AN1370: *Bluetooth*® Mesh Device Firmware Update Example Walkthrough

This version of AN1370 has been deprecated with the release of Simplicity SDK Suite 2025.6.1 For the latest version, see docs.silabs.com.

****

The Bluetooth mesh SDK comes with example projects that have the Bluetooth Mesh Device Firmware Update feature enabled to perform firmware updates in a Bluetooth mesh network. The examples assume use of Silicon Labs devices for the distributor node and the nodes whose firmware is to be updated, and the Silicon Labs Bluetooth mesh mobile app as the provisioner and initiator. This document describes the bootloader configurations and the firmware update models in the example projects, and walks through a firmware update demonstration.

KEY POINTS

- Short introduction to Bluetooth Mesh Device Firmware Update
- Bootloader configurations
- DFU Distributor example application
- Firmware update models
- Silicon Labs Bluetooth Mesh mobile application

1 Introduction

The Bluetooth Mesh Model specification v1.1 defines a standard way to update device firmware over a Bluetooth mesh network. The device firmware update process typically requires three types of nodes involved: **Initiator** and **Distributor** nodes, as well as the nodes to be updated, called **Target nodes**. These roles are fulfilled by enabling the Firmware Update and BLOB Transfer models:

- Firmware Distribution Client
- Firmware Distribution Server
- Firmware Update Client
- Firmware Update Server
- BLOB Transfer Client
- BLOB Transfer Server

To understand the basics of the Bluetooth Mesh Device Firmware Update specification, see *AN1319: Bluetooth® Mesh Device Firmware Update*.

This document explains the Bluetooth mesh examples, installed as part of the Bluetooth Mesh SDK, running as the Distributor and Target nodes, and the Silicon Labs Bluetooth Mesh mobile app running as the Initiator. Section 2 Bootloader Configurations for Firmware Updates describes the bootloader examples that are an essential program to boot up a Silicon Labs device and to update firmware on the device. Section 3 Firmware Update Examples and Models discusses the models required for running the Distributor and Target nodes, and how to create an update image archive. Section 4 Firmware Update Demonstration walks through a firmware update demo using the Silicon Labs Bluetooth Mesh mobile app, the Distributor example, and other Bluetooth mesh examples as Target nodes.

1.1 Requirements

The following is required to run the DFU examples.

- At least two mainboards with a supported board installed, one used for the Distributor and the other(s) for the device firmware update target(s).
- Simplicity Studio 5
- Gecko SDK Suite 4.2.2 (Bluetooth Mesh SDK 4.2.0) or later. The bootloader and DFU examples are included in the SDK.
- Silicon Labs Bluetooth Mesh Mobile Application
  - Used for discovering and provisioning devices.
  - Includes network, group, and publish-subscribe setup.
  - Allows device configuration for Device Firmware Update.

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

2 Bootloader Configurations for Firmware Updates

The Silicon Labs Gecko Bootloader is a common bootloader for all the newer wireless MCUs from Silicon Labs. The Gecko Bootloader can be configured to perform a variety of bootload functions, from device initialization to firmware upgrades. The Gecko Bootloader uses a proprietary format for its upgrade images, called GBL (Gecko Bootloader). These images are produced with the file extension ".gbl".

The bootloader performs a firmware image update by writing the firmware update image to a region of flash memory referred to as the download space. The download space is either an external memory device such as an EEPROM or a section of the device's internal flash. The bootloader partitions the download space into one or multiple storage slots and stores a single firmware update image in a storage slot.

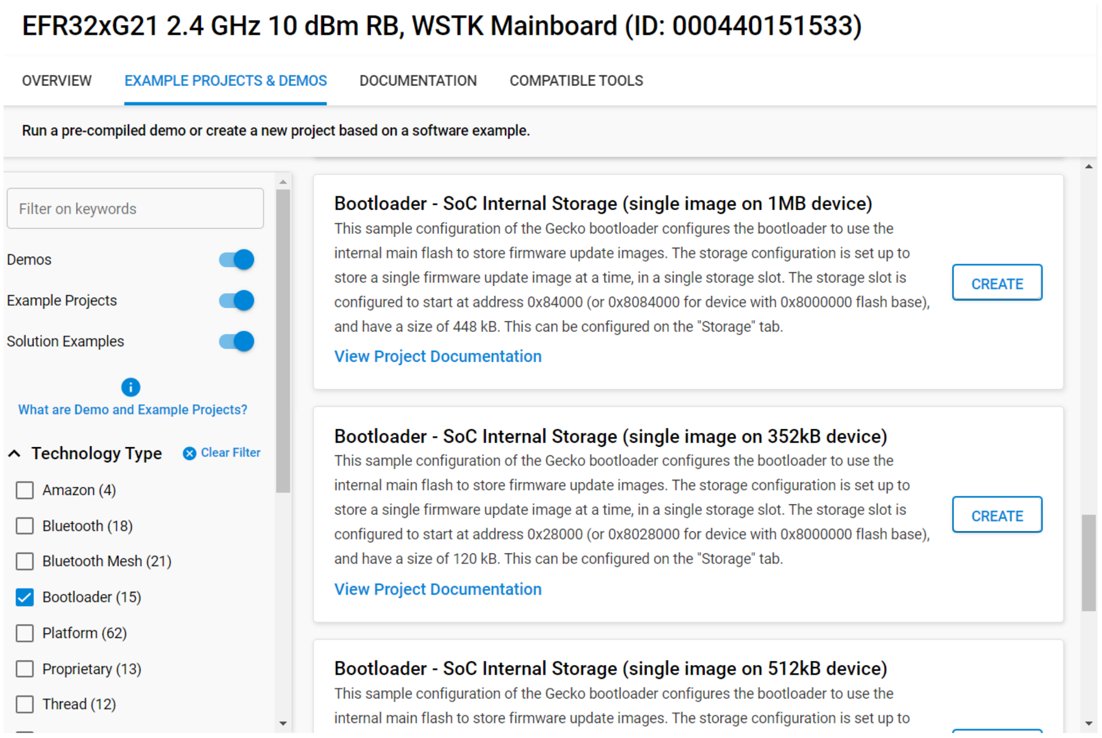
The bootloader example projects provided in the Gecko SDK Suite come with a preconfigured set of installed components and configurations for different specifications of Silicon Labs MCUs. This section discusses the bootloader example configurations you can use to build bootloaders for Bluetooth mesh applications and firmware updates. You may need to change the storage slot size by configuring the **Bootloader Storage Slot Setup** under Storage components in the Platform > Bootloader Software Components in Simplicity Studio. See Section 7 of UG489: Silicon Labs Gecko Bootloader User's Guide for GSDK 4.0 and Higher for more information about the Gecko Bootloader configurations.

2.1 Internal Flash Size

It is not recommended to use the devices that have internal flash smaller than 768 KB for Target nodes and smaller than 1 MB for the Distributor to store firmware update images. You can use one of the following example projects depending on the internal flash size.

- **Bootloader - SoC Internal Storage (single image on 768kB device)** for Target nodes
- **Bootloader - SoC Internal Storage (single image on 1MB device)**
- **Bootloader - SoC Internal Storage (single image on 1536kB device)**
- **Bootloader - SoC Internal Storage (single image on 1920kB device)**
- **Bootloader - SoC Internal Storage (single image on 2MB device)**

The Target node requires only a single storage slot to store and apply a firmware image. The Distributor can optionally store multiple firmware images but can distribute only one firmware image to Target nodes in a firmware update. To store multiple firmware images on the Distributor, Silicon Labs recommends using external flash for multiple storage slots. The **Bootloader - SoC Internal Storage (multiple images on 1MB device)** example project provides an example configuration of 2 storage slots on internal flash. Use this example for the Distributor if the target device has at least 1920 KB of internal flash.



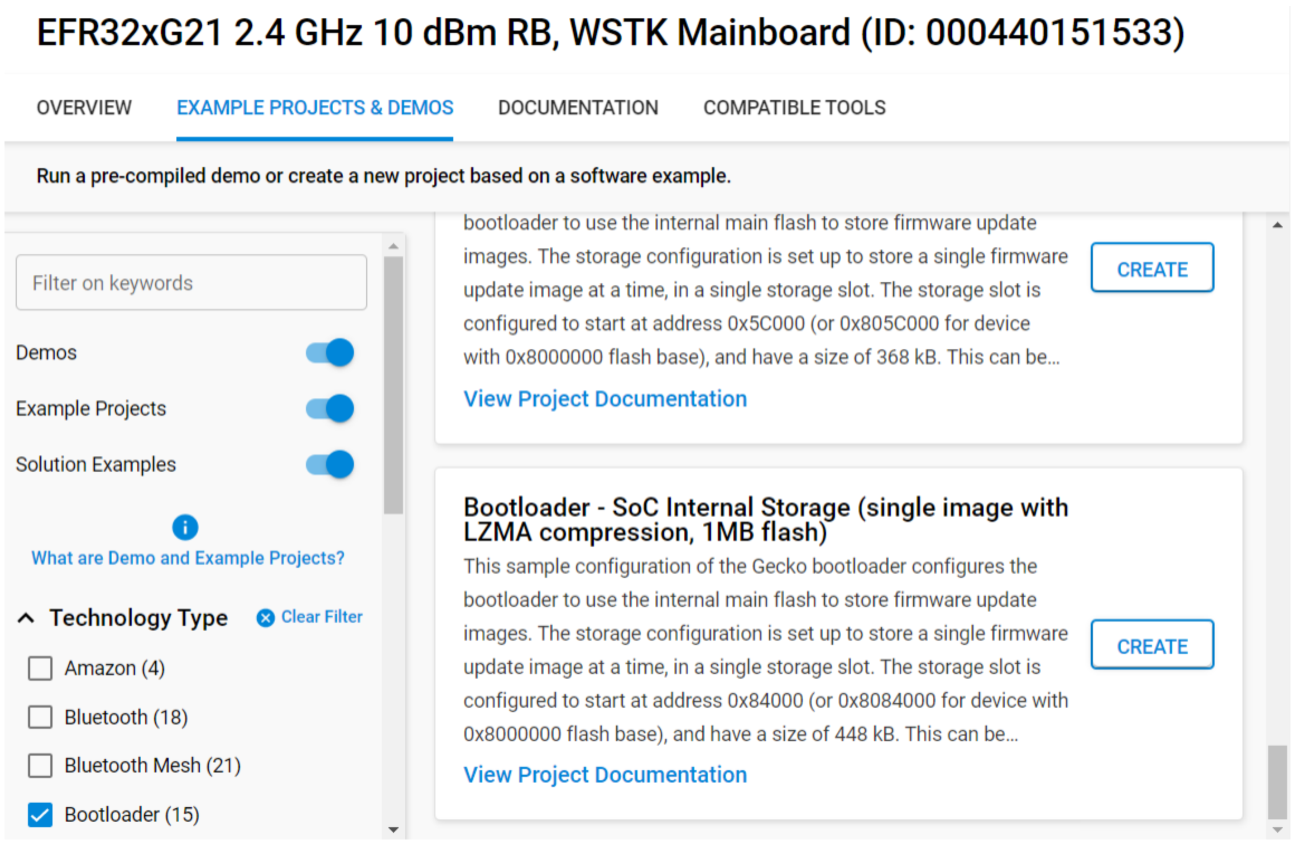
Bootloader - SoC Internal Storage (single image on 1MB device)
This sample configuration of the Gecko bootloader configures the bootloader to use the internal main flash to store firmware update images. The storage configuration is set up to store a single firmware update image at a time, in a single storage slot. The storage slot is configured to start at address 0x84000 (or 0x8084000 for device with 0x8000000 flash base), and have a size of 448 kB. This can be configured on the "Storage" tab.

