AN1371: *Bluetooth*® Mesh NCP Host Provisioner Example Walkthrough

The NCP Host Provisioner example demonstrates how to run a provisioner on a computer with a NCP node connected. The user can provision, configure, and reset other nodes through the NCP node. The Bluetooth mesh network is created and handled by the NCP node and therefore network management options are also available.

KEY POINTS

- Short introduction to Bluetooth mesh
- Using the NCP Empty example application
- Using the NCP host provisioner example application

1 Introduction

This document explains the Bluetooth mesh NCP host provisioner example, installed as part of the Bluetooth Mesh SDK. Most of the documentation focuses on the example application and its usage flow. This document also introduces some concepts of the specification that are important for understanding the example.

The following subsections briefly describe the relevant aspects of the Bluetooth mesh technology. Section 2 Getting Started with the NCP Host Provisioner Example describes the setup to run the example, and Section 3 Running the NCP Host Provisioner Example describes the features of the example.

1.1 Bluetooth Mesh Nodes and Features

A Bluetooth mesh network can consist of different types of nodes where not all nodes need to be equal. Some nodes relay messages while others do not, and some nodes can be battery operated low power nodes. This section provides a short overview of the different features Bluetooth mesh nodes can implement. First, all Bluetooth mesh nodes can receive and send messages, and all Bluetooth mesh nodes must implement the Bluetooth mesh security and the essential mesh models needed for configuration. The rest of the node functionality is optional.

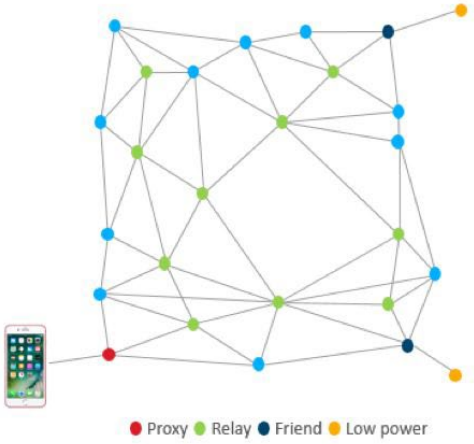


Figure 1-1: A Bluetooth Mesh Network

Relay Feature

Nodes with the relay feature are capable of relaying messages from other nodes and are essential in increasing the scale and range of a Bluetooth mesh network.

Proxy Feature

A proxy node acts as a proxy between the Bluetooth mesh nodes and network, and a device that only implements the GATT bearer, such as smart phones today. This means proxy nodes must implement both Bluetooth LE and Bluetooth mesh stacks and are the only nodes in the mesh network that must do this.

Low Power Feature

Nodes with the low power feature can spend most of their lifetime in a low power sleep mode and only need to wake up and participate in the Bluetooth mesh network communications once every four (4) days, which means their duty cycle can be almost zero. Nodes with the low power feature, however, need a node with a friend feature, which caches any messages targeted to the low power nodes while they are sleeping. When low power nodes are provisioned to the mesh network, they search for a nearby friend, agree on a communication interval with the friend when they will poll for messages, and after that they can go to sleep.

Friend Feature

Nodes with the friend feature must implement an additional message cache they use to cache messages for the nodes with the low power feature and store them until low power nodes wake up and fetch the messages. Nodes with the friend feature can also acknowledge messages on behalf of low power nodes, while they are sleeping. Nodes with friend features also advertise their capabilities using special beacons, so low power nodes can select the most optimal friends to associate with. The table below summarizes the Bluetooth mesh node features, but it is possible to combine features, so a node can have multiple features like Relay, Proxy, and Friend features.

