GR533x Mesh Simple On Off Models Example Application

Version: 1.1

Release Date: 2023-11-06

Shenzhen Goodix Technology Co., Ltd.

Trademarks and Permissions

GOODiX and other Goodix trademarks are trademarks of Shenzhen Goodix Technology Co., Ltd. All other trademarks and trade names mentioned in this document are the property of their respective holders.

Disclaimer

Information contained in this document is intended for your convenience only and is subject to change without prior notice. It is your responsibility to ensure its application complies with technical specifications.

Shenzhen Goodix Technology Co., Ltd. (hereafter referred to as “Goodix”) makes no representation or guarantee for this information, express or implied, oral or written, statutory or otherwise, including but not limited to representation or guarantee for its application, quality, performance, merchantability or fitness for a particular purpose. Goodix shall assume no responsibility for this information and relevant consequences arising out of the use of such information.

Without written consent of Goodix, it is prohibited to use Goodix products as critical components in any life support system. Under the protection of Goodix intellectual property rights, no license may be transferred implicitly or by any other means.

Shenzhen Goodix Technology Co., Ltd.

Headquarters: Floor 12-13, Phase B, Tengfei Industrial Building, Futian Free Trade Zone, Shenzhen, China

TEL: +86-755-33338828 Zip Code: 518000

Website: www.goodix.com

Preface

Purpose

This document provides an overview of the operation, verification, and critical code of the Simple On Off Models examples in GR533x Software Development Kit (SDK), to help users quickly get started with secondary development.

Audience

This document is intended for:

- Device user
- Developer
- Test engineer
- Technical support engineer

Release Notes

This document is the second release of *GR533x Mesh Simple On Off Models Example Application*, corresponding to GR533x System-on-Chip (SoC) series.

Revision History

Version	Date	Description
1.0	2023-10-18	Initial release
1.1	2023-11-06	Updated the approache for obtaining GRUart.

Contents

1 Introduction

To ensure the compatibility for message exchange between different kinds of Bluetooth Mesh devices, Bluetooth Special Interest Group (Bluetooth SIG) defines a series of generic and standard models for Bluetooth Mesh applications, including Generics, Sensors, Time and Scenes, and Lighting, which enable Bluetooth Mesh devices to control a peer Bluetooth device or obtain the device information.

If the standard models cannot meet your application requirements, you can customize models according to specific needs. For example, your application may require some new functions which are not supported by the standard models.

This document introduces how to develop and use custom models by taking the Simple On Off (SOO) Models for example.

Before getting started, you can refer to the following documents.

Table 1-1 Reference documents

Name	Description
GR533x Developer Guide	Introduces the software/hardware and quick start guide of GR533x System-on-Chips (SoCs).
Bluetooth Core Spec	Offers official Bluetooth standards and core specification from Bluetooth SIG.
Mesh Networking Specifications	Offers Bluetooth official Mesh profiles and specifications. Available at https://www.bluetooth.com/specifications/mesh-specifications/.
J-Link/J-Trace User Guide	Provides J-Link operational instructions. Available at https://www.segger.com/downloads/jlink/UM08001_JLink.pdf.
Keil User Guide	Offers detailed Keil operational instructions. Available at https://www.keil.com/support/man/docs/uv4/.

2 Overview of SOO Models

Goodix provides SOO Models for your reference, to demonstrate how to implement custom models.

SOO Models include:

- SOO Client Model: It sends messages to a server model to obtain or change the on/off state of the server model.
- SOO Server Model: It receives messages from the client model and responds with or changes the on/off state according to requests from the client model.

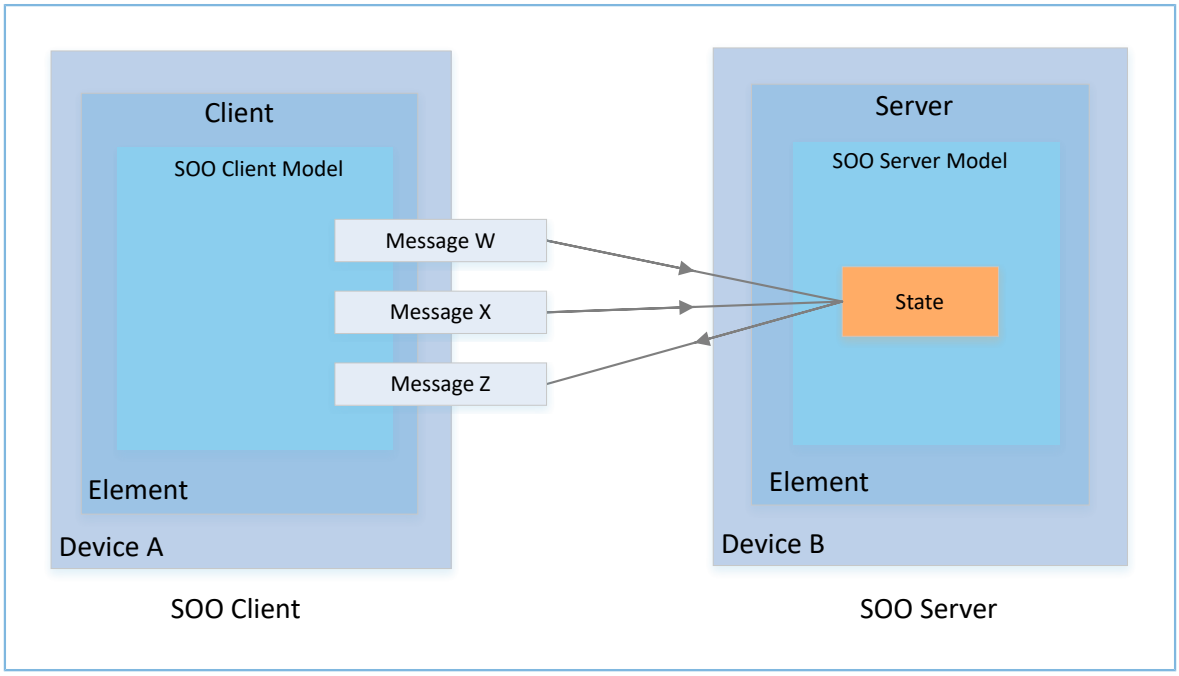

Figure 2-1 Message exchange between SOO Client Model and SOO Server Model

2.1 Published Message Type

The published message types supported by the SOO Client Model are listed in Table 2-1.

Table 2-1 Published message type supported by SOO Client Model

Message Type	Description
SIMPLE_ONOFF_OPCODE_GET	Obtain the current on/off state of the server.
SIMPLE_ONOFF_OPCODE_SET	Set the current on/off state of the server and wait for response.
SIMPLE_ONOFF_OPCODE_SET_UNACKNOWLEDGED	Set the current on/off state of the server without waiting for response.

The published message types supported by the SOO Server Model are listed in Table 2-2.

Table 2-2 Published message type supported by SOO Server Model

Message Type	Description
SIMPLE_ONOFF_OPCODE_STATUS	Publish the current on/off state of the server.

3 Example Running

This chapter introduces how to run the SOO Models example projects and test/verify the running results.

SOO Models example projects include:

- SOO Client Model example project: in SDK_Folder\projects\mesh\Vendor\mesh_app_simple_on_off_client
- SOO Server Model example project: in SDK_Folder\projects\mesh\Vendor\mesh_app_simple_on_off_server

Note:

SDK_Folder is the root directory of GR533x SDK.

3.1 Preparation

Perform the following tasks before running the SOO Models example projects.

- **Hardware preparation**

Table 3-1 Hardware preparation

Name	Description
J-Link debug probe	JTAG emulator launched by SEGGER. For more information, visit www.segger.com/products/debug-probes/j-link/.
Development board	GR5331 Starter Kit Board (GR5331 SK Board) (2 boards in total)
Connection cable	USB Type-C cable

- **Software preparation**

Table 3-2 Software preparation

Name	Description
Windows	Windows 7/Windows 10
J-Link driver	A J-Link driver. Available at www.segger.com/downloads/jlink/.
Keil MDK5	An integrated development environment (IDE). Available at www.keil.com/download/product/.
GRMesh (Android)	A Mesh debugging tool.
GRUart	A serial port debugging tool. Available at www.goodix.com/en/download?objectId=43&objectType=software.

3.2 Hardware Connection

Use two GR5331 SK Boards to serve as the SOO Client and the SOO Server respectively.

Connect the GR5331 SK Boards to a PC with USB Type-C cables.