Bootloader - SoC Internal Storage (single image on 352kB device)
This sample configuration of the Gecko bootloader configures the bootloader to use the internal main flash to store firmware update images. The storage configuration is set up to store a single firmware update image at a time, in a single storage slot. The storage slot is configured to start at address 0x28000 (or 0x8028000 for device with 0x8000000 flash base), and have a size of 120 kB. This can be configured on the "Storage" tab.

Bootloader - SoC Internal Storage (single image on 512kB device)
This sample configuration of the Gecko bootloader configures the bootloader to use the internal main flash to store firmware update images. The storage configuration is set up to

2.2 Compressed Update Image

Use the **Bootloader - SoC Internal Storage (single image with LZMA compression, 1MB flash)** example project if the firmware image is too big to fit the storage slot but the compressed firmware image can fit it. Section 3.4 Creating Update Image Archive describes how to create a compressed firmware image.


EFR32xG21 2.4 GHz 10 dBm RB, WSTK Mainboard (ID: 000440151533)

OVERVIEW | EXAMPLE PROJECTS & DEMOS | DOCUMENTATION | COMPATIBLE TOOLS

Run a pre-compiled demo or create a new project based on a software example.

Filter on keywords

Demos
Example Projects
Solution Examples

What are Demo and Example Projects?

Technology Type — Clear Filter
- Amazon (4)
- Bluetooth (18)
- Bluetooth Mesh (21)
- [✓ Bootloader (15)]

bootloader to use the internal main flash to store firmware update images. The storage configuration is set up to store a single firmware update image at a time, in a single storage slot. The storage slot is configured to start at address 0x5C000 (or 0x805C000 for device with 0x8000000 flash base), and have a size of 368 kB. This can be...

View Project Documentation — CREATE

Bootloader - SoC Internal Storage (single image with LZMA compression, 1MB flash)

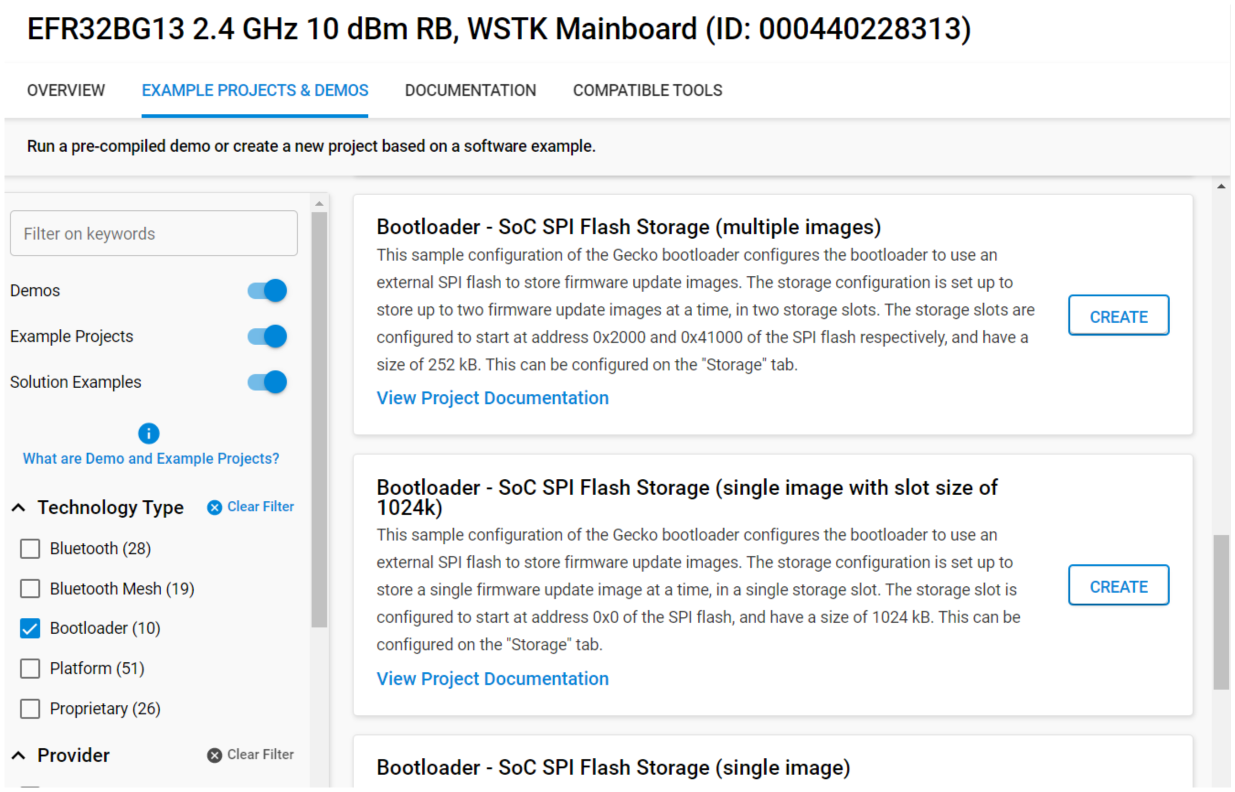
This sample configuration of the Gecko bootloader configures the bootloader to use the internal main flash to store firmware update images. The storage configuration is set up to store a single firmware update image at a time, in a single storage slot. The storage slot is configured to start at address 0x84000 (or 0x8084000 for device with 0x8000000 flash base), and have a size of 448 kB. This can be...

View Project Documentation — CREATE

2.3 External Flash

Use the **Bootloader - SoC SPI Flash Storage (single image)** example project if the device's external flash is equal to or larger than 512 KB and smaller than 1 MB. Use the **Bootloader - SoC SPI Flash Storage (single image with slot size of 1024k)** example project if the device's external flash is equal to or larger than 1 MB. The external flash size should be at least 512 KB for Target nodes and at least 768 KB for the Distributor.

The Target node requires only a single storage slot to store and apply a firmware image. The Distributor can optionally store multiple firmware images but can distribute only one firmware image to Target nodes in a firmware update. Use the **Bootloader - SoC SPI Flash Storage (multiple images)** example project if you need of store multiple firmware images on the Distributor. With the bootloader configuration of 2 storage slots, at least 1 MB of external flash is recommended.


EFR32BG13 2.4 GHz 10 dBm RB, WSTK Mainboard (ID: 000440228313)

OVERVIEW | EXAMPLE PROJECTS & DEMOS | DOCUMENTATION | COMPATIBLE TOOLS

Run a pre-compiled demo or create a new project based on a software example.

Filter on keywords

Demos
Example Projects
Solution Examples

What are Demo and Example Projects?

Technology Type — Clear Filter
- [ ] Bluetooth (28)
- [ ] Bluetooth Mesh (19)
- [x] Bootloader (10)
- [ ] Platform (51)
- [ ] Proprietary (26)

Provider — Clear Filter

Bootloader - SoC SPI Flash Storage (multiple images)
This sample configuration of the Gecko bootloader configures the bootloader to use an external SPI flash to store firmware update images. The storage configuration is set up to store up to two firmware update images at a time, in two storage slots. The storage slots are configured to start at address 0x2000 and 0x41000 of the SPI flash respectively, and have a size of 252 kB. This can be configured on the "Storage" tab.
View Project Documentation
CREATE

Bootloader - SoC SPI Flash Storage (single image with slot size of 1024k)
This sample configuration of the Gecko bootloader configures the bootloader to use an external SPI flash to store firmware update images. The storage configuration is set up to store a single firmware update image at a time, in a single storage slot. The storage slot is configured to start at address 0x0 of the SPI flash, and have a size of 1024 kB. This can be configured on the "Storage" tab.
View Project Documentation
CREATE

Bootloader - SoC SPI Flash Storage (single image)

3 Firmware Update Examples and Models

To perform firmware updates, you need an Initiator, a Distributor, and at least one Target node. This section describes the setup to run the **Distributor** and **Target nodes** examples provided in the Bluetooth mesh SDK, and the procedures to create an update image archive. See *QSG183: Bluetooth Mesh SDK Quick-Start Guide for Bluetooth Mesh 1.1* for an introduction to configuring and building your own projects, and for a guide to additional resources.

The Initiator and Stand-alone Updater are not discussed in this document. However, these functionalities are demonstrated using the Silicon Labs Bluetooth Mesh mobile app, described in Section 4 Firmware Update Demonstration and Section 5 Firmware Update with Stand-alone Updater, respectively.

