NET_LOG_SNB_INFO * snb = (MS_LOGGER_NET_LOG_SNB_INFO *)data;

        }
        break;
    }
}

static void mesh_core_monitor_generic_log(UCHAR module_id, UINT16 log_msg_type, UINT16 status,
void * data)
{
    switch (log_msg_type)
```

```
    {
        case MS_NET_GEN_LOG_ID_QUEUE_FULL:
            break;

        case MS_NET_GEN_LOG_ID_LOG_ID_INVALID_PKT:
            break;

        case MS_LTRN_GEN_LOG_ID_ACK_TIMER_INFO:
            break;

        case MS_LTRN_GEN_LOG_ID_SAR_INCOMPLETE_TIMER_INFO:
            break;

        case MS_LTRN_GEN_LOG_ID_SAR_TIMEOUT_INFO:
            break;
    }
}
```

3.8.5.1 Mesh Monitoring Sequence

This sample program enables Mesh Monitoring functionality of Mesh Stack. Protocol Data Unit (PDU) and Secure Network Beacon (SNB) that are transmitted and received by each layer of Mesh Stack are notified by callback function.

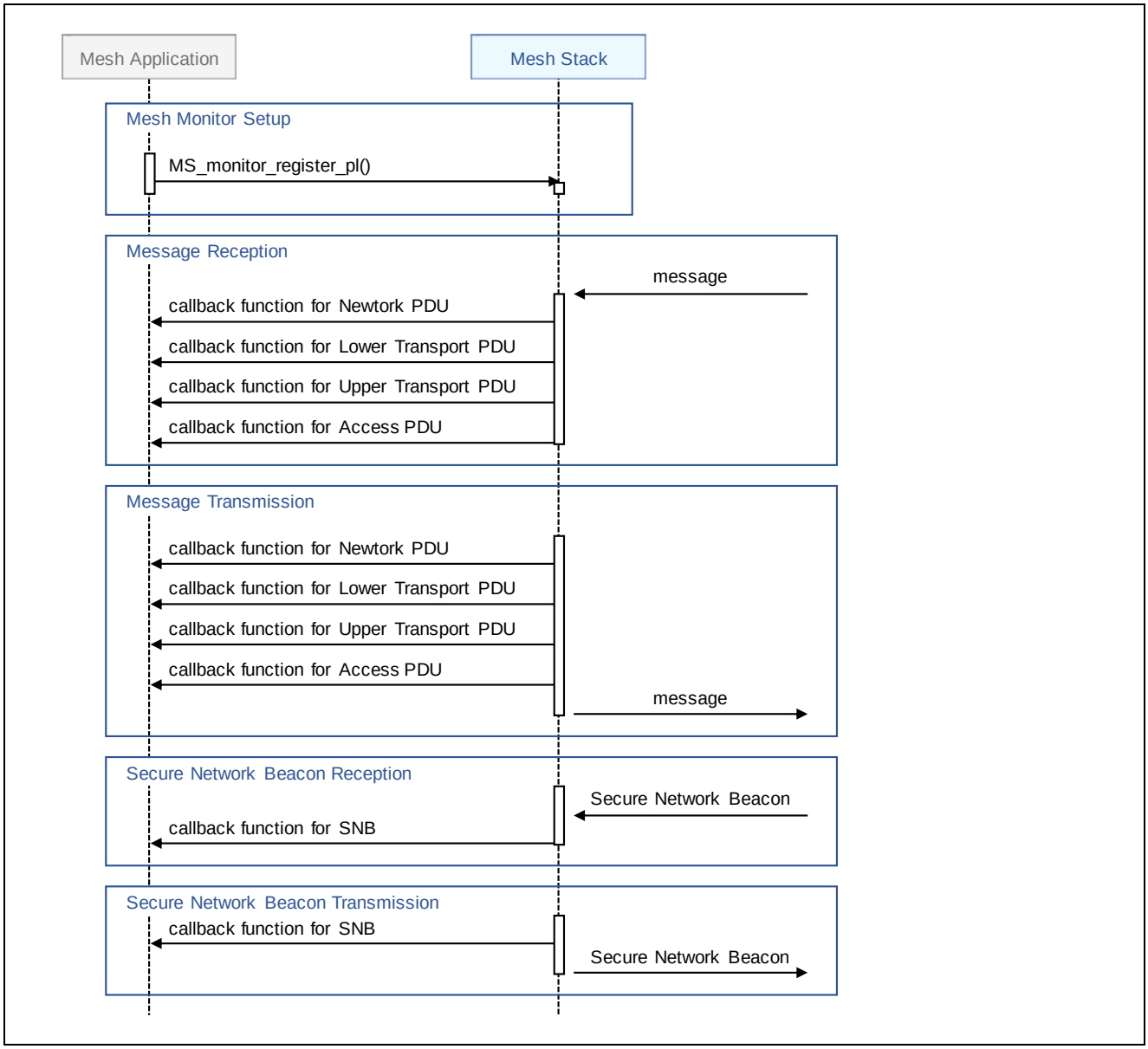

Figure 3-18 Mesh Monitoring

4. Appendix

4.1 Command Line Interface Program

Command Line Interface (CLI) is an interface to execute Mesh Stack API over serial interface from PC. Command Line Interface Program (mesh_cli) is included in the Bluetooth Mesh Stack Package.

By using this program, you can check wireless communication operation of Mesh Stack. In addition, you can refer to the implementation of this program as a example for using Mesh Stack API.

Figure 4-1 shows the example usage of Command Line Interface Program.

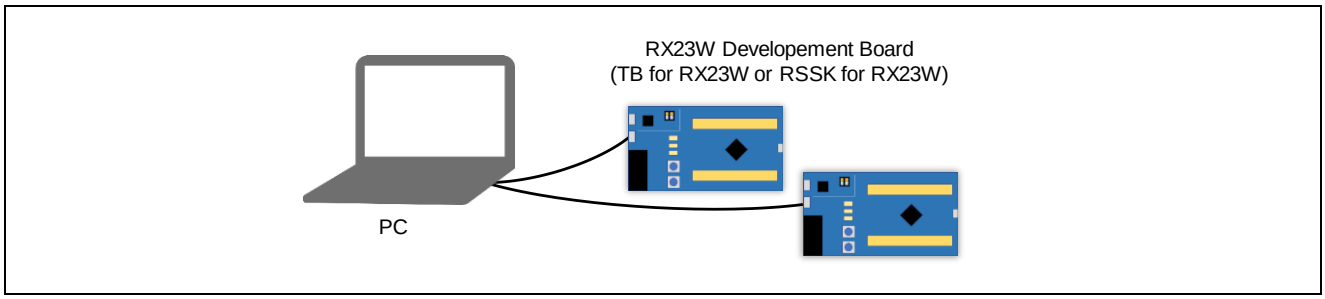

Figure 4-1 Example Usage of Command Line Interface Program

Figure 4-2 shows the example sequence of Command Line Interface. This program can work as both role such as Provisioning Client and Provisioning Server, Configuration Client and Configuration Server.