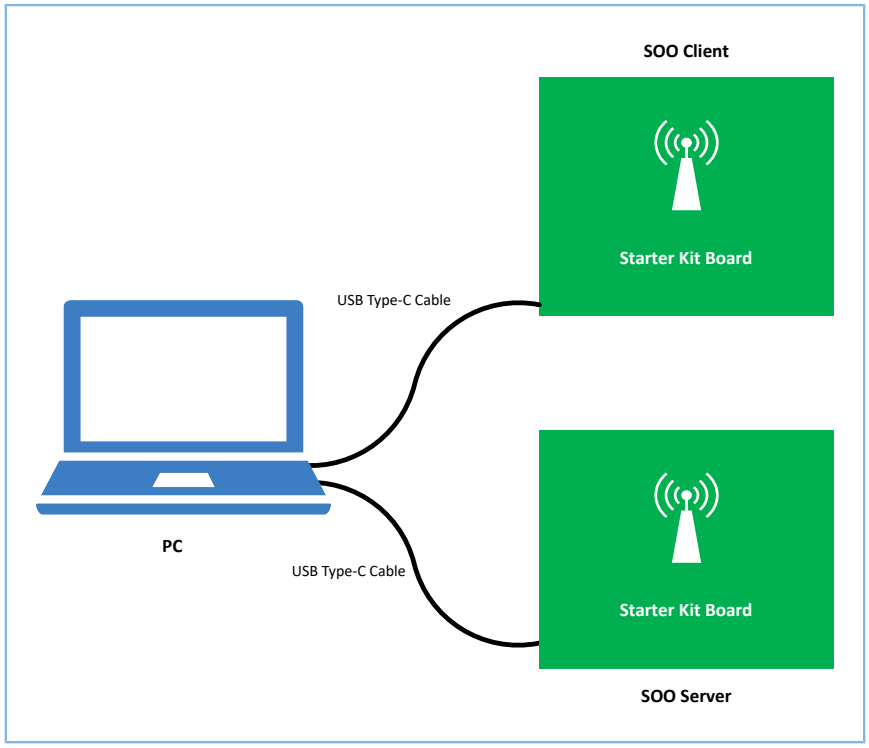

Figure 3-1 Hardware connection

3.3 Firmware Download

Compile projects mesh_app_simple_on_off_client and mesh_app_simple_on_off_server in Keil, and download the generated target files to the two GR5331 SK Boards.

3.4 Firmware Running

Press **S9** (**RESET** button) on the GR5331 SK Board to run the firmware after firmware download completes.

3.5 Configuration of Serial Port Tool

Start GRUart, and configure the parameters as follows.

Table 3-3 Configuring parameters on GRUart

PortName	BaudRate	DataBits	Parity	StopBits	Flow Control
Select on demand	115200	8	None	1	Uncheck

The figure below shows the interface of GRUart after configuration completes.

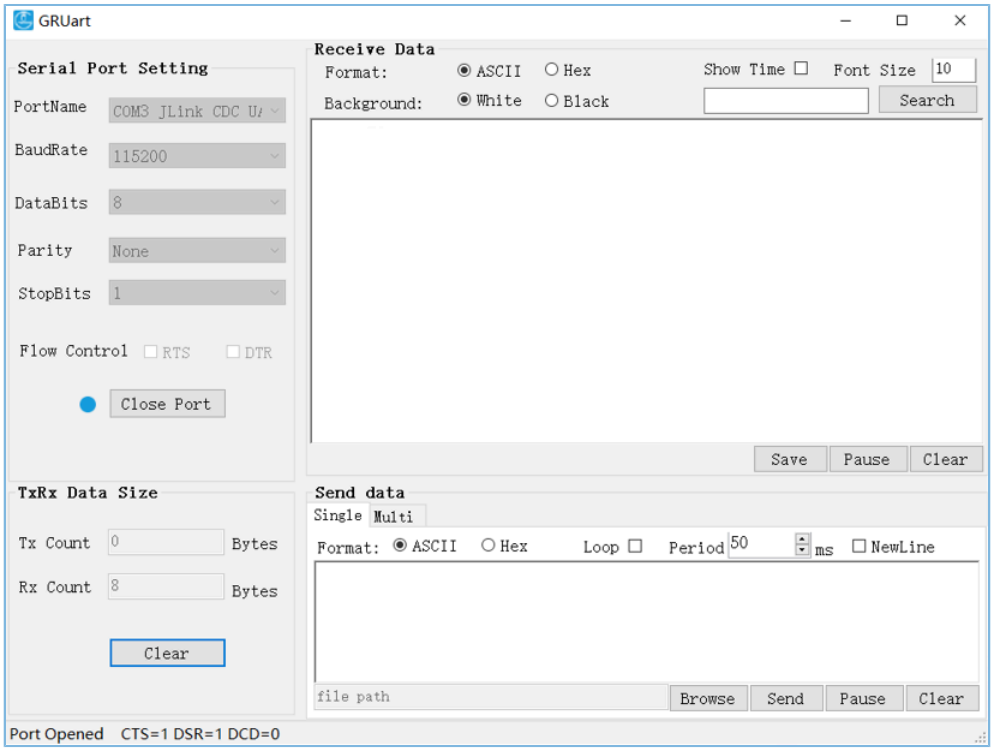

Figure 3-2 GRUart configuration

3.6 Test and Verification

After the previous preparations are ready, you can test and verify communication between the SOO Models according to the following steps.

1. Configure the SOO Client and the SOO Server through GRMesh.
2. Verify message interactions between the SOO Client and the SOO Server by checking the logs printed on GRUart.

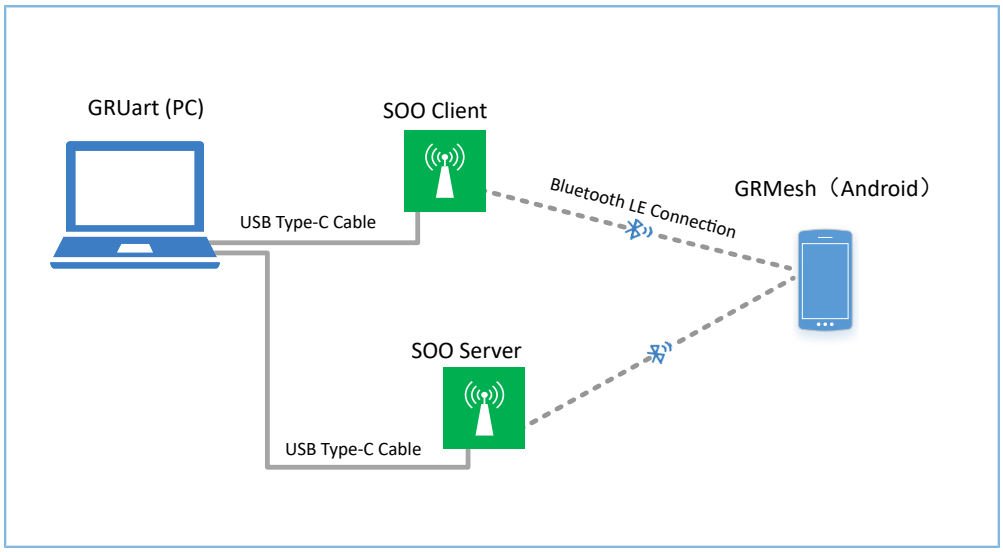

Figure 3-3 SOO Models test diagram

3.6.1 Device Configuration through GRMesh

Configure the SOO Client and the SOO Server through GRMesh as follows:

1. Power on the SOO Client (Goodix_Simple_Switch) and the SOO Server (Goodix_Simple_Light).
2. Provision Goodix_Simple_Switch on GRMesh.

   (1). To add a new device, tap **+** in the upper-right corner on the home page of GRMesh to start scanning. Devices with the advertising name of **Goodix_Simple_Switch** and **Goodix_Simple_Light** are discovered.

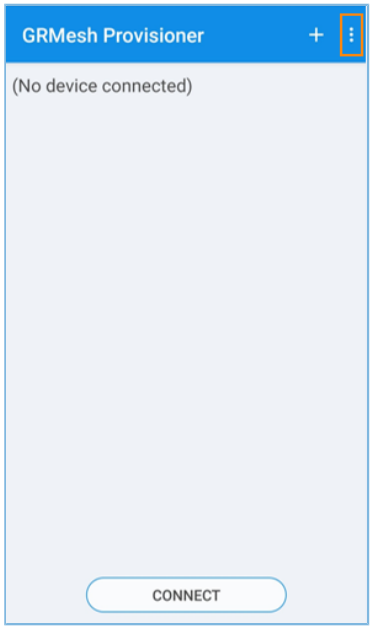

Figure 3-4 To add a new device

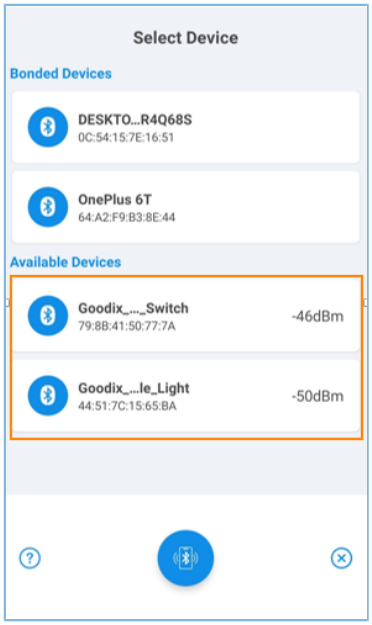

Figure 3-5 Discovering new devices

   (2). Tap **Goodix_Simple_Switch** > **IDENTIFY** > **PROVISION**.