Firmware storage is an important part of the device firmware update process. The flash memory is managed by and accessed via the bootloader. See Section 2 Bootloader Configurations for Firmware Updates for the flash configurations. The table in Section 6 Appendix – Silicon Labs Product Positioning for Bluetooth Mesh DFU is a recommendation of the Silicon Labs parts for running the DFU roles.

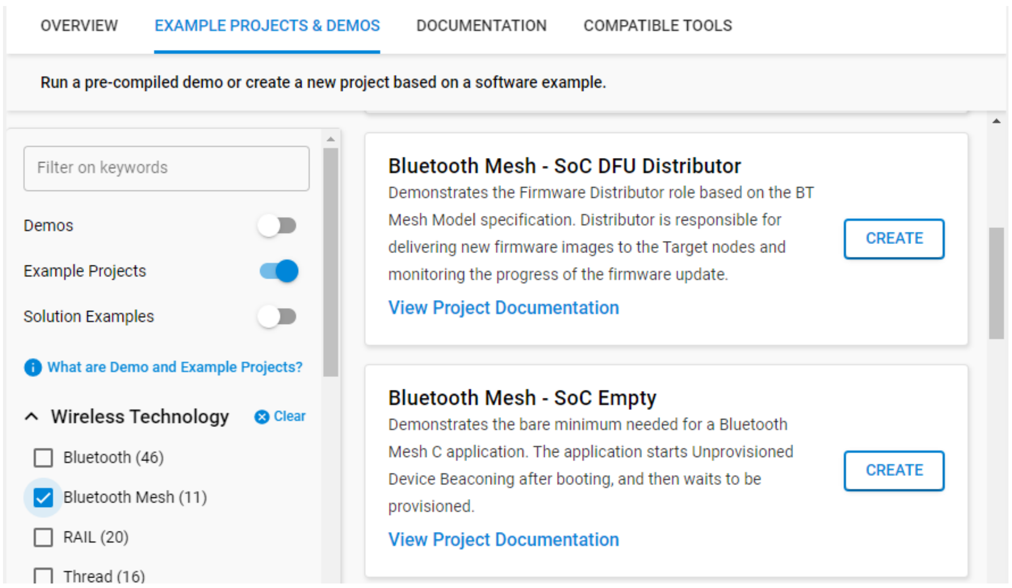
3.1 Distributor Example Application

The Distributor example application is provided as a pre-built demo binary image, ready to download and use, and a corresponding example project that you can modify and then build for the target part. The precompiled demo is only available for selected EFR32xG21 and EFR32xG24 parts.

This section describes how to build the example project and run the example application on a Silicon Labs device. The example is only available for a limited set of parts, including selected EFR32xG12, xG21 and xG24 parts.

Open Simplicity Studio 5 with a compatible SoC wireless kit connected to the computer. Select the part in Debug Adapters view to open the Launcher perspective.

1. Click the **Example Projects & Demos** tab.
2. To see only the example projects, turn off **Demos**.
3. Under Technology Type, filter on **Bluetooth Mesh**.
4. Next to the **Bluetooth Mesh – SoC DFU Distributor**, click **CREATE**, modify project settings, click **FINISH**.
5. Build and flash the project to the device.



Run a pre-compiled demo or create a new project based on a software example.

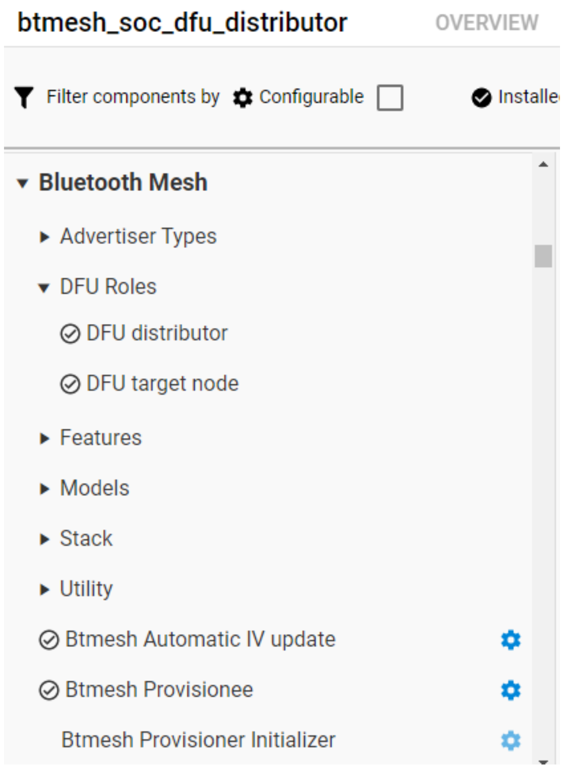
Bluetooth Mesh - SoC DFU Distributor
Demonstrates the Firmware Distributor role based on the BT Mesh Model specification. Distributor is responsible for delivering new firmware images to the Target nodes and monitoring the progress of the firmware update.

Bluetooth Mesh - SoC Empty
Demonstrates the bare minimum needed for a Bluetooth Mesh C application. The application starts Unprovisioned Device Beaconing after booting, and then waits to be provisioned.

The example has the components that are required for the Distributor functionality installed by default. To enable the Distributor functionality in other Bluetooth mesh projects, install the **DFU distributor** component that automatically brings the necessary model components:

1. Open the project .slcp file in Project Explorer view of the Simplicity IDE perspective.
2. Click the **Software Components** tab.

3. Select the **DFU distributor** component under **Bluetooth Mesh > DFU Roles** and click **Install**.


btmesh_soc_dfu_distributor OVERVIEW

Filter components by Configurable ☐ Installe

- Bluetooth Mesh
  - Advertiser Types
  - DFU Roles
    - DFU distributor
    - DFU target node
  - Features
  - Models
  - Stack
  - Utility
  - Btmesh Automatic IV update
  - Btmesh Provisionee
  - Btmesh Provisioner Initializer

This component automatically installs the following model components:

- **Bluetooth Mesh > Models > Firmware Update > Firmware Distribution Server**
- **Bluetooth Mesh > Models > Firmware Update > Firmware Update Client**
- **Bluetooth Mesh > Models > Transport > BLOB Transfer Client**
- **Bluetooth Mesh > Models > Transport > BLOB Transfer Server**

See Section 3.3 DFU Model Configurations for the configurations of these Firmware Update and BLOB Transfer models.

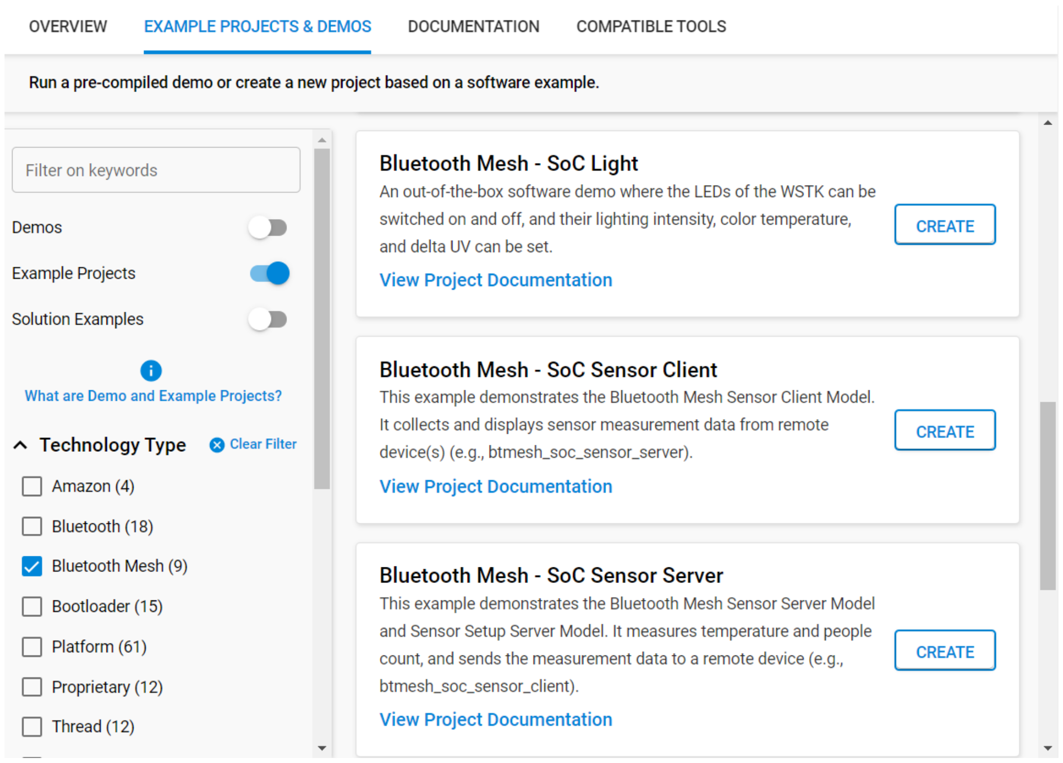
3.2 Target Node Applications

A Target node is a node whose firmware is to be updated. The Bluetooth mesh examples provided in the Bluetooth mesh SDK except **Bluetooth Mesh – NCP Empty** and **Bluetooth Mesh – SoC Empty** have the firmware update functionality enabled by default, i.e., are Target nodes.

These example applications are provided both as prebuilt demo binary images, ready to download and use, and corresponding example projects that you can modify and then build for the target part. The precompiled demos are only available for a limited set of parts, including selected EFR32xG13, xG21 and xG24 parts and BGM13 and MGM21 modules. The examples can be built for any part supported by the Bluetooth Mesh SDK.

Note: EFR32xG22 parts have limited support for Bluetooth Mesh.

To build an example project for the device firmware update target, refer to the procedure described in Section 3.1 Distributor Example Application.


OVERVIEW | EXAMPLE PROJECTS & DEMOS | DOCUMENTATION | COMPATIBLE TOOLS

Run a pre-compiled demo or create a new project based on a software example.