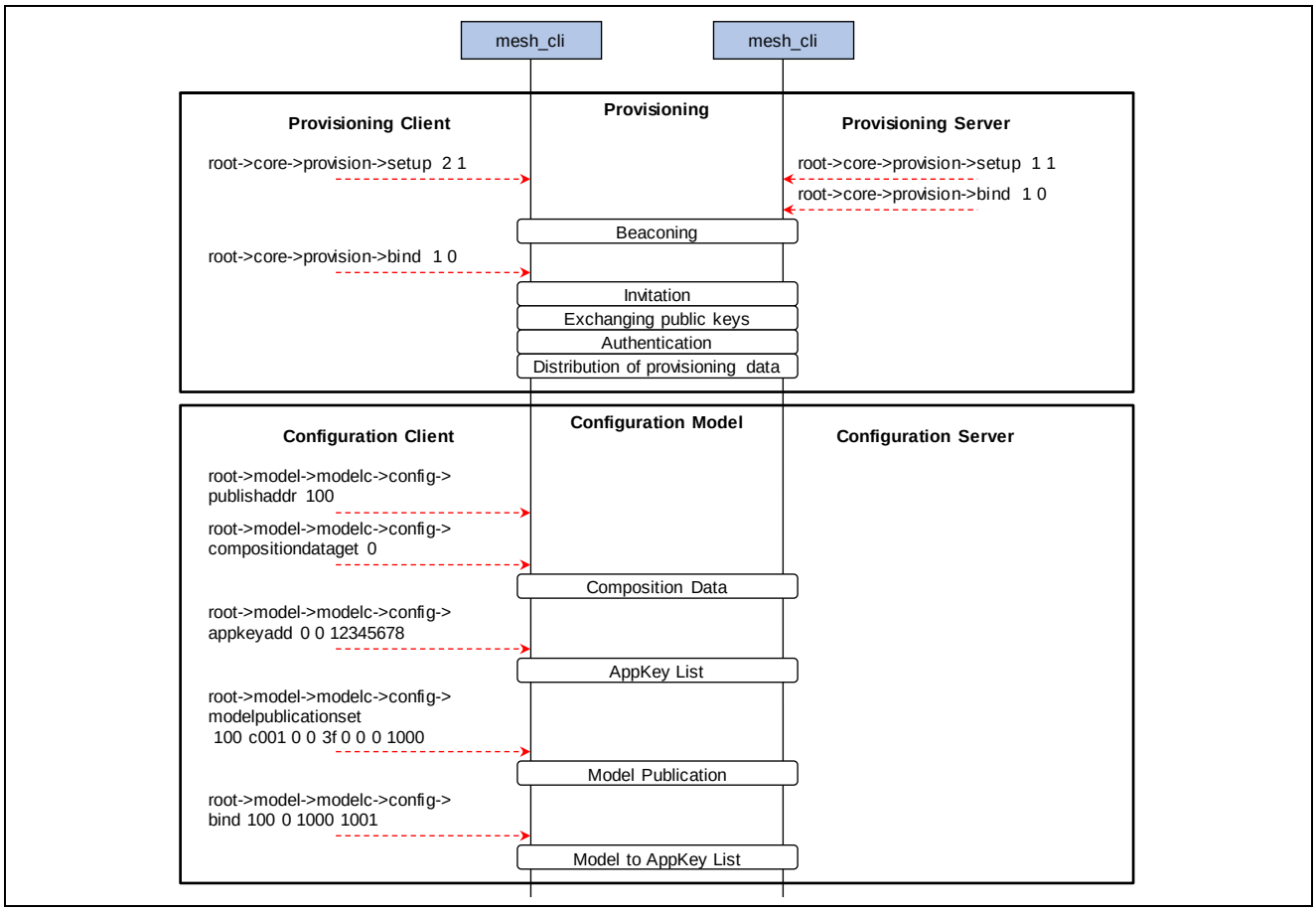

Figure 4-2 Example Sequence of Command Line Interface

Regarding the environment setup for building Command Line Interface, refer to Section 6 in "RX23W Group Bluetooth Mesh Stack Startup Guide" (R01AN4874) and use "mesh_cli" project generated in the workspace directory.

Target Board for RX23W and RSSK for RX23W have USB Serial Converter for communicating with PC. To operate Command Line Interface, use serial terminal tool on PC. (e.g. Tera Term)

Table 4-1 shows the serial port setting to communicate with Command Line Interface Program.

Table 4-1 Serial Port Setting

Item	Setting
Baud rate	115200 bps
Data	8 bits
Parity	none
Stop	1 bit
Flow Control	none

Regarding the specification of Command Line Interface, refer to "FITDemos\mesh_cli\mesh_cli_guide.pdf" included in the Bluetooth Mesh Stack Package.

Trademark and Copyright

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

crackle; AES-CCM, AES-128bit functionality

Revision History

Rev.	Date	Description	
1.00	Sep. 30, 2019	-	First edition
1.01	Nov. 29, 2019	P.4	Added Figure 1 "Basic Composition of Bluetooth Mesh Network"
		P.5	Added Section 1.4 "State"
		P.6	Added Section 1.6 "Message"
		P.12	Added Section 1.14 "Lifecycle of Mesh Device"
		P.26	Added Subsection 2.5.1 "Scheduler"
		P.33	Added Section 3.4 "Configuration"
		P.35	Added Section 3.5 "Friendship"
		P.39	Added Implementation for Generic OnOff Model in Section 3.6
		P.44	Added Section 3.7 "Vendor-Specific Mesh Model"
		P.45	Added Chapter 4 "Appendix"
1.10	Sep. 30, 2020	P.4	Merged "Network" Section with Section 1.1 "Node"
		P.4	Merged "Address" Section with Section 1.2 "Element"
		P.7	Added Figure 4 to Section 1.5 "Message"