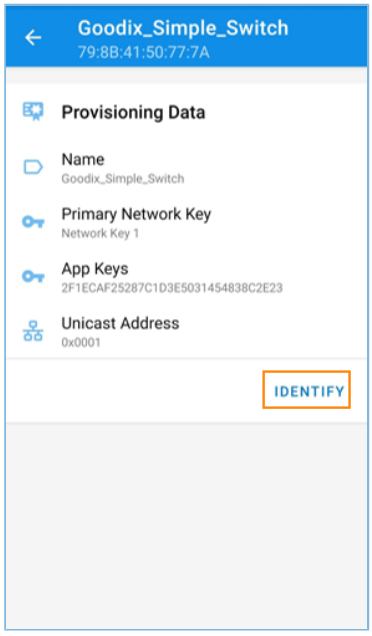

Figure 3-6 Tapping **IDENTIFY**

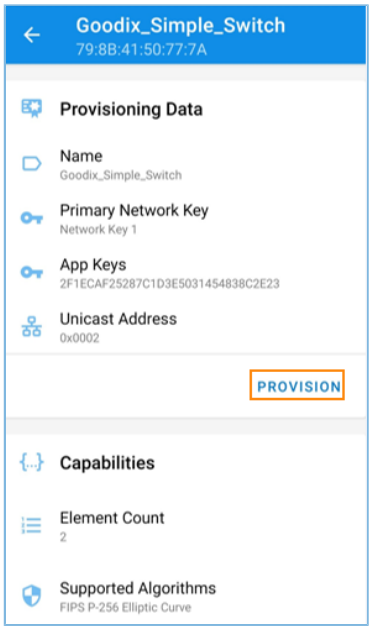

Figure 3-7 Tapping **PROVISION**

(3). On the **Select OOB Type** pane, select **No OOB** from the drop-down list and then tap **OK** to complete provisioning of the Goodix_Simple_Switch. Then the SOO Client is added to the Mesh network.

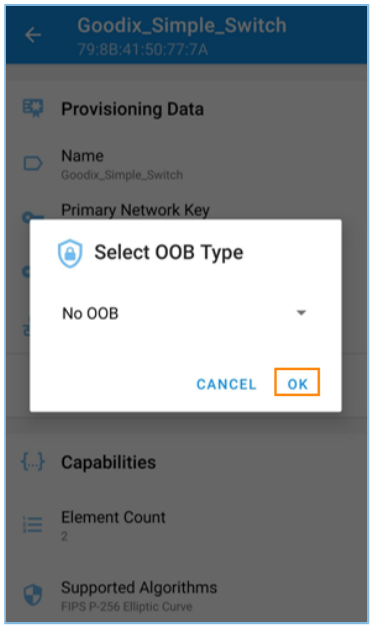

Figure 3-8 Tapping **OK**

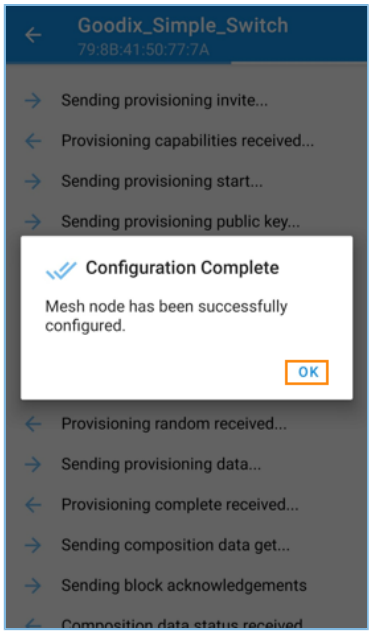

Figure 3-9 Completing provisioning

3. In the **GRMesh Provisioner** interface, tap **Goodix_Simple_Switch** > **Element:0x0002** > **Vendor Model** to configure **Vendor Model** of the first element of Goodix_Simple_Switch in the Mesh network.

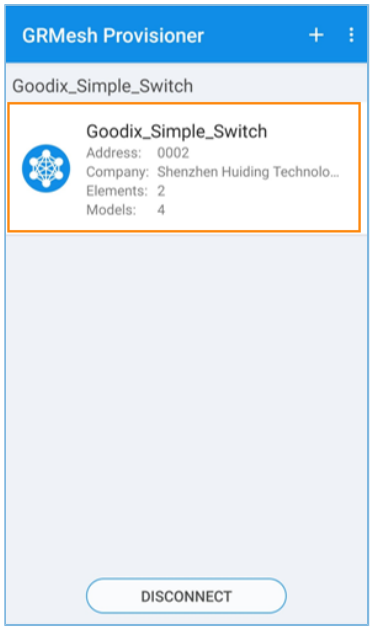

Figure 3-10 Configuring Goodix_Simple_Switch

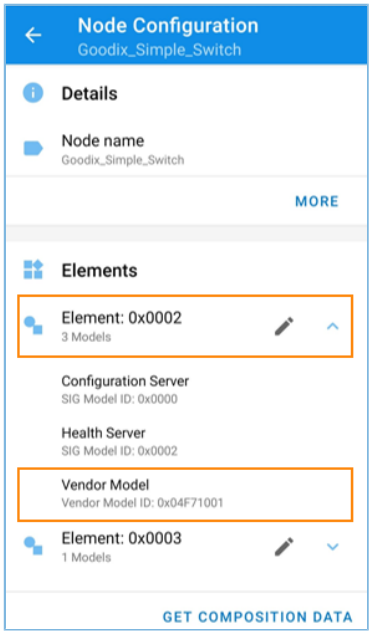

Figure 3-11 Tapping **Vendor Model**

(1). In the **Vendor Model** interface, configure **Bound App Keys** as **index 0**, and **Publish Address** as the unicast address **0x0004** (corresponding to the address of the first element of Goodix_Simple_Light).

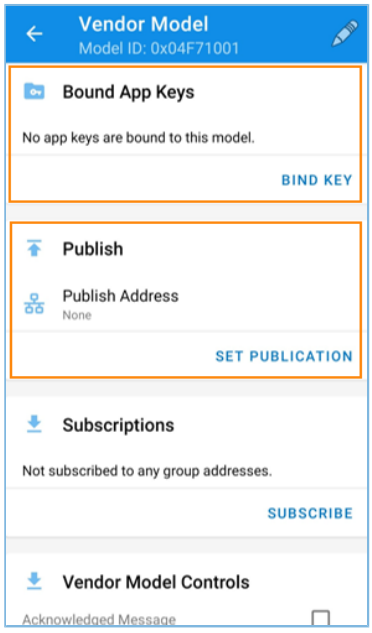

Figure 3-12 **Vendor Model** interface

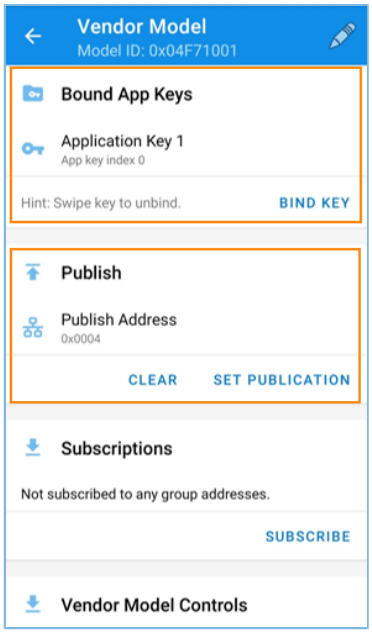

Figure 3-13 Configuring **Bound App Keys** and **Publish Address**

4. Repeat Step 2 and Step 3 to provision Goodix_Simple_Light, and configure **Publish Address** of **Vendor Model** of the first element as the unicast address **0x0002** (corresponding to the address of the first element of Goodix_Simple_Switch).

5. After provisioning of the SOO Client and the SOO Server, you can view device information in the **GRMesh Provisioner** interface, as shown below.