Filter on keywords

Demos
Example Projects
Solution Examples

What are Demo and Example Projects?

Technology Type — Clear Filter
- [ ] Amazon (4)
- [ ] Bluetooth (18)
- [x] Bluetooth Mesh (9)
- [ ] Bootloader (15)
- [ ] Platform (61)
- [ ] Proprietary (12)
- [ ] Thread (12)

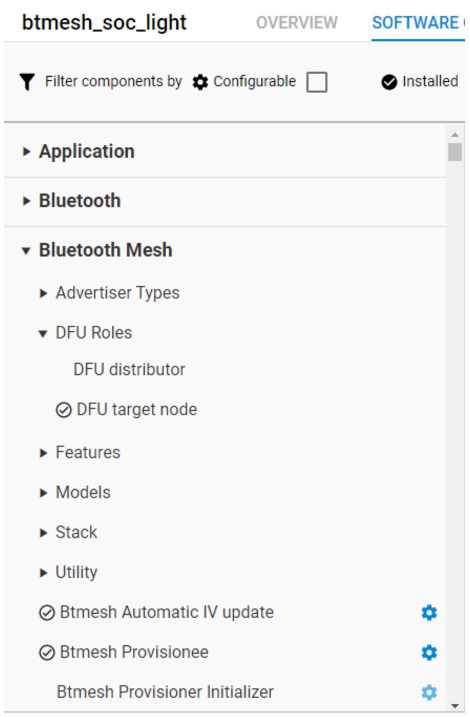
Bluetooth Mesh - SoC Light
An out-of-the-box software demo where the LEDs of the WSTK can be switched on and off, and their lighting intensity, color temperature, and delta UV can be set.
View Project Documentation — CREATE

Bluetooth Mesh - SoC Sensor Client
This example demonstrates the Bluetooth Mesh Sensor Client Model. It collects and displays sensor measurement data from remote device(s) (e.g., btmesh_soc_sensor_server).
View Project Documentation — CREATE

Bluetooth Mesh - SoC Sensor Server
This example demonstrates the Bluetooth Mesh Sensor Server Model and Sensor Setup Server Model. It measures temperature and people count, and sends the measurement data to a remote device (e.g., btmesh_soc_sensor_client).
View Project Documentation — CREATE

The examples have the components that are required for the Target node functionality installed by default. To enable the Target node functionality in other Bluetooth mesh projects, install the **DFU target node** component that automatically brings the necessary model components:

1. Open the project .slcp file in Project Explorer view of the Simplicity IDE perspective.
2. Click the **Software Components** tab.
3. Select the **DFU target node** component under **Bluetooth Mesh > DFU Roles** in the left panel and click **Install**.


btmesh_soc_light — OVERVIEW | SOFTWARE

Filter components by: Configurable ☐ | Installed

- Application
- Bluetooth
- Bluetooth Mesh
  - Advertiser Types
  - DFU Roles
    - DFU distributor
    - ✓ DFU target node
  - Features
  - Models
  - Stack
  - Utility
  - ✓ Btmesh Automatic IV update
  - ✓ Btmesh Provisionee
  - Btmesh Provisioner Initializer

This component automatically installs the following model components:

- **Bluetooth Mesh > Models > Firmware Update > Firmware Update Server**
- **Bluetooth Mesh > Models > Transport > BLOB Transfer Server**

See Section 3.3 DFU Model Configurations for the configurations of these Firmware Update and BLOB Transfer models.

3.3 DFU Model Configurations

This section describes the main settings of the Firmware Update and BLOB Transfer models. Open the project **.slcp** file in Project Explorer view of the Simplicity IDE perspective.

1. Click the **Software Components** tab.
2. Select **Bluetooth Mesh > Models > Firmware Update** or **Bluetooth Mesh > Models > Transport** in the left panel.
3. Click the gear icon next to the model to be configured.

3.3.1 Firmware Distribution Server

Configuration of **Bluetooth Mesh > Models > Firmware Update > Firmware Distribution Server**:

Configuration Option	Description	Default Value
Max node list size	Maximum number of firmware update server nodes which can participate in the distribution. Range: 1 to 65535.	8
Default Multicast Threshold	If the number of servers for any step exceeds or is equal to this number then the group address will be used, otherwise servers will be looped through one by one. Range: 1 to 65535. The value of 0 disables the feature.	1
Retry time of message transmissions	Retry time of firmware update message transmissions. Range: 0 to 65535.	3000
NVM key of the firmware list	NVM key of the firmware list. Range: 0 to 65535.	16393
Enable Logging	Enable / disable logging of Firmware Distribution Server model specific messages.	On
Enable Logging: Enable BT Mesh Stack Platform Callback Logging	Enable / disable logging of BT Mesh Stack Firmware Distribution Server model platform callbacks. The FW Distribution Server model in BT Mesh stack calls platform callback functions to query the remaining space, firmware count and firmware information.	Off
Enable Logging: Text prepended to every log message	Every log message in the component is started with this text.	FwDistributor

3.3.2 Firmware Update Server

Configuration of **Bluetooth Mesh > Models > Firmware Update > Firmware Update Server**:

Configuration Option	Description	Default Value
General		
Number of firmware on device	Number of firmware on device. Range: 1 to 255.	1
Maximum length of metadata	Maximum length of metadata. Range: 0 to 255.	255 / LPN:31
Firmware Information		
Firmware identifier	Firmware identifier	fwid or <specific_node_id>
Update URI	Update URI	https://example.com/upd_uri
Company Identifier: CID MSB	Most Significant Byte of the Company ID. Hexadecimal string literal.	\x02
Company Identifier: CID LSB	Least Significant Byte of the Company ID. Hexadecimal string literal.	\xFF

Configuration Option	Description	Default Value
Logging		
Logging	Enable / disable logging of Firmware Update Server model specific messages.	On
Logging: Period of verification UI update [ms]	Setting it to 0 the user interface (log & display) is updated every time when progress is made. Range: 0 to 2000. Step: 100.	200

The weak functions for the Firmware Update Server model are mostly implementation-dependent and can be overwritten in the application:

Function	Description
sl_btmesh_fw_update_server_metadata_check_start()	User callback indicating start of metadata check
sl_btmesh_firmware_update_server_metadata_check_step()	User callback executing one step of metadata check
sl_btmesh_fw_update_server_verify_start()	User callback for determining the maximum chunk size of verification
sl_btmesh_fw_update_server_verify_step()	User callback to execute one step of the verification
sl_btmesh_fw_update_server_apply()	User callback indicating firmware apply request

See `<sdk>/app/bluetooth/common/btmesh_firmware_update_server/sl_btmesh_firmware_update_server_api.h` for further information.

Note: A Target node will become unprovisioned by default after a firmware update. You can overwrite the following functions in `sl_btmesh_firmware_update_server_api.c` and modify the code described below to change the behavior:

- `sl_btmesh_fw_update_server_apply()` – remove `sl_btmesh_node_reset()` and `sl_bt_nvm_erase_all()`.

```
 if (BOOTLOADER_OK == bootloader_setImageToBootload(idx)) {
   // Reset node
   sl_btmesh_node_reset();
   // Erase NVM data
   sl_bt_nvm_erase_all();
   // Delay install
   sl_simple_timer_start(&timer, 1000, apply_step, NULL, true);
   apply_cntdwn = APPLY_DELAY;
 }
```

- `sl_btmesh_fw_update_server_metadata_check_start()` – set another value to `*additional_information`. The additional information is provided to the Bluetooth mesh stack and will appear in the Firmware Update Status and Firmware Update Firmware Metadata Status messages.

```
 *additional_information = BTMESH_FW_UPDATE_SERVER_ADDITIONAL_INFORMATION_UNPROVISION;
```

- `sl_btmesh_firmware_update_server_metadata_check_step()` – set another value to `*additional_information`. The additional information is provided to the Bluetooth mesh stack and will appear in the Firmware Update Status and Firmware Update Firmware Metadata Status messages.

```
 *additional_information = BTMESH_FW_UPDATE_SERVER_ADDITIONAL_INFORMATION_UNPROVISION;
```

3.3.3 BLOB Transfer Client

Configuration of **Bluetooth Mesh > Models > Transport > BLOB Transfer Client**:

Configuration Option	Description	Default Value
Enable Logging	Enable / disable logging of BLOB Transfer Client model specific messages.	On
Enable Logging: Text prepended to every log message	Every log message in the component is started with this text.	BlobTfClient
Enable Logging: Log BLOB Status messages	Log the content of BT Mesh BLOB status messages.	1
BLOB Transfer Limits: Max number of servers	Maximum number of BLOB transfer servers that can be serviced in a transfer (affects BT Mesh stack memory usage). Range: 1 to 1008.	8
BLOB Transfer Limits: Max number of blocks	Maximum number of blocks supported in a BLOB Transfer (affects BT Mesh stack memory usage). Range: 1 to 1888.	1850
BLOB Transfer Limits: Max number of chunks per block	Maximum number of chunks per block supported in a BLOB Transfer (affects BT Mesh stack memory usage). Range: 1 to 2000.	128
BLOB Transfer Limits: Max chunk size	Maximum chunk size that can be selected during BLOB Transfer. Range: 1 to 241.	241
Retry and Separation parameters: Default separation time between chunks	Default minimum separation time between two chunks in the same block. Range: 0 to 65535.	0
Retry and Separation parameters: Default max retry of message transmissions	Default max retries of message transmissions (query info, transfer start, block start, block query). Range: 0 to 1000.	50
Retry and Separation parameters: Default retry time of message transmissions	Default retry time of message transmissions (query info, transfer start, block start, block query). Range: 0 to 65535.	2000

The weak functions for the BLOB Transfer Client model are mostly implementation-dependent and can be overwritten in the application:

Function	Description
sl_btmesh_blob_transfer_client_calculate_block_size_log()	Calculates the binary logarithm of the block size for the current BLOB transfer from the provided parameters, which are the result of the Retrieve Capabilities procedure of the BLOB Transfer. The parameters passed represent the aggregated capabilities of the BLOB transfer client and every BLOB transfer server which participates in the current transfer. The default implementation calculates the greatest possible block size from the parameters.
sl_btmesh_blob_transfer_client_calculate_chunk_size()	Calculates the chunk size for the next block in the current BLOB transfer from the previously selected binary logarithm of the block size and from the result of the Retrieve Capabilities procedure of the BLOB Transfer. The default implementation calculates the greatest possible chunk size.

