AN1422: Provisioning and Firmware Update Using the DFU Python Script

This version of AN1422 has been deprecated with the release of Simplicity SDK Suite 2025.6.1 For the latest version, see docs.silabs.com.

****

This document walks through a device firmware update demonstration using the DFU Python script. The script is an NCP host application that requires an NCP node connected. The user can provision, configure, and manage mesh devices, as well as perform a device firmware update through the NCP node.

KEY POINTS

- Using the DFU Python script
- Using the NCP Empty example
- Bluetooth Mesh Provisioning
- Bluetooth Mesh Device Firmware Update

1 Introduction

The Bluetooth Mesh Model specification v1.1 defines a standard way to update device firmware over a Bluetooth mesh network. This document walks through a firmware update demo using the DFU Python script and the Bluetooth mesh examples, installed as part of the Bluetooth Mesh SDK. The DFU Distributor example runs as the Distributor node. The Light, Switch, Sensor Client, or Sensor Server example runs as the node whose firmware is to be updated, called the Target node, and the DFU Python script is used to provision these nodes and runs as the Initiator node.

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

2 Getting Started with the DFU Python Script

The DFU Python script is an NCP host application running on a system with a Python interpreter, and the application requires the Bluetooth mesh stack running in NCP mode on a Silicon Labs device connected to the system. Two or more additional Silicon Labs devices running the DFU examples are needed to perform a device firmware update. To run the DFU examples provided in the Bluetooth Mesh SDK, see *AN1370: Bluetooth® Mesh Device Firmware Update Example Walkthrough*.

2.1 Requirements

The following is required to run the DFU Python script.

- One mainboard with a supported board installed for the NCP target application.
- Simplicity Studio 5
- Gecko SDK Suite 4.2.2 (Bluetooth Mesh SDK 4.2.0) or later, distributed through Simplicity Studio 5. The prebuilt demos and examples are included in the SDK.
- A host system with a Python interpreter, such as MacOS, Linux, or Windows.
- The Python package PyBGAPI installed in the host system.

To install PyBGAPI, run:

```
pip3 install pybgapi
```

Run `pip3 list`. The `pybgapi` package should be version 1.2.0 or later:

```
Package    Version
---------- -------
...
pybgapi    1.2.0
pyserial   3.5
...
```