Table 1.1. Bluetooth Mesh Node Feature Comparison

	Relay	Proxy	Low Power	Friend
Send Messages	Yes	Yes	Yes	Yes
Receive Messages	Yes	Yes	Yes	Yes
Relay Messages	Yes	Yes	No	Yes
GATT bearer	Yes/No	Yes	Yes/No	Yes/No
Battery operated	Typically no	Typically no	Yes	Typically no

1.2 Provisioning, Configuration, and Node Lifecycle

The lifecycle of a Bluetooth mesh node starts as an unprovisioned device, meaning it has not been added as part of any mesh network, nor has it been configured to operate in a network. The process of adding such a device as part of a Bluetooth mesh network is called provisioning and can be done, for example, using a smart phone application or during production time when devices are manufactured and firmware gets installed. An unprovisioned Bluetooth mesh device may send Bluetooth LE advertisement packets, which announce that it supports the GATT provisioning service. The GATT provisioning service allows a Bluetooth LE device with provisioning capabilities to establish a connection to the unprovisioned device and start the provisioning process.

Alternatively, the unprovisioned mesh device can start sending unprovisioned mesh beacons allowing another Bluetooth mesh node with provisioning capabilities to provision it over the mesh bearer. In the Bluetooth Mesh 1.0 specification, the provisioning can only be done over a single hop, but a future version of the specification may allow devices to be provisioned over multiple hops making this feature much more versatile. The Bluetooth mesh device provisioning is a secure process in which the provisioner typically performs the following actions:

- The provisioner assigns the device a network key used for authenticating and encrypting the mesh communications. The key is transferred to the device being provisioned in an encrypted format.
- The provisioner assigns a unicast address for each individual element the device has. The unicast address is unique in the mesh network. The address of the device is usually the address of its primary element.
- During the process, device-specific keys are also generated both at the device and the provisioner, and they are used for any future device management or configuration operations. The device-specific keys are never sent over the air. They are generated and stored locally. Once the above steps have been made, an unprovisioned Bluetooth mesh device becomes a mesh node.

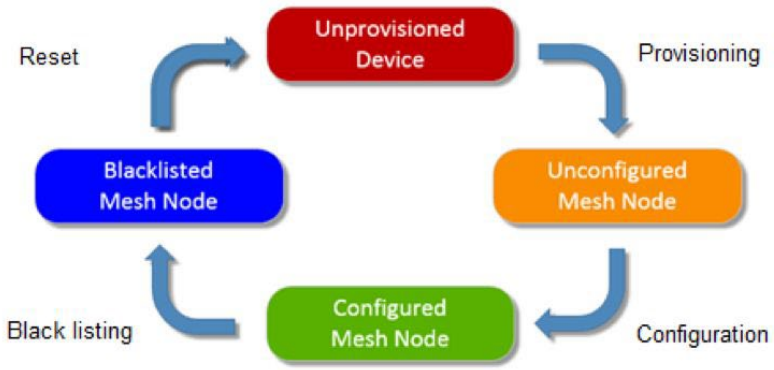


Figure 1-2: Lifecycle of a Bluetooth Mesh Node

The typical next step after device provisioning is device configuration. This again can be done by the provisioner, such as a smart phone application. To ensure that the mesh node is operational in a Bluetooth mesh network, the following steps are typically needed:

- The Device Composition Data (DCD) is read from the mesh node. The DCD contains the models supported by the device, the manufacturer information, and information on which features (proxy, relay etc.) are supported by the node.
- Depending on the node capabilities, certain node features (proxy, relay etc.) can be enabled or disabled.
- The network settings, such as addresses and time-to-live counter values, are configured.
- One or more application keys are generated and assigned to the node depending on the applications the node needs to support.
- The publish and subscribe configuration is defined indicating to which groups the node sends messages to and which group messages it listens to.

The configuration of a node can be changed any time during its lifetime.

A Bluetooth mesh node may need to be removed from a Bluetooth mesh network at some point, for example, when a broken device is replaced or a device gets stolen. Bluetooth mesh network management operations provide two ways of removing a Bluetooth mesh node from the network:

1. A node can be informed that it will be removed from the network, so it can behave accordingly, but this operation should not be used as an only node removal process. Instead, the node removal from the list operation described below should be done to securely remove a node from a network.
2. A key refresh operation can be performed in the whole network meaning every other node in the network is assigned with new network and application keys except the node to be removed. This operation is a heavier process but guarantees the node to be removed is removed from the network.