See `<sdk>/app/bluetooth/common/btmesh_blob_transfer_client/sl_btmesh_blob_transfer_client.h` for further information.

3.3.4 BLOB Transfer Server

Configuration of **Bluetooth Mesh > Models > Transport > BLOB Transfer Server**:

Configuration Option	Description	Default Value
General		

Configuration Option	Description	Default Value
Min Block Size Log	Block states need to be monitored. The smaller the blocks, the bigger the state storage. Range: 6 to 32. Note that decreasing the minimum block size will result in increased heap usage. Change this value with care.	9
Max Block Size Log	Blocks are cached on heap before being written into NVM. Range: 6 to 32. Note that increasing the maximum block size will result in increased heap usage. Change this value with care.	9
Maximum of number of chunks per block	Maximum of number of chunks per block. Range: 8 to 64. Step: 8.	16 / LPN:64
Maximum chunk size	If the max chunk size is 8, the chunk data fits into a single BT Mesh advertisement message. If the chunk data is segmented, N segments can transfer (N*12)-7 bytes of data. Range: 8 to 241. The advantage of a higher chunk size is higher throughput in a low noise environment. The advantage of a lower chunk size is fewer messages are retransmitted in a high noise environment due to lost chunk messages. LPN only: the number of chunk messages (segments) multiplied by requested chunk count in Partial Block Report should fit into the friend queue.	241 / LPN:8
Logging	Enable / disable logging of BLOB Transfer Server model-specific messages.	On
Transfer Start user callback	Enable / disable callback function when BLOB transfer starts.	On
Transfer Progress user callback	Enable / disable callback function when block transfer is finished.	On
Transfer Done user callback	Enable / disable callback function when BLOB transfer is finished.	On
	Supported Transfer Modes	
Push Mode	Push BLOB Transfer Mode.	On / LPN: Off
Pull Mode	Pull BLOB Transfer Mode.	On
Pull Mode: Number of chunks requested in Block Status or Partial Block Report	Number of chunks requested in Block Status or Partial Block Report. Range: 1 to 32.	4
Pull Mode: Interval, in milliseconds, between Partial Block Reports, if nothing is received	Interval, in milliseconds, between Partial Block Reports, if nothing is received. Range: 1000 to 30000. Step: 100.	1000
Pull Mode: Number of retries sending the same Partial Block Report, before giving up	Number of retries sending the same Partial Block Report, before giving up. Range: 1 to 10.	8
Pull Mode: LPN Mode	Used on LPN nodes.	Off / LPN:On
Pull Mode – LPN Mode: LPN high throughput mode	In high throughput mode the LPN node polls the friend node more frequently to increase the throughput at the expense of power consumption.	On
Pull Mode – LPN Mode – LPN high throughput mode: LPN poll delay in milliseconds	The delay of first LPN poll when the BLOB Transfer Server expects messages from the client after an event. The major part of BLOB transfer to LPN is waiting for the poll timeout to elapse in order to poll the friend node for BLOB Transfer messages. The maximum number of messages that can be transferred per polling is equal to the friend queue size during BLOB transfer to LPN. This poll delay parameter makes the polling more frequent when BLOB Transfer messages are expected to increase the throughput. Range: 100 to 30000. Step: 100. The LPN poll delay should be less than SL_BTMESH_LPN_POLL_TIMEOUT_CFG_VAL in sl_btmesh_lpn_config.h file.	500

Configuration Option	Description	Default Value
Pull Mode – LPN Mode – LPN high throughput mode: LPN poll logging	Enable / disable logging of LPN polls.	Off

3.4 Creating the Update Image Archive

An update image archive is a **gzip** archive file that consists of:

- **Manifest** – a file named **manifest.json**, which is in the format of JavaScript Object Notation (JSON). It contains a description of the firmware package, including the name of the firmware image file and an optional metadata file for the update.
- **Firmware Image** – the firmware image file in GBL format, as identified in the manifest file.
- **Metadata** – an optional file provided by the vendor and identified in the manifest file, that may provide metadata for the firmware image.

This section describes the procedure to generate a firmware image, manifest file, and update image archive.

Step 1 – generating a firmware image

Building a C-based Bluetooth mesh application in Simplicity Studio does not automatically generate the OTA DFU update images (GBL files). The GBL files need to be created separately by running a script located in the project's root folder. Two scripts are provided in the SDK examples:

- **create_bl_files.bat (for Windows)**
- **create_bl_files.sh (for Linux / Mac)**

Define two environmental variables, PATH_SCMD and PATH_GCCARM shown in the following table, before running the script.

Variable Name	Variable Value
PATH_SCMD	C:\SiliconLabs\SimplicityStudio\v5\developer\adapter_packs\commander
PATH_GCCARM	C:\SiliconLabs\SimplicityStudio\v5\developer\toolchains\gnu_arm\7.2_2017q4

Running the **create_bl_files** script creates six GBL files in a subfolder named `output_gbl`. The file named **application.gbl** is the firmware update image.

If you use a bootloader that supports LZMA compression of the firmware update image, as described in Section 2.2 Compressed Update Image, you must compress the GBL file with the `–compress` option of the `commander` utility. The **create_bl_files** script does not support compressed GBL update image generation. Therefore, run the following commands instead of the **create_bl_files** script and then the file named **application-lzma.gbl** is the firmware update image. Change `soc_btmesh_empty.axf` to the name of your project firmware image in the command line below.

```
arm-none-eabi-objcopy -O srec -R .text_apploader* -R .text_signature* soc_btmesh_empty.axf application.srec
commander gbl create application-lzma.gbl --app application.srec --compress lzma
```

For more information about the GBL file format, see Section 2 of UG489: Silicon Labs Gecko Bootloader User’s Guide for GSDK 4.0 and Higher. For more information on using the Simplicity Commander commands, see UG162: Simplicity Commander Reference Guide.

Step 2 – creating a manifest file

Create a file named **manifest.json** that is a text file in JSON format containing the name of the firmware update image, the firmware ID and optionally the name of the metadata file. The format of the `manifest/firmware/firmware_id` member is **Base16**. The `manifest/firmware/metadata_file` member contains the metadata file name if present. The following is an example of the content of the manifest.json file.

```
{
    "manifest": {
        "firmware": {
            "firmware_file": "application.gbl",
            "firmware_id": "4181C01592"
        }
    }
}
```

Step 3 – making an update image archive

Run the following command to generate a zipped tar archive:

```
tar czf firmware.gz application.gbl manifest.json
```

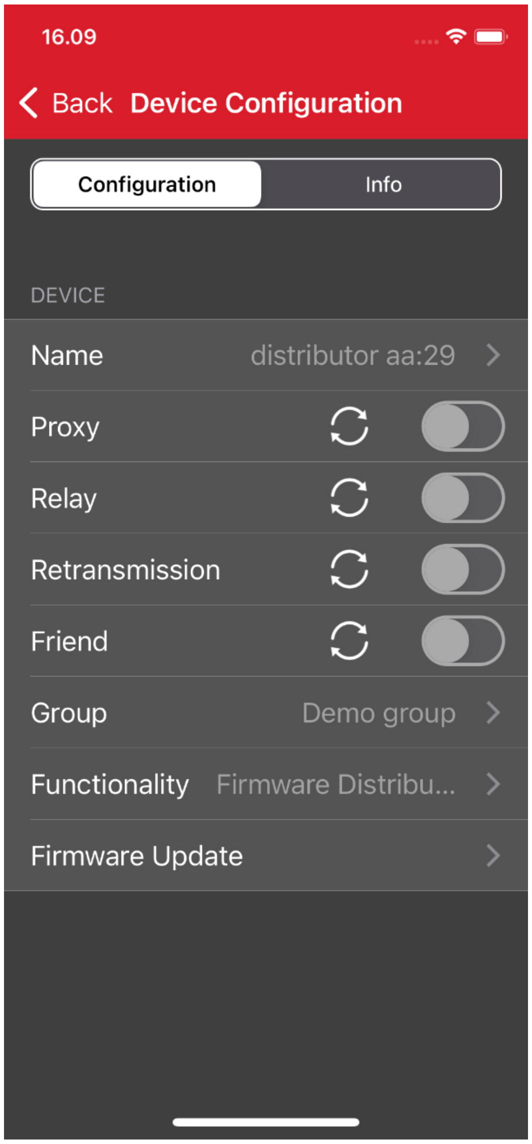
4 Firmware Update Demonstration

This section assumes you have flashed the **Bluetooth Mesh – SoC DFU Distributor** example application to one of the devices and the **Bluetooth Mesh – SoC Light** example application to the other(s) and have created an update image archive for the light example. Section 3 Firmware Update Examples and Models describes the setup of the examples and the update image archive.

The Bluetooth Mesh SDK comes with a DFU Python script that supports the provisioner and initiator functionalities. To use the script to update device firmware, see *AN1422: Provisioning and Firmware Update Using the DFU Python Script*.