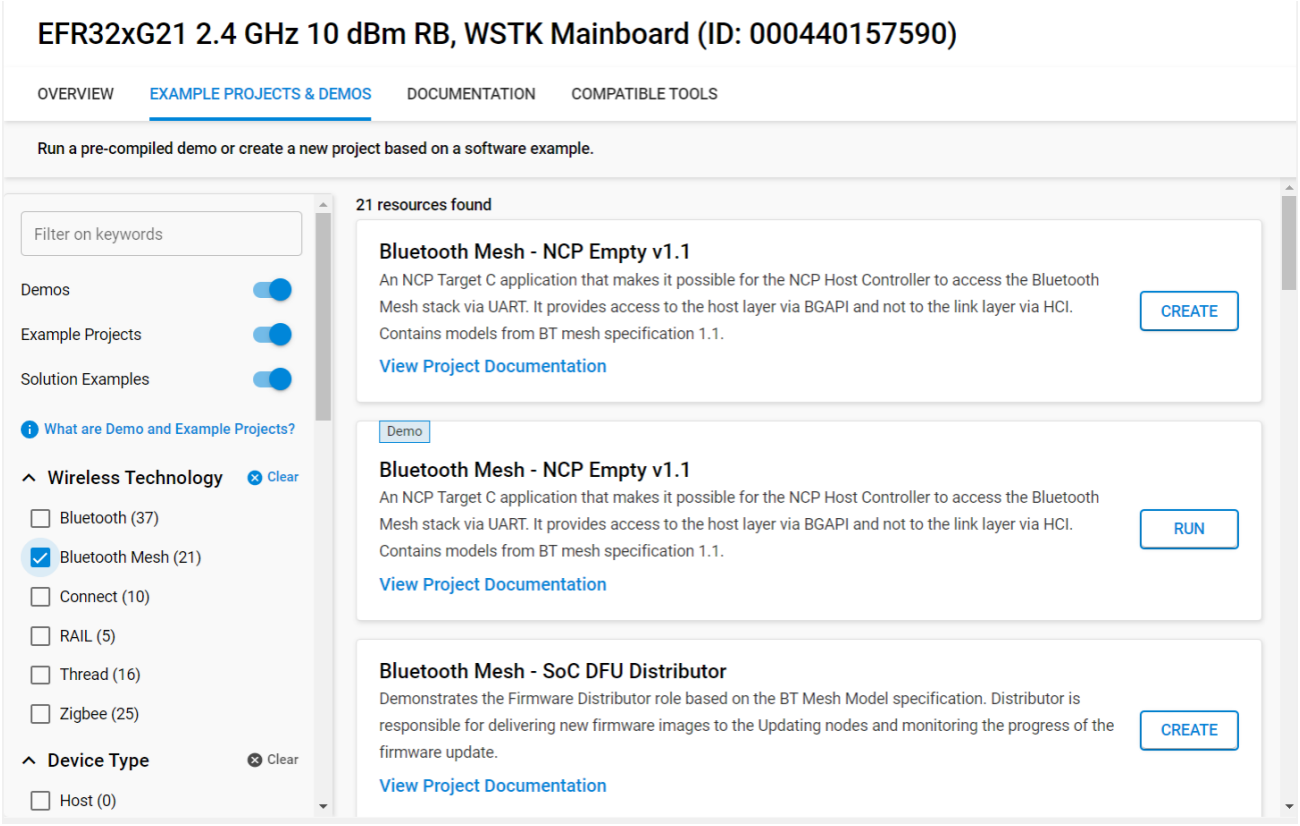
2.2 Bluetooth Mesh – NCP Empty v1.1 Application

The **Bluetooth Mesh – NCP Empty v1.1** example application is the target application running on a Silicon Labs device. The application is provided as a prebuilt demo binary image, ready to download and use, and a corresponding example project that you can modify and then build for the target part. If you want to build your own projects based on the example project, see *QSG176: Silicon Labs Bluetooth® Mesh SDK v2.x Quick-Start Guide*. This section describes how to install the prebuilt demo binary to the device.

The precompiled demos are only available for a limited set of parts, including selected EFR32xG13 and xG21 parts and BGM13 and MGM21 modules. The examples can be built for any part supported by the Bluetooth Mesh SDK.

Note: EFR32xG22 parts can run the **Bluetooth Mesh – NCP Empty v1.1** example but do not support provisioner functionality.

1. Open Simplicity Studio 5 with a compatible SoC wireless kit connected to the computer.
2. Select the part in **Debug Adapters** view to open the Launcher perspective.
3. Click the **Example Projects & Demos** tab.
4. To see only the demos, turn off the **Example Projects**.
5. Under **Technology Type**, filter on **Bluetooth Mesh**. Next to **Bluetooth Mesh – NCP Empty v1.1**, click **RUN**.



2.3 DFU Python Script

The DFU Python script is the host application running on a computer. Connect the Silicon Labs device running the **Bluetooth Mesh – NCP Empty v1.1** example application to the computer on a USB port. Check the serial port information for the connected device.

On Windows, you can find the serial port number in Device Manager, e.g. `COM7`.

On Linux, you can run the `ls /dev/ttyS*` command in a terminal to list the device file, e.g. `/dev/ttyS0`.

On MacOS, you can run the `ls /dev/tty.usb*` command in a terminal to list the device file, e.g. `/dev/tty.usbmodem0004401234561`.