		P.9	Removed "Lifecycle of Mesh Device" Section and added Figure 6
		P.10	Added Section 1.9 "Optional Features"
		P.12	Added Demo Project Composition to Section 2.1 "System Architecture"
		P.14	Added Notable Features of Mesh Sample Program to Section 2.2 "Mesh Application"
		P 14	Added Figure 13 to Section 2.2 "Mesh Application"
		P.22	Added Subsection 2.2.1.3 "Low Power"
		P.24	Added Subsection 2.2.1.4 "Mesh Monitoring"
		P.26	Added Table 5, Table 6, and Figure 24 to Subsection 2.2.2.2 "Configuration Model"
		P.29	Added Table 7, Table 8, and Figure 25 to Subsection 2.2.2.3 "Generic OnOff Model"
		P.30	Added Subsection 2.2.2.4 "Vendor Model"
		P.32	Added Table 11 to Section 2.3 "Bluetooth Mesh Stack"
		P.35	Added Subsection 2.4.3 "ADV Bearer Operation"
		P.36	Added Subsection 2.4.4 "GATT Bearer Operation"
		P.37	Removed "Bluetooth Protocol Stack" Section
		P.39	Added Section 2.6 "Mesh Sample Program Configuration"
		P.41	Added Section 2.7 "Bluetooth Bearer Wrapper Configuration"
		P.44	Added Section 2.8 "Mesh Driver Configuration"
		P.60	Removed "Vendor Specific Mesh Model"
		Overall	Updated some section order, description, figures, and source codes
1.20	Sep. 30, 2021	P.6	Added Subsection 1.5.2 "Foundation Models"
		P.6	Added Subsection 1.5.4 "Health Model"
		P.14	Moved sequence diagrams from Chapter 2 Bluetooth Mesh Stack Package" to Chapter 3 "Application Development"
		P.20	Added Subsection 2.2.3.2 "Health Model"
		P.26	Added Subsection 2.4.3 "Mesh GATT Services"
		P.47	Added Section 3.4 "Proxy"
		P.60	Added Section 3.7 "Health Model"
		P.69	Added Subsection 3.8.5 "Mesh Monitoring"
		Overall	Updated some section order, description, figures, and source codes

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/.