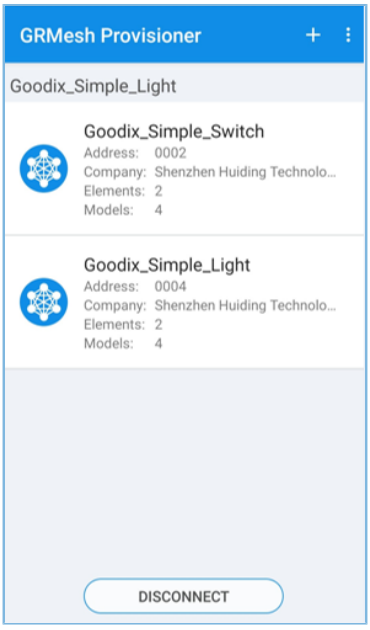

Figure 3-14 Completing provisioning

3.6.2 Verification of SOO Models through GRUart

Test communication interaction between the SOO Client and the SOO Server through GRUart as follows:

1. Open two GRUart windows: one for SOO Client and the other for SOO Server.
2. Input **0001** to GRUart (SOO Client).
   (1). GRUart (SOO Client) sends the **SIMPLE_ONOFF_OPCODE_SET** message (with the on command).
   (2). GRUart (SOO Server) sets the current state to “on” and sends a response after receiving the message.

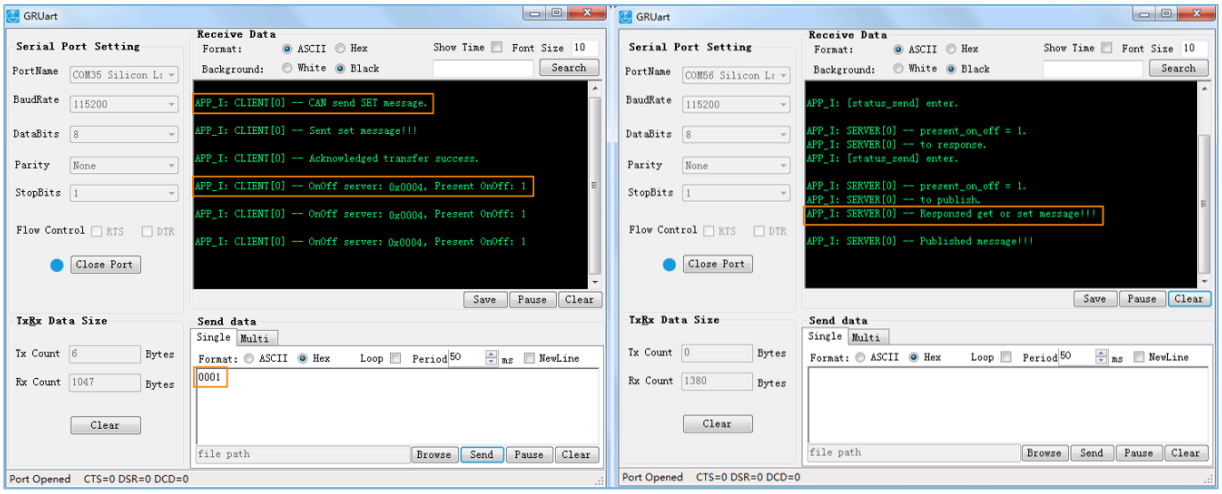

Figure 3-15 Message exchange between SOO Client (left) and SOO Server (right)

3. Input **0002** to GRUart (SOO Client).
   (1). GRUart (SOO Client) sends the **SIMPLE_ONOFF_OPCODE_SET** message (with the off command).

(2). GRUart (SOO Server) sets the current state to “off” and sends a response after receiving the message.

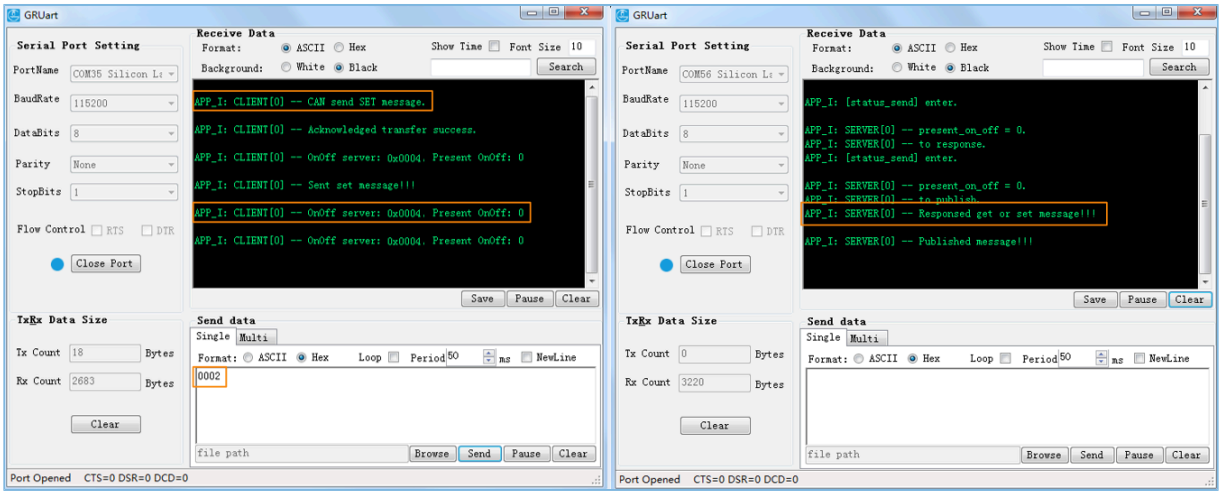

Figure 3-16 Message exchange between SOO Client (left) and SOO Server (right)

4 Application Details

This chapter introduces the project directory of the SOO Models examples and the implementation code of critical functionalities.

4.1 Project Directory

GR533x SDK provides source code and project files of the SOO Models examples, to help users implement custom SOO Models by referring to the example projects.

4.1.1 SOO Client Model Project

The source code and project file of the SOO Client Model example are in SDK_`Folder\projects\mesh\Vendor\mesh_app_simple_on_off_client`, and the project file is in the Keil_5 folder.

Open *mesh_app_simple_on_off_client.uvprojx* in Keil, to view the project directory structure of the SOO Client Model example. Details of the files are listed below.

Table 4-1 File description of mesh_app_simple_on_off_client

Group	File	Description
gr_models	simple_onoff_client.c	Implement SOO Client Model.
user_callback	user_gap_callback.c	Obtain connection and disconnection events.
	user_mesh_callback.c	Obtain provisioning events.
user_platform	user_periph_setup.c	Configure App logs, serial port parameters, and device address.
user_app	main.c	Contain the main() function.
	user_app.c	Register SOO Client Model and process application logics.
	custom_config.h	Common configurations for application projects
	mesh_stack_config.h	Configurations related to Mesh protocols

4.1.2 SOO Server Model Project

The source code and project file of the SOO Server Model example are in SDK_`Folder\projects\mesh\Vendor\mesh_app_simple_on_off_server`, and the project file is in the Keil_5 folder.

Open *mesh_app_simple_on_off_server.uvprojx* in Keil, to view the project directory structure of the SOO Server Model example. Details of the files are listed below.

Table 4-2 File description of mesh_app_simple_on_off_server

Group	File	Description
gr_models	simple_onoff_server.c	Implement SOO Server Model.
user_callback	user_gap_callback.c	Obtain connection and disconnection events.
	user_mesh_callback.c	Obtain provisioning events.
user_platform	user_periph_setup.c	Configure App logs, serial port parameters, and device address.

Group	File	Description
user_app	main.c	Contain the main() function.
	user_app.c	Register SOO Server Model and process application logics.
	custom_config.h	Common configurations for application projects
	mesh_stack_config.h	Configurations related to Mesh protocols

4.2 Code Description

4.2.1 Message Processing of SOO Models

This section demonstrates how to implement messages processing for custom models by introducing the implementation code for message processing of SOO Models.

4.2.1.1 SOO Client Model

The example code is in SDK_Folder\components\mesh\models\Vendor\simple_onoff\src\simple_onoff_client.c.