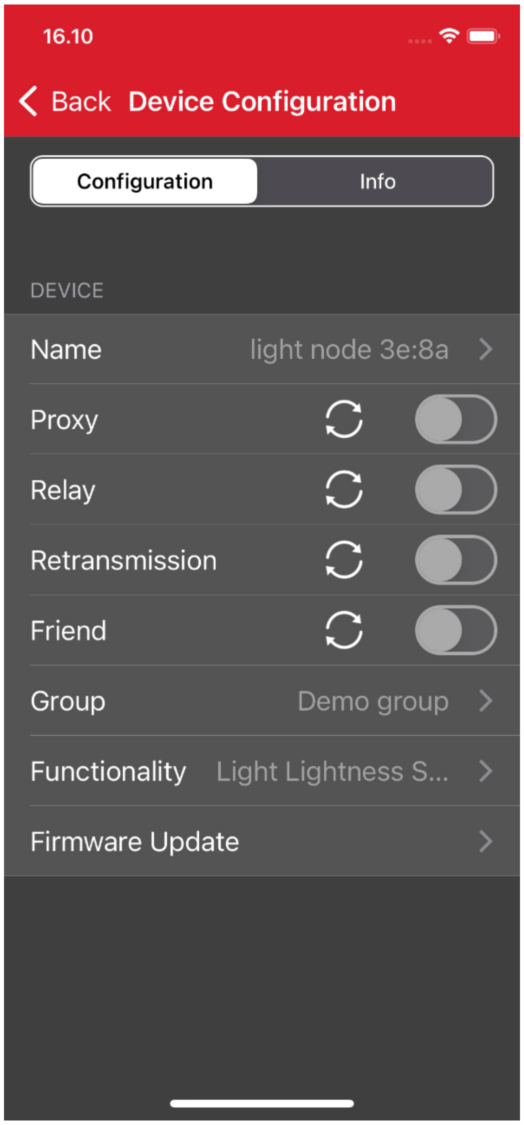
The examples display firmware update status on the device's LCD and output detailed information of the firmware update process to VCOM UART. To see the logs, open a serial terminal on the serial port assigned for the device with the following serial settings: baud rate 115200, data bits 8, stop bits 1 and parity None.

1. Provision and configure the Distributor node.
    1. Go to the Provision view and tap **Scan** at the top right.
    2. Tap **PROVISION** next to the Distributor device and tap **Continue** at the top right.
    3. In the Device Configuration view, tap **Group** and then select Demo group.
    4. In the Device Configuration view, tap **Functionality** and then select **Firmware Distribution Server.**



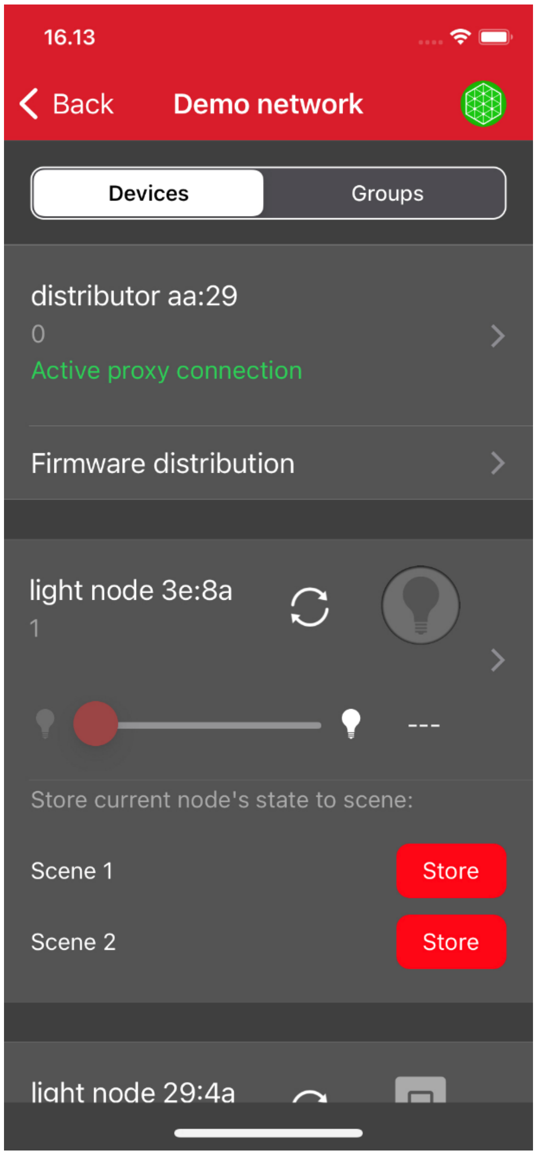
Name: distributor aa:29
Group: Demo group
Functionality: Firmware Distribu...

2. Provision and configure at least one Target node.
    1. Go to the Provision view and tap **Scan** at the top right.
    2. Tap **PROVISION** next to a Light device and tap **Continue** at the top right.
    3. In the Device Configuration view, tap **Group** and then select Demo group.
    4. In the Device Configuration view, tap **Functionality** and then select a functionality for the node. The **Firmware Update Server** functionality required for the firmware update is invisible but is automatically bound to the selected functionality.
    5. Repeat above steps for all Target nodes.



Name light node 3e:8a
Proxy
Relay
Retransmission
Friend
Group Demo group
Functionality Light Lightness S...
Firmware Update

3. Go to the Networks view and tap **Demo network**. Make sure the Distributor node has the active proxy connection.



16.13

Back

Demo network

Devices | Groups

distributor aa:29
0
Active proxy connection

Firmware distribution

light node 3e:8a
1

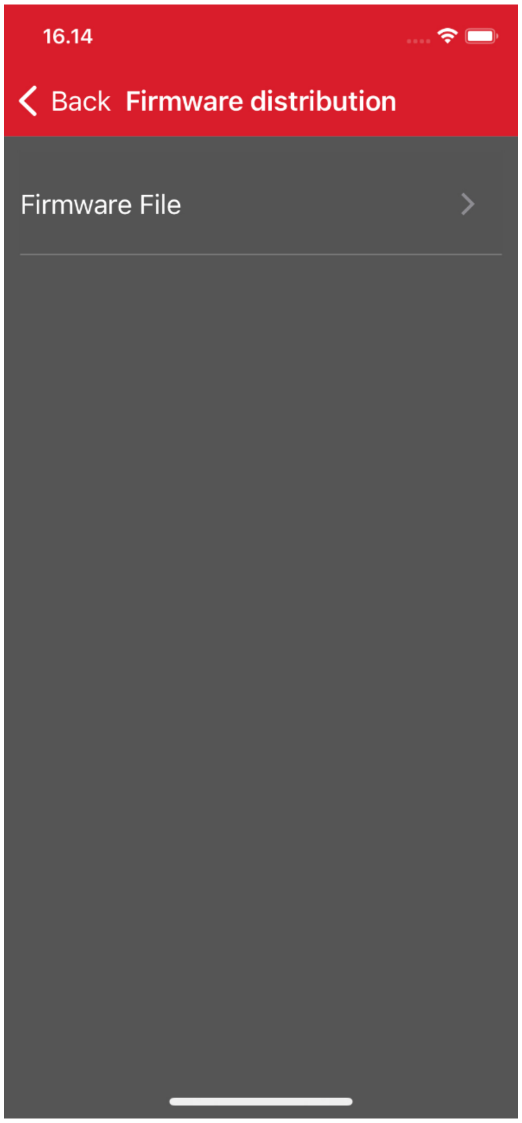
Store current node's state to scene:

Scene 1 Store

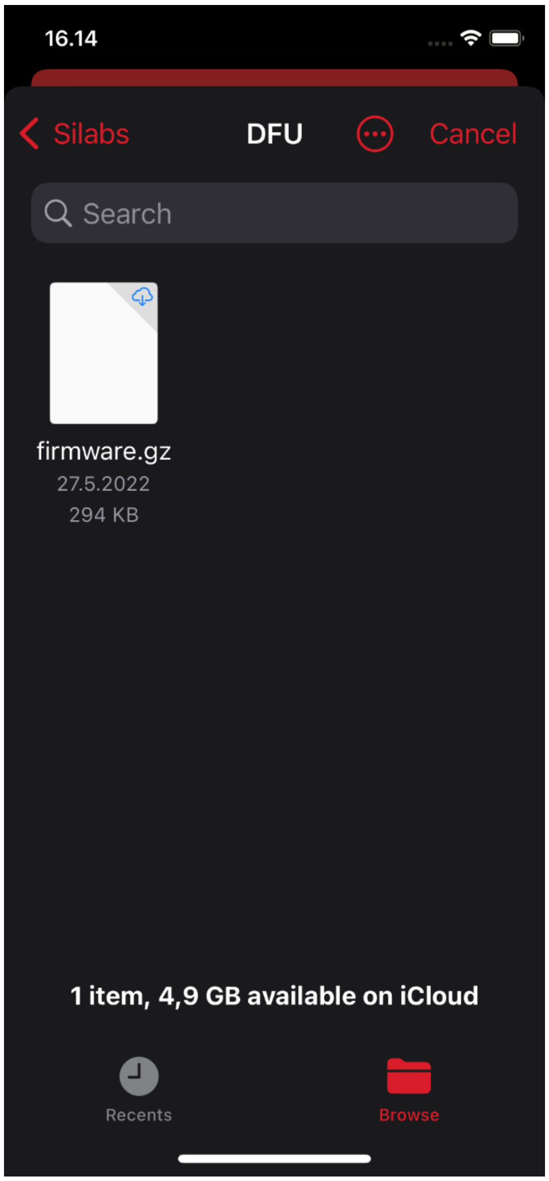
Scene 2 Store

light node 29:4a

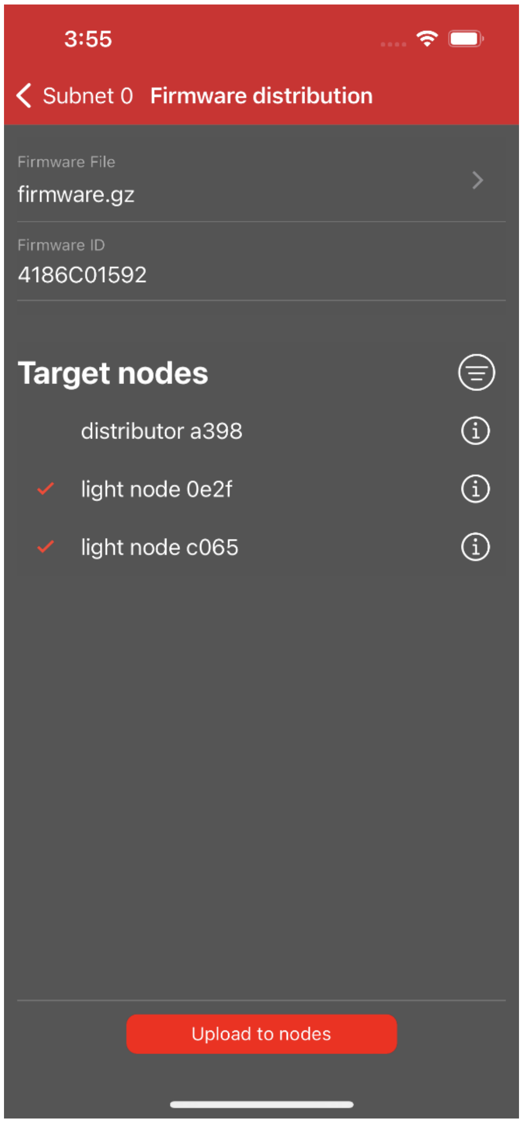
4. Tap **Firmware distribution** under the Distributor node and then tap **Firmware File**.