Open a terminal and change the directory to the path *app/btmesh/example_host/btmesh_host_dfu* of Gecko SDK. Run `python3 btmesh_host_dfu.py --usb <usb>` and you should not get any error message.

```
python3 btmesh_host_dfu.py --usb /dev/tty.usbmodem0004401234561
```

If you are running the script in a directory other than the path to the script folder under Gecko SDK, you need to specify the API files with the `--xapi` option.

```
python3 btmesh_host_dfu.py --usb /dev/tty.usbmodem0004401234561 --xapi /Users/<user>/Simplic-
ityStudio/SDKs/gecko_sdk/protocol/bluetooth/api/sl_bt.xapi --xapi /Users/<user>/Simplic-
ityStudio/SDKs/gecko_sdk/protocol/bluetooth/api/sl_btmesh.xapi
```

The script generates 3 files on the first run:

- btmesh_host_dfu_cfg.ini – the configuration file the script reads to set command parameters and settings of model profiles. The script generates default values of the configuration options in the script if the configuration file doesn't exist.
- btmesh_host_dfu_persistence.json – the script saves its persistent states to the file and loads data from the file on each CLI command and interactive session.
- btmesh_host_dfu.log – the log file of the script's executions containing what BGAPI commands and events the script has sent and received. The log file could be helpful when something went wrong.

The script provides command line and interactive modes. You can run the script with the `--help` option to see the script usage and command descriptions in command line mode. The demonstration in Section 3 Firmware Update Demonstration Using the DFU Python Script will run the script in interactive mode.

```
python3 btmesh_host_dfu.py --usb /dev/tty.usbmodem0004401234561 --interactive
```

3 Firmware Update Demonstration Using the DFU Python Script

This section assumes you have installed the **Bluetooth Mesh – NCP Empty v1.1** demo binary to one of the devices, the **Bluetooth Mesh – SoC DFU Distributor** example application to another, and the **Bluetooth Mesh – SoC Light** example application to the other(s). You should also have generated a firmware update image for the light example. Copy the firmware update image `application.gbl` to the root of the script's directory, run the script in the interactive mode, and then follow the steps below to perform a firmware update.

Note: the DFU Python script sends the GBL file directly to the mesh network.

Section 2 Getting Started with the DFU Python Script and *AN1370: Bluetooth® Mesh Device Firmware Update Example Walkthrough* describe the setup of the examples and the preparation of the firmware update image.

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. Perform a factory reset on the provisioner.

```
>>>reset --type factory
Factory reset completed
```

2. Provision all devices.

Issue the `prov` command with the `--scan` option. The parameter `.3` stands for 300 miliseconds.

```
>>>prov --scan .3
Scanning for unprovisioned nodes...
Unprovisioned beacon: uuid=994c62c4bf480f51bb3995094a266ec2, bearer=PB-GATT, address=68:0a:e2:dd:29:4a, address_type=Public, rssi=-33
Unprovisioned beacon: uuid=5100db6c367ef95aaec40a95fee47966, bearer=PB-GATT, address=d0:cf:5e:68:aa:29, address_type=Public, rssi=-28
Unprovisioned beacon: uuid=36494ea9987db45588b29a7b900a9da2, bearer=PB-GATT, address=84:71:27:6e:f2:cd, address_type=Public, rssi=-1
...
...
+-----------+--------------------------------------+-----------+---------------------+----------------+
|   Idx     |                UUID                  |  Bearer   |       Address       |  Address Type  |
+-----------+--------------------------------------+-----------+---------------------+----------------+
|    0      |  36494ea9987db45588b29a7b900a9da2    |  PB-GATT  |  84:71:27:6e:f2:cd  |     Public     |
+-----------+--------------------------------------+-----------+---------------------+----------------+
|    1      |  36494ea9987db45588b29a7b900a9da2    |  PB-ADV   |  84:71:27:6e:f2:cd  |     Public     |
+-----------+--------------------------------------+-----------+---------------------+----------------+
|    2      |  5100db6c367ef95aaec40a95fee47966    |  PB-GATT  |  d0:cf:5e:68:aa:29  |     Public     |
+-----------+--------------------------------------+-----------+---------------------+----------------+
|    3      |  5100db6c367ef95aaec40a95fee47966    |  PB-ADV   |  d0:cf:5e:68:aa:29  |     Public     |
+-----------+--------------------------------------+-----------+---------------------+----------------+
|    4      |  994c62c4bf480f51bb3995094a266ec2    |  PB-GATT  |  68:0a:e2:dd:29:4a  |     Public     |
+-----------+--------------------------------------+-----------+---------------------+----------------+
|    5      |  994c62c4bf480f51bb3995094a266ec2    |  PB-ADV   |  68:0a:e2:dd:29:4a  |     Public     |
+-----------+--------------------------------------+-----------+---------------------+----------------+
Select devices to provision
Comma/Space separated list of indexes, BT addresses or UUIDs
```