4.2.1.1.1 Processable Message List

The Mesh messages to be received and processed by the SOO Client Model are listed in the processable message list simple_on_off_client_opcode_list.

```
static const uint16_t simple_on_off_client_opcode_list[] =
{
    SIMPLE_ONOFF_OPCODE_STATUS,
};
```

4.2.1.1.2 Callback Function Handling List

The callback function handling list simple_on_off_client_msg_cb consists of three members:

- cb_rx: message callback function received by the SOO Client Model
- cb_sent: callback function published or returned by the SOO Client Model to indicate message state
- cb_publish_period: callback function when the periodic publish state of the SOO Client Model changes

```
static const mesh_model_cb_t simple_on_off_client_msg_cb =
{
    .cb_rx             = simple_on_off_client_rx_cb,
    .cb_sent           = simple_on_off_client_sent_cb,
    .cb_publish_period  = NULL,
};
```

The code snippet for processing of Mesh messages received by simple_on_off_client_rx_cb() is as follows.

```
static void simple_on_off_client_rx_cb(mesh_model_msg_ind_t *p_model_msg, void *p_args)
{
    uint16_t company_opcode = p_model_msg->opcode.company_opcode;
```

```
    mesh_opcode_handler_cb_t handler = m_opcode_handlers[company_opcode
                                                - SIMPLE_ONOFF_OPCODE_STATUS].handler;

    if (NULL ! = handler)
    {
        handler(p_model_msg, p_args);
    }
}

static const mesh_opcode_handler_t m_opcode_handlers[] =
{
    {SIMPLE_ONOFF_OPCODE_STATUS, status_handle},
};
static void status_handle(const mesh_model_msg_ind_t *p_rx_msg,  void *p_args)
{
    simple_onoff_client_t * p_client = (simple_onoff_client_t *) p_args;

    if (p_rx_msg->msg_len == SIMPLE_ONOFF_STATUS_LEN)
    {
        p_client->settings.p_callbacks->onoff_status_cb(p_client, p_rx_msg,
                                        (simple_onoff_status_msg_pkt_t *)p_rx_msg->msg);
    }
}
```

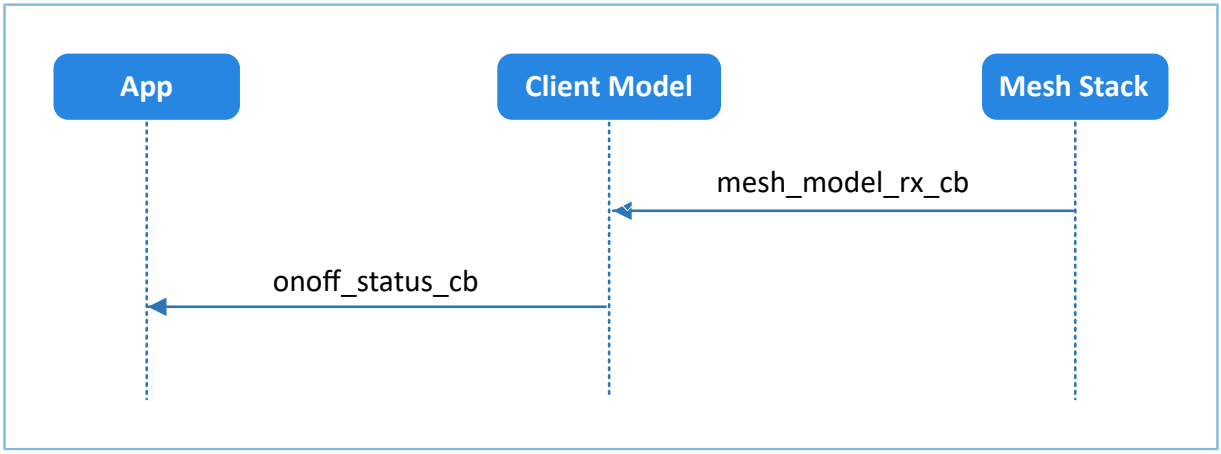

Figure 4-1 SIMPLE_ONOFF_OPCODE_STATUS message processing

4.2.1.1.3 Processing of To-be-sent Message

The to-be-sent messages of the SOO Client Model are divided into three types: GET, SET, and SET_UNACK messages.

- SIMPLE_ONOFF_OPCODE_SET message

  To send this message, the second parameter of mesh_model_publish() shall carry reliable_info, which contains opcode and wait time of the massage to be replied.

```
uint16_t simple_onoff_client_set(simple_onoff_client_t * p_client,
                                        const simple_onoff_set_msg_pkt_t * p_params)
{
    mesh_model_send_info_t model_msg_send;
    bool reliable_trans_state = false;
    uint8_t tx_hdl = SIMPLE_ONOFF_CLIENT_SET_SEND_TX_HDL +
                     p_client->model_instance_index * SIMPLE_ONOFF_CLIENT_TX_HDL_TOTAL;

    mesh_model_reliable_info_t reliable_info =
```

```
    {
        .reply_opcode = MESH_ACCESS_OPCODE_VENDOR(SIMPLE_ONOFF_OPCODE_STATUS,
                                                  SIMPLE_ONOFF_COMPANY_ID),
        .status_cb = p_client->settings.p_callbacks->ack_transaction_status_cb,
        .timeout_ms = p_client->settings.timeout_ms,
    };

    if (p_client == NULL || p_params == NULL)
    {
        return MESH_ERROR_SDK_INVALID_PARAM;
    }

    if (MESH_ERROR_NO_ERROR ==
                            mesh_model_reliable_trans_is_on(p_client->model_lid,
                                                            &reliable_trans_state))
    {
        if (reliable_trans_state)
        {
            return MESH_ERROR_SDK_RELIABLE_TRANS_ON;
        }
        else
        {
            message_create(&model_msg_send, p_client->model_lid,
                           SIMPLE_ONOFF_OPCODE_SET,
                           tx_hdl, (uint8_t *)p_params, SIMPLE_ONOFF_SET_LEN);

            return mesh_model_publish(&model_msg_send, &reliable_info);
        }
    }
    else
    {
        return MESH_ERROR_SDK_INVALID_PARAM;
    }
}
```

After the SET message is sent, the SOO Client Model notifies users of the operation state using a callback function (methods 1 and 2 in Figure 4-2). Users can call simple_onoff_client_setget_cancel() to abort the operation, and the SOO Client Model will also notify users of the operation state using a callback function (method 3 in Figure 4-2).