5. Choose an update image.



6. In the Firmware distribution view, select the nodes to update firmware, and then tap **Upload to nodes**.



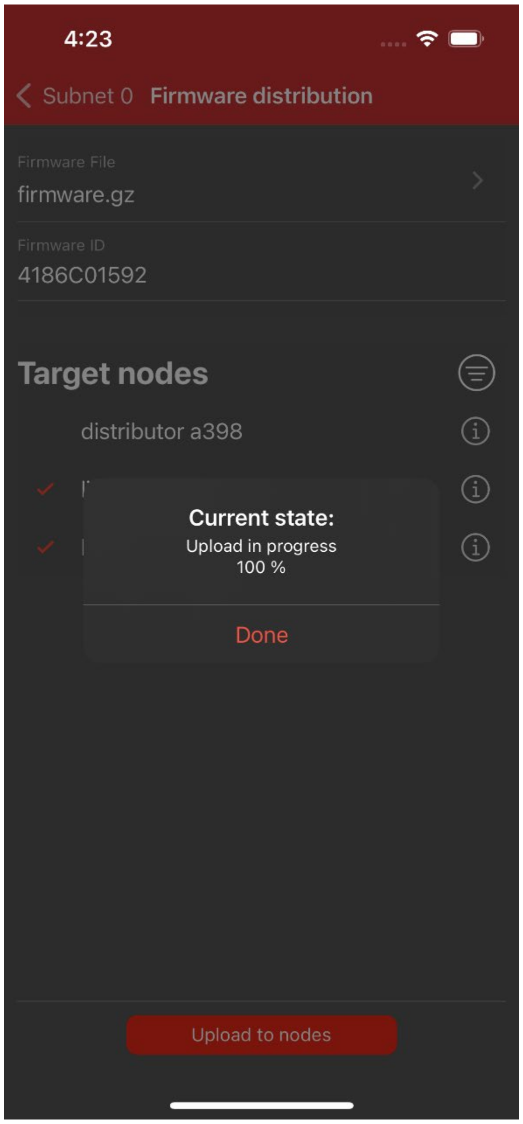
Firmware File
firmware.gz

Firmware ID
4186C01592

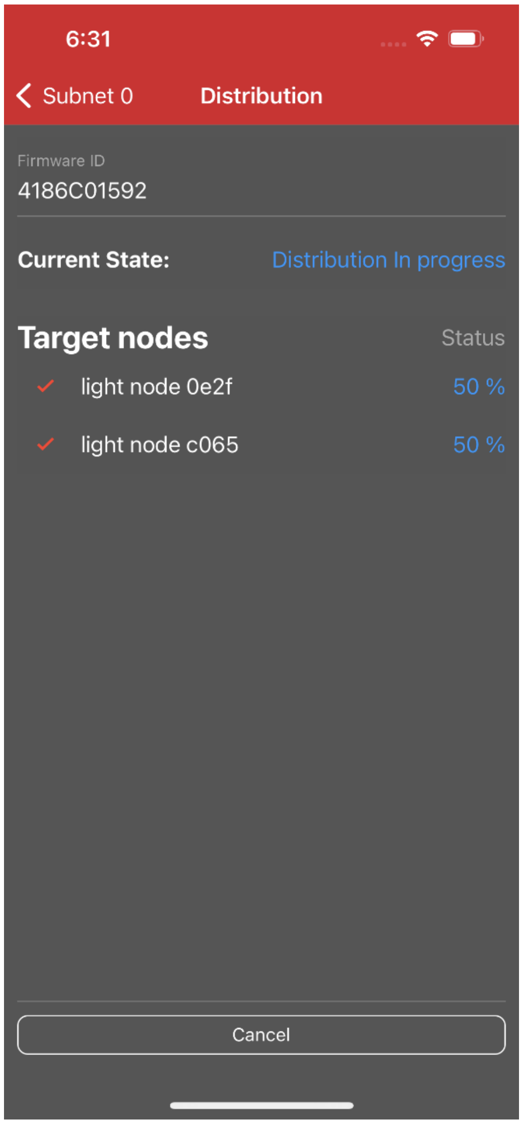
Target nodes

distributor a398
✓ light node 0e2f
✓ light node c065

7. The mobile app as the Initiator uploads the update image to the Distributor node and shows the progress. When the progress reaches 100%, tap **Done**.



8. The Distributor node then distributes the update image simultaneously to the selected nodes. The mobile app updates the distribution progress periodically.

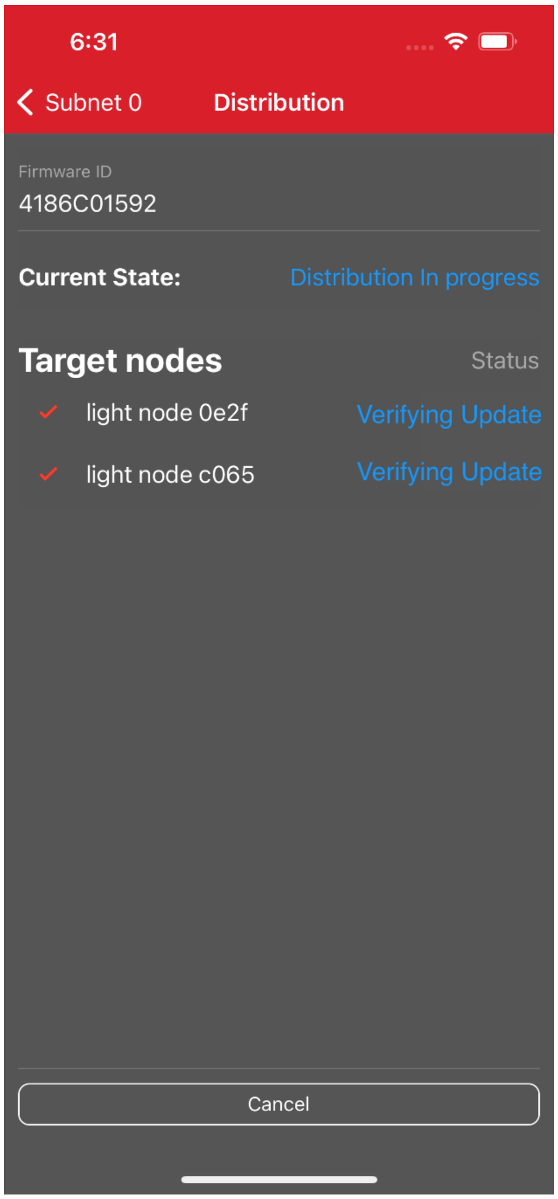


Firmware ID
4186C01592

Current State: Distribution In progress

Target nodes	Status
✓ light node 0e2f	50 %
✓ light node c065	50 %

9. When the distribution is done, the Target nodes verify the update image.



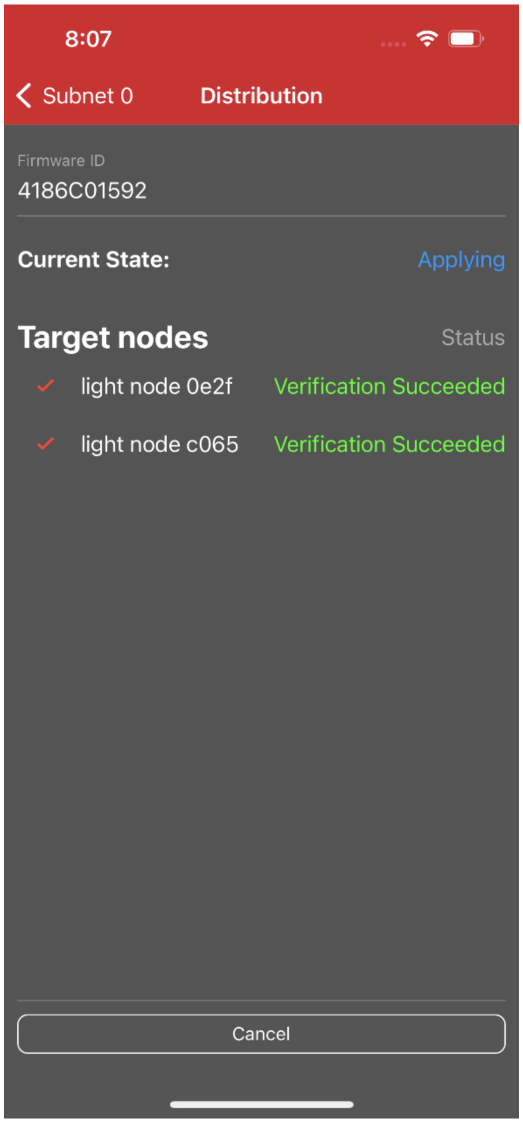
Firmware ID
4186C01592

Current State: Distribution In progress

Target nodes — Status

✓ light node 0e2f	Verifying Update
✓ light node c065	Verifying Update

10. The Distributor node then instructs the nodes that have successfully verified the update image to apply the update image.