When it prompts to select devices to provision, select the **PB-ADV** bearer of devices. In the example, the index 3 is the Distributor device and the indices 1 and 5 are the Light devices.

```
Devices:3,1,5
The device with 5100db6c367ef95aaec40a95fee47966 UUID is provisioned.
The device with 36494ea9987db45588b29a7b900a9da2 UUID is provisioned.
The device with 994c62c4bf480f51bb3995094a266ec2 UUID is provisioned.
```

3. Rename nodes for easy identification.

Issue the `node list` command to show the nodes in the mesh network.

```
>>>node list
+------------+-----------------+--------------------------------------+-------------+--------------+
|    Idx     |       Name      |                 UUID                 |   Address   |   Elements   |
+------------+-----------------+--------------------------------------+-------------+--------------+
|     0      |    Node_2005    |  5100db6c367ef95aaec40a95fee47966    |   0x2005    |      1       |
+------------+-----------------+--------------------------------------+-------------+--------------+
|     1      |    Node_2006    |  36494ea9987db45588b29a7b900a9da2    |   0x2006    |      3       |
+------------+-----------------+--------------------------------------+-------------+--------------+
|     2      |    Node_2009    |  994c62c4bf480f51bb3995094a266ec2    |   0x2009    |      3       |
+------------+-----------------+--------------------------------------+-------------+--------------+
|     3      |   Provisioner   |  764b0e71f3fc5c5ab856d6e1a90b4e32    |   0x2001    |      4       |
+------------+-----------------+--------------------------------------+-------------+--------------+
```

Issue the `node rename` command to rename nodes.

```
>>>node rename Node_2005 Distributor_2005
>>>node rename Node_2006 Light_2006
>>>node rename Node_2009 Light_2009
>>>node list
+-----------+--------------------+--------------------------------------+------------+-------------+
|    Idx    |        Name        |                 UUID                 |  Address   |  Elements   |
+-----------+--------------------+--------------------------------------+------------+-------------+
|     0     |  Distributor_2005  |  5100db6c367ef95aaec40a95fee47966    |   0x2005   |      1      |
+-----------+--------------------+--------------------------------------+------------+-------------+
|     1     |     Light_2006     |  36494ea9987db45588b29a7b900a9da2    |   0x2006   |      3      |
+-----------+--------------------+--------------------------------------+------------+-------------+
|     2     |     Light_2009     |  994c62c4bf480f51bb3995094a266ec2    |   0x2009   |      3      |
+-----------+--------------------+--------------------------------------+------------+-------------+
|     3     |     Provisioner    |  764b0e71f3fc5c5ab856d6e1a90b4e32    |   0x2001   |      4      |
+-----------+--------------------+--------------------------------------+------------+-------------+
```

4. Configure nodes with built-in profiles.

Issue the `group add` command to create a new group, specify a group address, specify a configuration profile to apply, and specify what node(s) to be added to the group.