2 Getting Started with the NCP Host Provisioner Example

The NCP host provisioner example consists of two components: NCP target and host applications. The NCP target application runs on a Silicon Labs device and the host application runs on a MacOS, Linux, or Windows system with the Silicon Labs device connected. One or more additional Silicon Labs devices running Bluetooth mesh applications are needed so you can use the host provisioner to add these devices to a mesh network and to manage these devices.

2.1 Requirements

The following is required to run the example.

- At least two mainboards with a supported board installed, one used for the target application of the NCP host provisioner and the other(s) for Bluetooth mesh example application(s) to be provisioned.
- Simplicity Studio 5
- Bluetooth Mesh SDK 2.2.0 or later, distributed through Simplicity Studio 5. The prebuilt demos and examples are included in the SDK.
- A Mac or Linux computer or MSYS2 on a Windows computer with GCC toolchain installed.

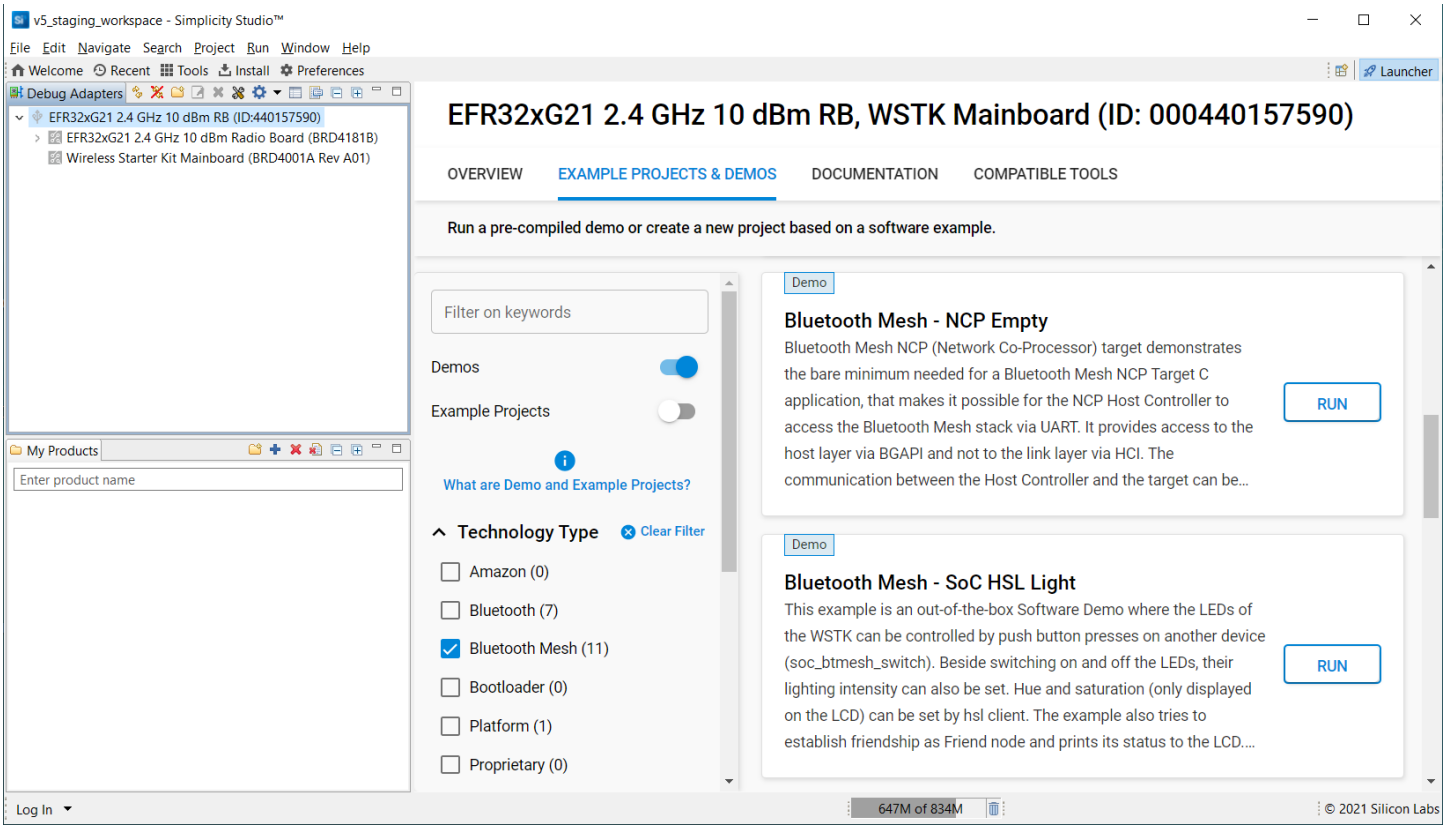
2.2 Bluetooth Mesh – NCP Empty Application