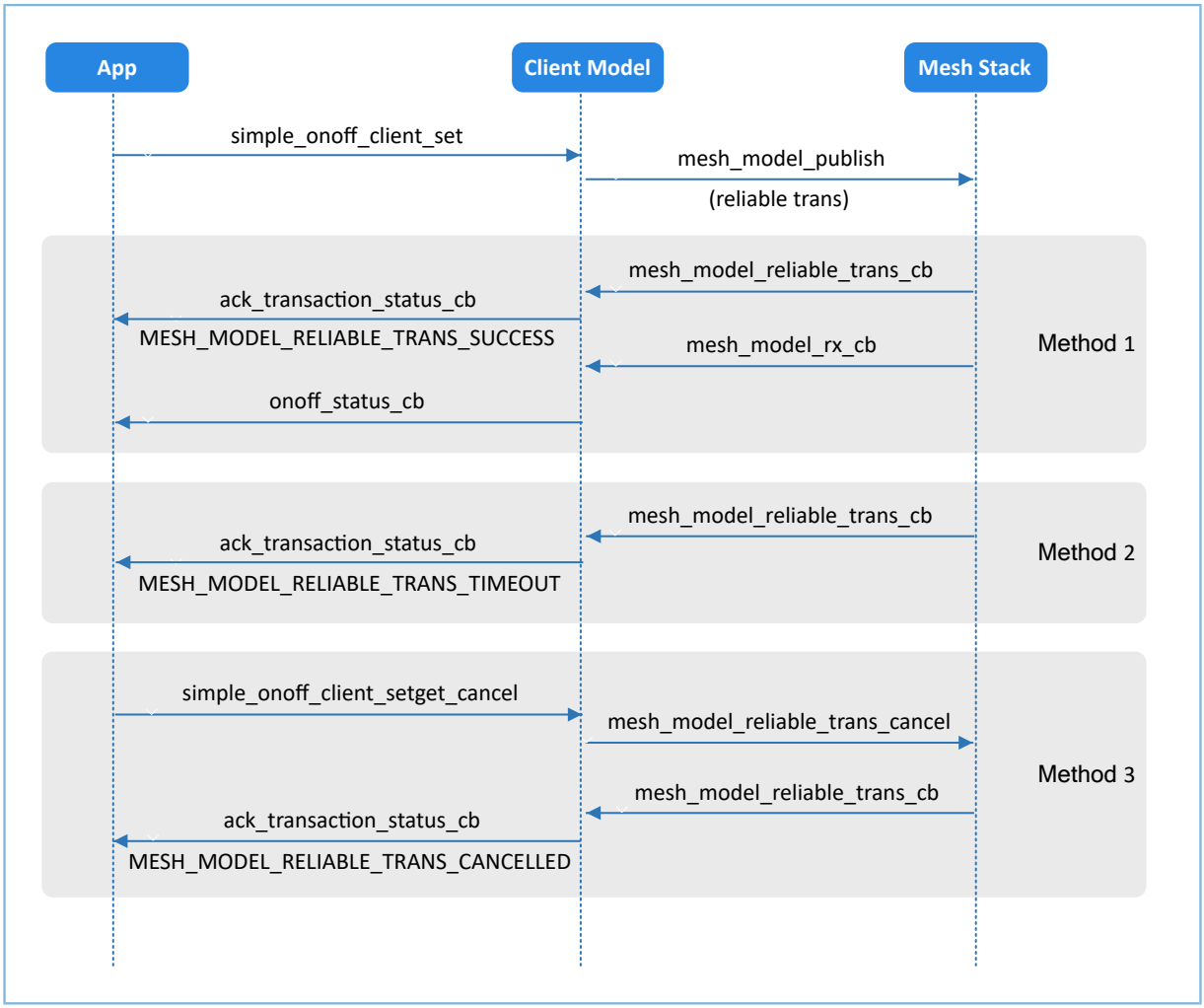

Figure 4-2 SIMPLE_ONOFF_OPCODE_SET message sending

- SIMPLE_ONOFF_OPCODE_SET_UNACKNOWLEDGED message

  To send this message, the second parameter of mesh_model_publish() shall be set to NULL. If no response from the SOO Server Model is required, the SOO Client Model can send this message.

```
uint16_t simple_onoff_client_set_unack(simple_onoff_client_t * p_client,
                                       const simple_onoff_set_msg_pkt_t * p_params)
{
    mesh_model_send_info_t model_msg_send;
    uint8_t tx_hdl = SIMPLE_ONOFF_CLIENT_SET_UNRELIABLE_SEND_TX_HDL
                     + p_client->model_instance_index * SIMPLE_ONOFF_CLIENT_TX_HDL_TOTAL;

    if (p_client == NULL || p_params == NULL)
    {
        return MESH_ERROR_SDK_INVALID_PARAM;
    }

    message_create(&model_msg_send, p_client->model_lid,
                   SIMPLE_ONOFF_OPCODE_SET_UNACKNOWLEDGED,
                   tx_hdl, (uint8_t *)p_params, SIMPLE_ONOFF_SET_LEN);

    return mesh_model_publish(&model_msg_send, NULL);
```

```
}
```

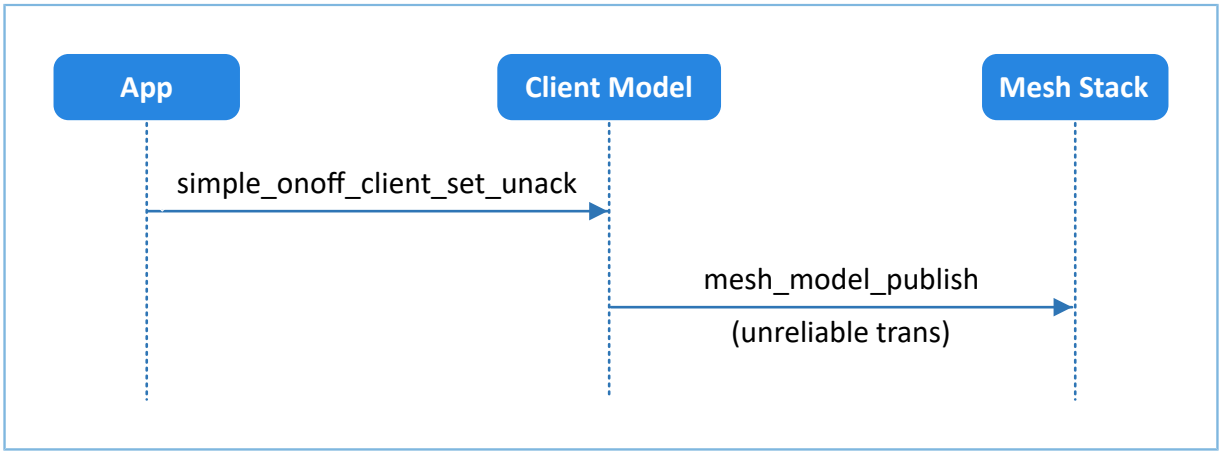

Figure 4-3 SIMPLE_ONOFF_OPCODE_SET_UNACKNOWLEDGED message sending

- SIMPLE_ONOFF_OPCODE_GET message

  To send this message, the second parameter of mesh_model_publish() shall carry reliable_info, which contains opcode and wait time of the massage to be replied.

```
uint16_t simple_onoff_client_get(simple_onoff_client_t * p_client)
{
    mesh_model_send_info_t model_msg_send;
    bool reliable_trans_state = false;
    uint8_t tx_hdl = SIMPLE_ONOFF_CLIENT_GET_SEND_TX_HDL
          + p_client->model_instance_index * SIMPLE_ONOFF_CLIENT_TX_HDL_TOTAL;

    mesh_model_reliable_info_t reliable_info =
    {
        .reply_opcode = MESH_ACCESS_OPCODE_VENDOR(SIMPLE_ONOFF_OPCODE_STATUS,
                                                  SIMPLE_ONOFF_COMPANY_ID),
        .status_cb = p_client->settings.p_callbacks->ack_transaction_status_cb,
        .timeout_ms = p_client->settings.timeout_ms,
    };

    if (p_client == NULL)
    {
        return MESH_ERROR_SDK_INVALID_PARAM;
    }

    if (MESH_ERROR_NO_ERROR == mesh_model_reliable_trans_is_on(
                                                   p_client->model_lid,
                                                   &reliable_trans_state))
    {
        if (reliable_trans_state)
        {
            return MESH_ERROR_SDK_RELIABLE_TRANS_ON;
        }
        else
        {
            message_create(&model_msg_send, p_client->model_lid,
                           SIMPLE_ONOFF_OPCODE_GET, tx_hdl, NULL, 0);

            return mesh_model_publish(&model_msg_send, &reliable_info);
        }
    }
```

```
    else
    {
        return MESH_ERROR_SDK_INVALID_PARAM;
    }
}
```

After the GET message is sent, the SOO Client Model notifies users of the operation state using a callback function (methods 1 and 2 in Figure 4-4). Users can call simple_onoff_client_setget_cancel() to abort the operation, and the SOO Client Model will also notify users of the operation state using a callback function (method 3 in Figure 4-4).

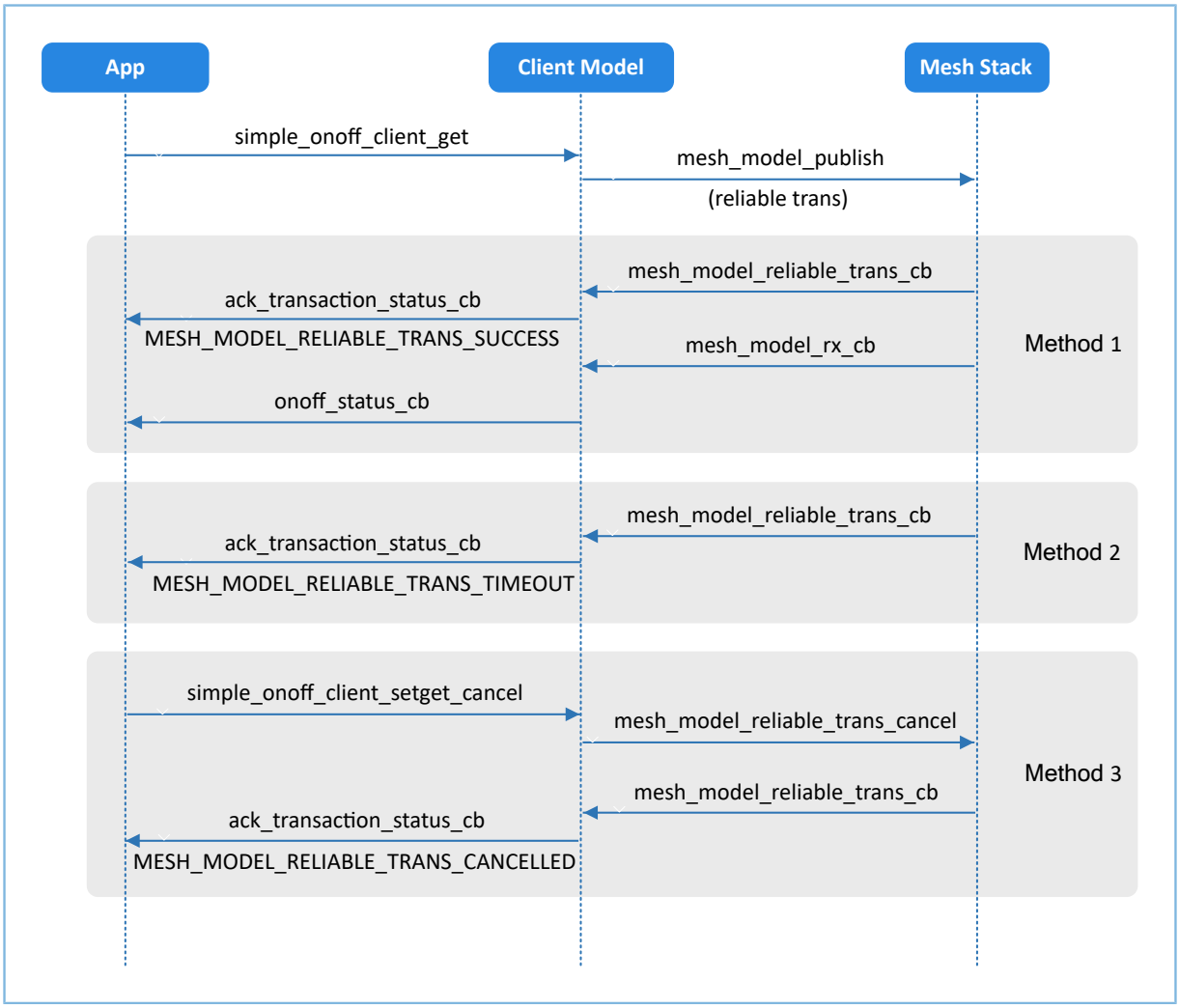

Figure 4-4 SIMPLE_ONOFF_OPCODE_GET message sending

4.2.1.2 SOO Server Model

The example code is in SDK_Folder\components\mesh\models\Vendor\simple_onoff\src\simple_onoff_server.c.