Add the Distributor node to the group named GrpDist1:

```
>>>group add --appkey-idx 0 --group-addr 0xC000 --sub-addrs Dist*[0] --profile distributor --name GrpDist1
App group GrpDist1 adds appkey binding to BLOB Transfer Client model on 0x2005 element address.
App group GrpDist1 adds appkey binding to Firmware Update Client model on 0x2005 element address.
App group GrpDist1 adds appkey binding to Firmware Distribution Server model on 0x2005 element address.
App group GrpDist1 adds appkey binding to Firmware Update Server model on 0x2005 element address.
App group GrpDist1 adds appkey binding to BLOB Transfer Server model on 0x2005 element address.
App group GrpDist1 adds subscription to Firmware Update Server model on 0x2005 element address.
App group GrpDist1 adds subscription to BLOB Transfer Server model on 0x2005 element address.
```

Add the Light node(s) to the group named GrpLight1:

```
>>>group add --appkey-idx 0 --group-addr 0xC001 --sub-addrs Light*[0] --profile target_node --name GrpLight1
App group GrpLight1 adds appkey binding to Firmware Update Server model on 0x2009 element address.
App group GrpLight1 adds appkey binding to BLOB Transfer Server model on 0x2009 element address.
App group GrpLight1 adds appkey binding to Firmware Update Server model on 0x2006 element address.
App group GrpLight1 adds appkey binding to BLOB Transfer Server model on 0x2006 element address.
App group GrpLight1 adds subscription to Firmware Update Server model on 0x2009 element address.
App group GrpLight1 adds subscription to BLOB Transfer Server model on 0x2009 element address.
App group GrpLight1 adds subscription to Firmware Update Server model on 0x2006 element address.
App group GrpLight1 adds subscription to BLOB Transfer Server model on 0x2006 element address.
```

Show the group information:

```
>>>group list
+------------------+----------------------+--------------------------+--------------------------+
|        Idx       |         Name         |       Group Address      |       Appkey Index       |
+------------------+----------------------+--------------------------+--------------------------+
|         0        |       GrpDist1       |          0xC000          |            0             |
+------------------+----------------------+--------------------------+--------------------------+
|         1        |       GrpLight1      |          0xC001          |            0             |
+------------------+----------------------+--------------------------+--------------------------+
```

5. Upload a firmware image to the Distributor.

Issue the `dist upload` command to upload `application.gbl` to the Distributor.

```
>>>dist upload --distributor Distributor_2005 --fwid 0x02FF:s:light --metadata s:test --timeout-base 1 applica-
tion.gbl
FW data (427260 bytes) is loaded from application.gbl.
Upload progress:  0.00%
...
...
...
Upload progress: 100.00%
Firmware with 0x02FF:light FWID is uploaded to Distributor (0x2005).
```

Show the firmware list from the Distributor.

```
>>>dist info --fw-list --distributor Distributor_2005
+---------------------------------------------+-------------------------------------------------------+
|                    Index                    |                         FWID                          |
+---------------------------------------------+-------------------------------------------------------+
|                      0                      |                    0x02FF:light                       |
+---------------------------------------------+-------------------------------------------------------+
```

6. Distribute the firmware image to Light node(s).

Issue the `dist start` command to instruct the Distributor to start firmware image distribution.

```
>>>dist start --distributor Distributor_2005 --fw-list-idx 0 --group GrpLight1
Distribution phase is changed to transfer active.
Distribution transfer progress:   0%
...
...
Distribution transfer progress: 100%
Distribution phase is changed to applying update.
Distribution phase is changed to completed.
The FW distribution of 0 FW list index is completed on the Distributor (0x2005).
+-----------------+----------------+----------------------+--------------------+--------------------+
|     Address     |     FW Idx     |        Phase         |    BLOB status     |    DFU status      |
+-----------------+----------------+----------------------+--------------------+--------------------+
|     0x2006      |       0        |    apply success     |      success       |      success       |
+-----------------+----------------+----------------------+--------------------+--------------------+
|     0x2009      |       0        |    apply success     |      success       |      success       |
+-----------------+----------------+----------------------+--------------------+--------------------+
```