The **Bluetooth Mesh – NCP Empty** 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** 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**, click **RUN**.



2.3 Bluetooth Mesh NCP Host Provisioner Application

The Bluetooth mesh NCP host provisioner application is the host application running on a computer. A Silicon Labs device running the **Bluetooth Mesh – NCP Empty** example application is connected to the computer over a USB simulated serial port.

The application source code can be found in the path *app/btmesh/example_host/btmesh_host_provisioner* of Gecko SDK Suite. To build the application, open a terminal, change the directory to the above directory and run `make`. The executable will be generated in the **exe** directory. To delete the executable and intermediate files, run `make clean`. If you need to modify SDK files, you can run `make export` to copy all source and header files to the **export** directory, or `make export <export_dir>` to copy all source and header files to a specified directory, and then modify the files in the exported directory.

If you develop your own project based on the application or build the application somewhere other than the default path, you need to specify the GSDK path to the variable `SDK_DIR` in the 'makefile' file.

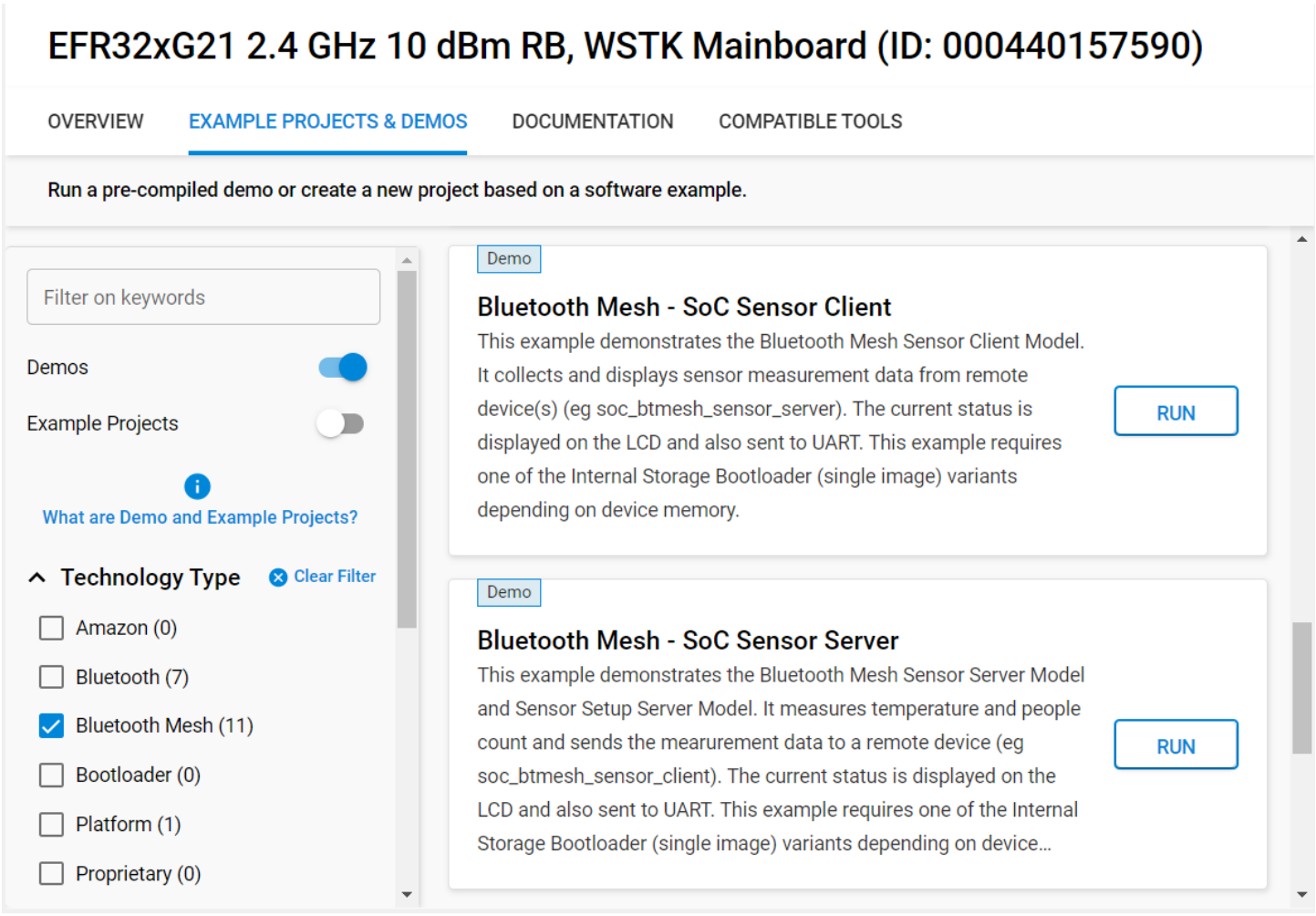
2.4 Bluetooth Mesh Example Applications

To use the host provisioner to perform the provisioning and configuration process, you need to have one or more Bluetooth mesh devices running to be added to a mesh network. In this example, we use two Silicon Labs devices to run Sensor Server and Sensor Client example applications.

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.

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 demos, turn off the **Example Projects**.
3. Under Technology Type, filter on **Bluetooth Mesh**.
4. Next to either **Bluetooth Mesh – SoC Sensor Client** or to **Bluetooth Mesh – SoC Sensor Server**, click **RUN**.