4.2.1.2.1 Processable Message List

The Mesh messages to be received and processed by the SOO Server Model are listed in the processable message list simple_on_off_server_opcode_list.

```
static const uint16_t simple_on_off_server_opcode_list[] =
{
    SIMPLE_ONOFF_OPCODE_GET,
    SIMPLE_ONOFF_OPCODE_SET,
    SIMPLE_ONOFF_OPCODE_SET_UNACKNOWLEDGED,
};
```

4.2.1.2.2 Callback Function Handling List

The callback function handling list simple_on_off_server_msg_cb consists of three members:

- cb_rx: message callback function received by the SOO Server Model
- cb_sent: callback function published or returned by the SOO Server Model to indicate message state
- cb_publish_period: callback function when the periodic publish state of the SOO Server Model changes

```
static const mesh_model_cb_t simple_on_off_server_msg_cb =
{
    .cb_rx             = simple_on_off_server_rx_cb,
    .cb_sent           = simple_on_off_server_sent_cb,
    .cb_publish_period  = NULL,
};
```

The code snippet for processing of Mesh messages received by simple_on_off_server_rx_cb() is as follows.

```
static void simple_on_off_server_rx_cb(mesh_model_msg_ind_t *p_model_msg, void *p_args)
{
    uint16_t company_opcode = p_model_msg->opcode.company_opcode;

    mesh_opcode_handler_cb_t handler = m_opcode_handlers[company_opcode -
                                            SIMPLE_ONOFF_OPCODE_GET].handler;

    if (NULL ! = handler)
    {
        handler(p_model_msg, p_args);
    }
}

static const mesh_opcode_handler_t m_opcode_handlers[] =
{
    {SIMPLE_ONOFF_OPCODE_GET, handle_get_cb},
    {SIMPLE_ONOFF_OPCODE_SET, handle_set_cb},
    {SIMPLE_ONOFF_OPCODE_SET_UNACKNOWLEDGED, handle_set_cb},
};
```

- handle_set_cb() can process the following messages:
    - SIMPLE_ONOFF_OPCODE_SET
    - SIMPLE_ONOFF_OPCODE_SET_UNACKNOWLEDGED

```
static void handle_set_cb(const mesh_model_msg_ind_t *p_rx_msg,  void *p_args)
{
    uint32_t send_status = MESH_ERROR_NO_ERROR;
    simple_onoff_server_t  * p_server = (simple_onoff_server_t  *) p_args;
```

```
    APP_LOG_INFO("SERVER[%d] -- Receive message, want to set on-off state!!!" , p_server-
>model_instance_index);

    simple_onoff_set_msg_pkt_t * p_msg_params_packed = (simple_onoff_set_msg_pkt_t *)
 p_rx_msg->msg;
    simple_onoff_status_msg_pkt_t out_data =
    {
        .present_on_off = p_msg_params_packed->on_off,
    };

    if (set_params_validate(p_rx_msg, p_msg_params_packed))
    {
        if (model_tid_validate(&p_server->tid_filter, p_rx_msg, p_msg_params_packed->tid))
        {
            p_server->client_address = p_rx_msg->src;
            p_server->settings.p_callbacks->onoff_cbs.set_cb(p_server->model_instance_index,
 p_msg_params_packed->on_off);

            // response
            if (SIMPLE_ONOFF_OPCODE_SET == p_rx_msg->opcode.company_opcode)
            {
                send_status = status_send(p_server, p_rx_msg, &out_data);
                if (MESH_ERROR_NO_ERROR ! = send_status)
                {
                    APP_LOG_WARNING("SERVER[%d] -- Response status faild, error code: 0x
%04x", p_server->model_instance_index, send_status);
                }
            }

            // publish
            send_status = status_send(p_server, NULL, &out_data);
            if (MESH_ERROR_NO_ERROR ! = send_status)
            {
                APP_LOG_WARNING("SERVER[%d] -- Publish status faild, error code: 0x%04x",
 p_server->model_instance_index, send_status);
            }
        }
    }
}
static uint32_t status_send(simple_onoff_server_t * p_server,
                                const mesh_model_msg_ind_t *p_rx_msg,
                                const simple_onoff_status_msg_pkt_t * p_params)
{
    simple_onoff_status_msg_pkt_t msg_pkt;
    uint8_t tx_hdl = (NULL == p_rx_msg) ? SIMPLE_ONOFF_SERVER_PUBLISH_SEND_TX_HDL +
 p_server->model_instance_index * SIMPLE_ONOFF_SERVER_TX_HDL_TOTAL
                                        : SIMPLE_ONOFF_SERVER_RSP_SEND_TX_HDL + p_server-
>model_instance_index * SIMPLE_ONOFF_SERVER_TX_HDL_TOTAL;

    if (p_params->present_on_off > SIMPLE_ONOFF_MAX)
    {
        return MESH_ERROR_SDK_INVALID_PARAM;
    }

    msg_pkt.present_on_off = p_params->present_on_off;

    mesh_model_send_info_t msg_send =
    {
        .model_lid = p_server->model_lid,
        .opcode = MESH_ACCESS_OPCODE_VENDOR(SIMPLE_ONOFF_OPCODE_STATUS,
 SIMPLE_ONOFF_COMPANY_ID),
        .tx_hdl = tx_hdl,
```

```
        .p_data_send = (uint8_t *) &msg_pkt,
        .data_send_len = SIMPLE_ONOFF_STATUS_LEN,
        .dst = (NULL == p_rx_msg) ? MESH_INVALID_ADDR : p_rx_msg->src,
        .appkey_index = (NULL == p_rx_msg) ? MESH_INVALID_KEY_INDEX : p_rx_msg-
>appkey_index,
    };

    APP_LOG_INFO("SERVER[%d] -- present_on_off = %d." , p_server->model_instance_index,
 msg_pkt.present_on_off);

    if (NULL == p_rx_msg)
    {

        APP_LOG_INFO("SERVER[%d] -- to publish." , p_server->model_instance_index);
        return mesh_model_publish(&msg_send, NULL);
    }
    else
    {
        APP_LOG_INFO("SERVER[%d] -- to response." , p_server->model_instance_index);
        return mesh_model_rsp_send(&msg_send);
    }
}
```

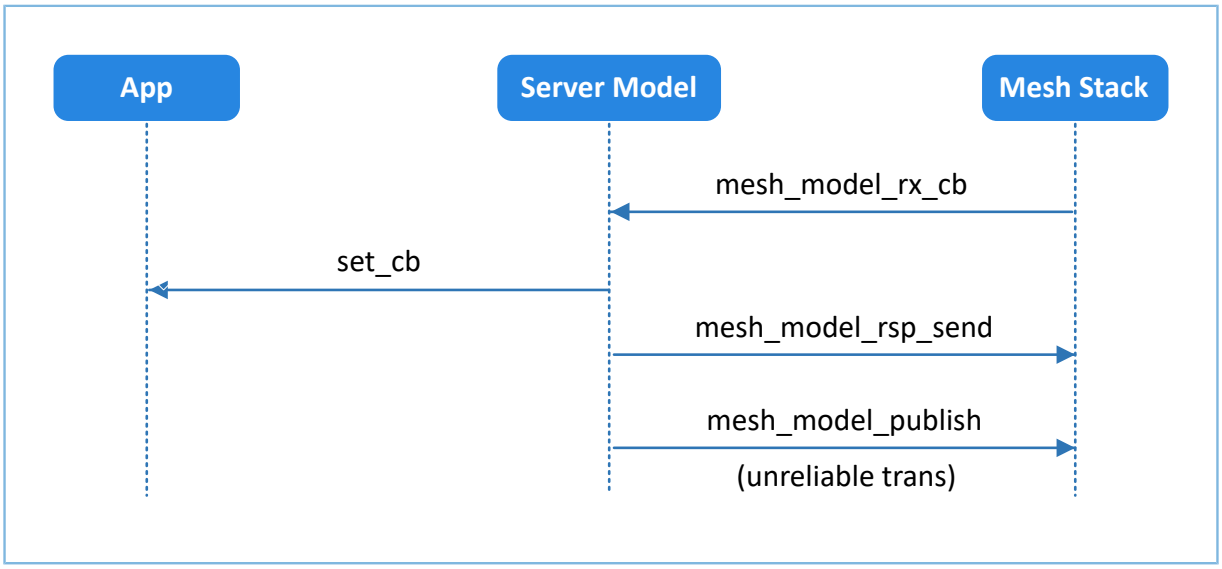

Figure 4-5 SIMPLE_ONOFF_OPCODE_SET message processing

The *_SET and *_SET_UNACKNOWLEDGE messages are processed in different ways. The former needs to call status_send() one more time than the latter, to call mesh_model_rsp_send() to respond to messages. Both the two types of messages finally call status_send() to call mesh_model_publish() to publish messages.