8:07

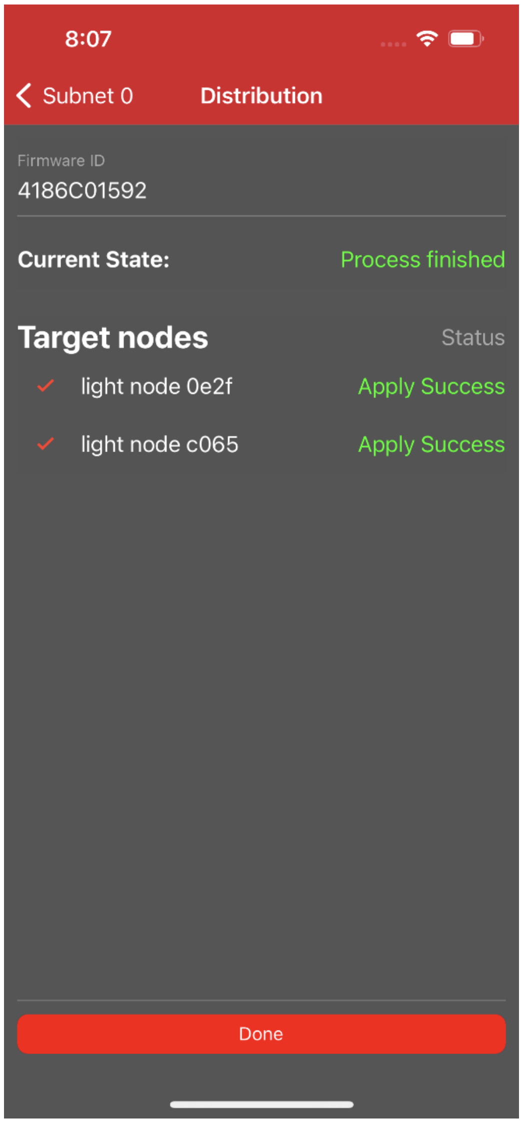
Subnet 0 — Distribution

Firmware ID
4186C01592

Current State: Applying

Target nodes	Status
✓ light node 0e2f	Verification Succeeded
✓ light node c065	Verification Succeeded

11. Finally, the mobile app shows update results.



Firmware ID
4186C01592

Current State: Process finished

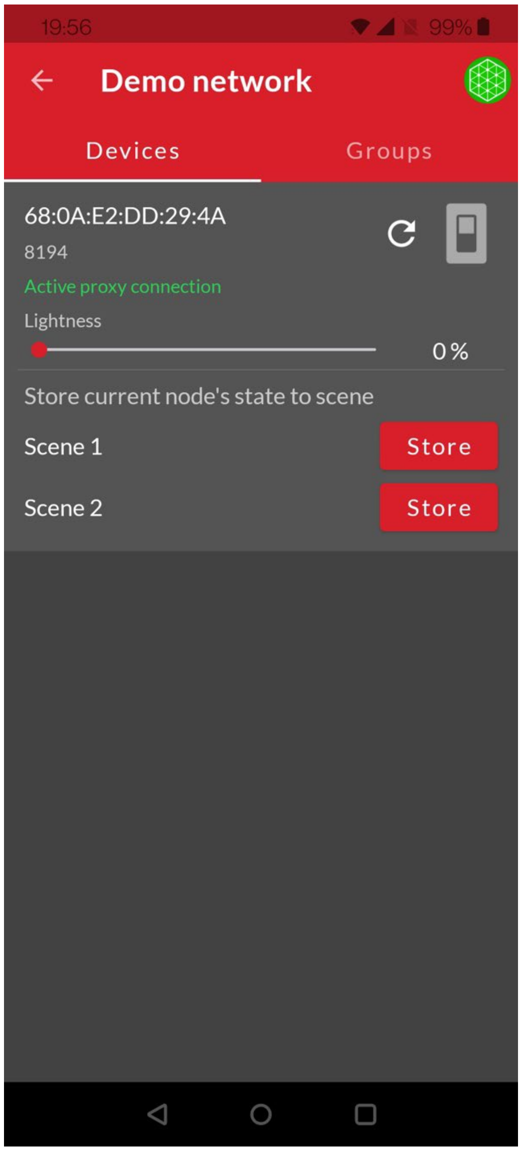
Target nodes	Status
✓ light node 0e2f	Apply Success
✓ light node c065	Apply Success

5 Firmware Update with Stand-alone Updater

The Silicon Labs Bluetooth Mesh mobile app has the Stand-alone Updater feature that manages the delivery of the firmware image to Target nodes without an intermediary Distributor. To use the feature to perform a firmware update, you should have the node whose firmware is to be updated provisioned to a mesh network.

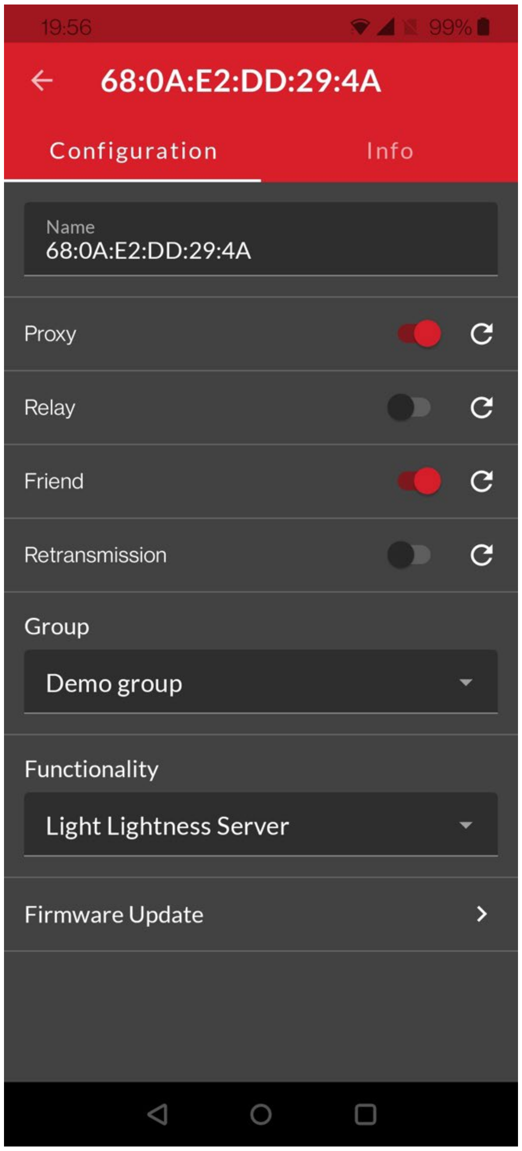
Refer to Step 2 in Section 4 Firmware Update Demonstration for provisioning and configuring a light node, and then follow the steps below to update the device’s firmware directly.

1. Go to the Networks view, tap **Demo network**, and select the light node.

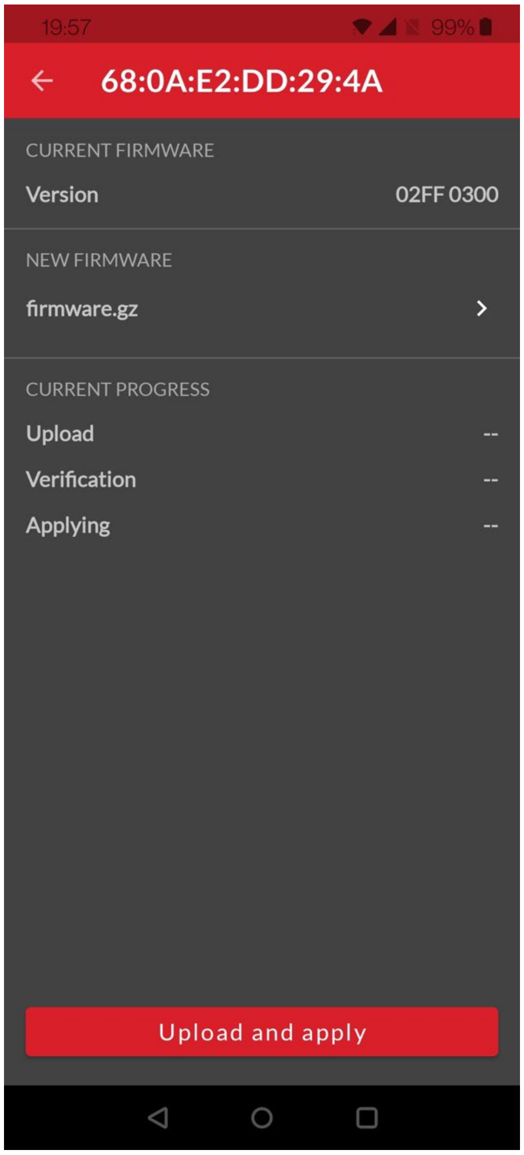


68:0A:E2:DD:29:4A
8194
Active proxy connection
Lightness
0 %
Store current node's state to scene
Scene 1 Store
Scene 2 Store

2. Tap **Firmware Update** in the Configuration view.



3. Tap **Pick a file** and choose an update image, and then tap **Upload and apply**.


19:57 99%

68:0A:E2:DD:29:4A

CURRENT FIRMWARE

Version 02FF 0300

NEW FIRMWARE

firmware.gz

CURRENT PROGRESS

Upload --

Verification --

Applying --

6 Appendix – Silicon Labs Product Positioning for Bluetooth Mesh DFU

Bluetooth Mesh DFU requires significant space in flash memory to store firmware images. The table below suggests suitable roles for Silicon Labs parts.

Part (Flash, RAM)	Distributor	Target Node
xG1		
256kB, 32kB	No	No
xG12		
512kB, 64kB	No	No (External Flash: Yes)
1MB, 256kB	Yes	Yes
xG13		
512kB, 64kB	No	No (External Flash: Yes)
xG21		
512kB, 96kB	No	No (External Flash: Yes)
768kB, 96kB	No (External Flash: Yes)	Yes
1MB, 96kB	Yes	Yes
xG22		
352kB, 32kB	No	No
512kB, 32kB	No	No (External Flash: Yes)
xG24		
1536kB, 256kB	Yes	Yes