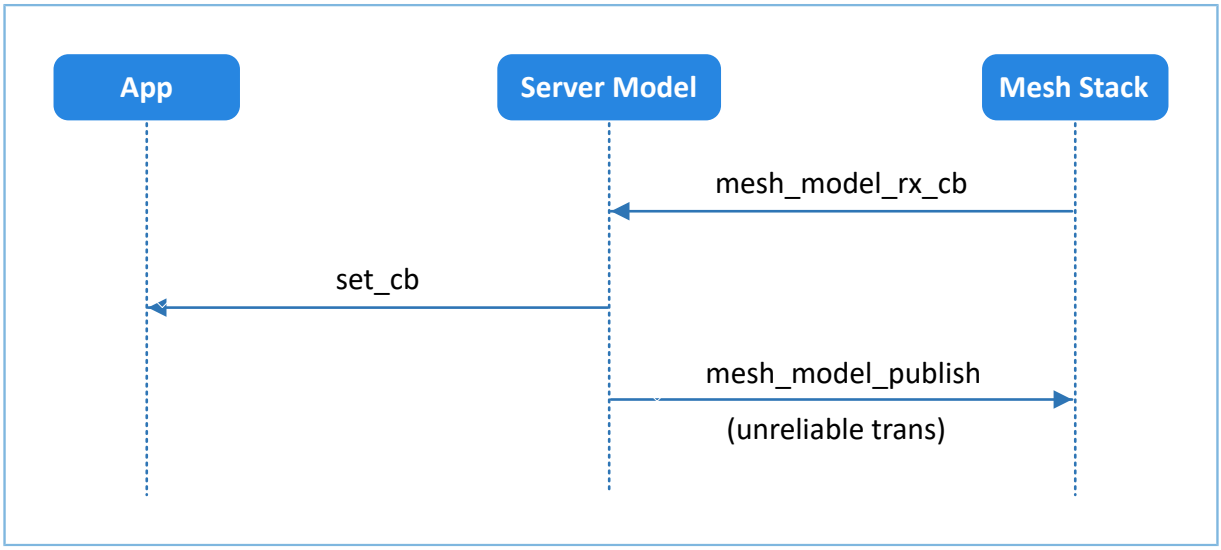

Figure 4-6 SIMPLE_ONOFF_OPCODE_SET_UNACKNOWLEDGED message processing

- handle_get_cb()

  This function can process the SIMPLE_ONOFF_OPCODE_GET message. status_send() calls mesh_model_rsp_send() to respond to messages.

```
static void handle_get_cb(const mesh_model_msg_ind_t *p_rx_msg,  void *p_args)
{
    simple_onoff_server_t * p_server = (simple_onoff_server_t *) p_args;

    APP_LOG_INFO("[%s] enter." , __func__);
    APP_LOG_INFO("SERVER[%d] -- Receive message, want to get on-off state!!!" , p_server-
>model_instance_index);

    simple_onoff_status_msg_pkt_t out_data = {0};

    if (get_params_validate(p_rx_msg))
    {
        p_server->client_address = p_rx_msg->src;
        p_server->settings.p_callbacks->onoff_cbs.get_cb(p_server->model_instance_index,
 &out_data.present_on_off);
        uint32_t send_status = status_send(p_server, p_rx_msg, &out_data);
        if (MESH_ERROR_NO_ERROR ! = send_status)
        {
            APP_LOG_ERROR("SERVER[%d] -- Publish status faild, error code: 0x%04x",
 p_server->model_instance_index, send_status);
        }
    }
}
```

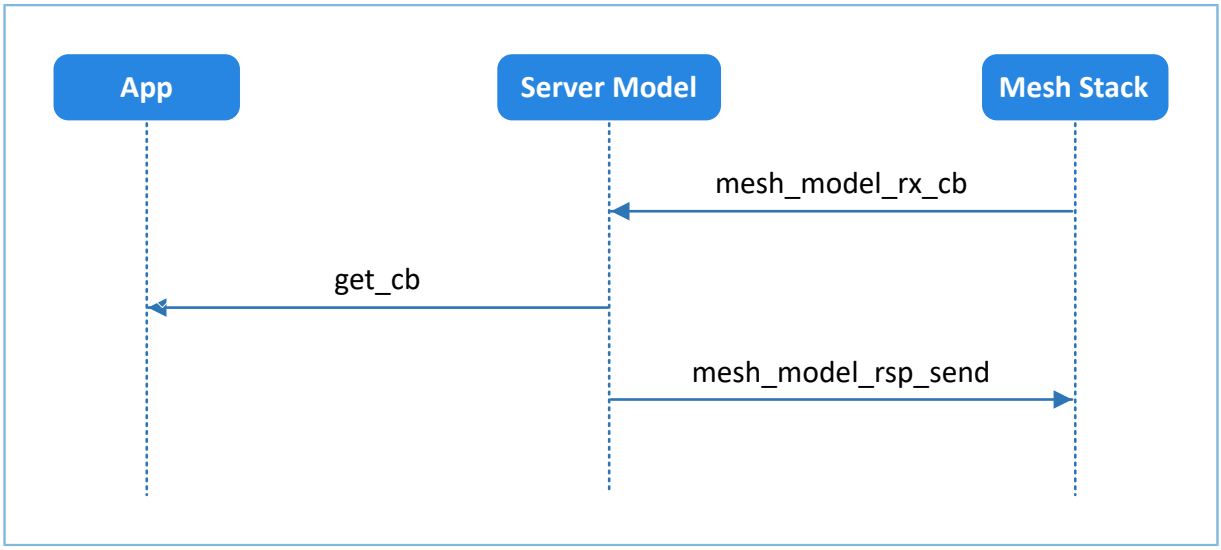

Figure 4-7 SIMPLE_ONOFF_OPCODE_GET message processing

4.2.1.2.3 Processing of To-be-sent Message

The SOO Server Model can send the SIMPLE_ONOFF_OPCODE_STATUS message actively. status_send() calls mesh_model_publish() to publish messages.

```
uint16_t simple_onoff_server_status_publish(simple_onoff_server_t * p_server,
                                    const simple_onoff_status_msg_pkt_t * p_params)
{
    if (NULL == p_server || NULL == p_params)
    {
        return MESH_ERROR_SDK_INVALID_PARAM;
    }

    return status_send(p_server, NULL, p_params);
}
```

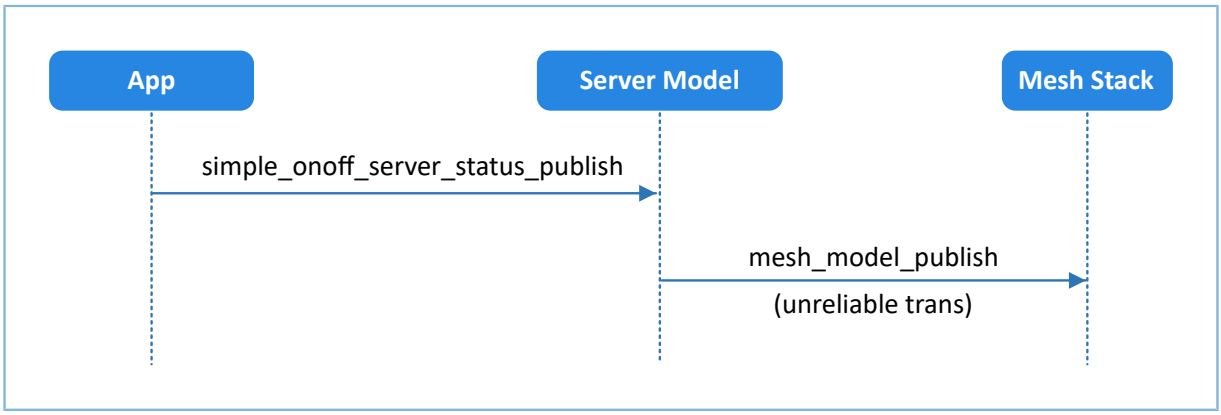

Figure 4-8 SIMPLE_ONOFF_OPCODE_STATUS message sending