5. Connect your other device and repeat with the other application.


EFR32xG21 2.4 GHz 10 dBm RB, WSTK Mainboard (ID: 000440157590)
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
What are Demo and Example Projects?
Technology Type
Clear Filter
Amazon (0)
Bluetooth (7)
Bluetooth Mesh (11)
Bootloader (0)
Platform (1)
Proprietary (0)
Demo
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) (eg soc_btmesh_sensor_server). The current status is displayed on the LCD and also sent to UART. This example requires one of the Internal Storage Bootloader (single image) variants depending on device memory.
RUN
Demo
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 mearurement data to a remote device (eg soc_btmesh_sensor_client). The current status is displayed on the LCD and also sent to UART. This example requires one of the Internal Storage Bootloader (single image) variants depending on device...
RUN

3 Running the NCP Host Provisioner Example

This section assumes you have installed the **Bluetooth Mesh – NCP Empty** demo binary to one of the devices, the **Bluetooth Mesh – SoC Sensor Client** to another, and the **Bluetooth Mesh – SoC Sensor Server** to the other, and have built the NCP host provisioner application.

3.1 Launching NCP Host Provisioner

Connect the SoC wireless kit running the **Bluetooth Mesh – NCP Empty** application to the computer. Open a terminal on Mac/Linux or a MSYS2 terminal on Windows and change the directory to the application directory. Check what serial port is assigned for the SoC wireless kit and run `./exe/btmesh_host_provisioner[.exe] -u <serial_port> [host provisioner commands]`. If the computer and the SoC wireless kit are connected to the same TCP/IP network, you can find its IP address on the LCD of the SoC wireless kit and run `./exe/btmesh_host_provisioner[.exe] -t <tcp_address> [host provisioner commands]`.

The application supports the command line options described below. You can run `./exe/btmesh_host_provisioner[.exe] -h` to see the usage.

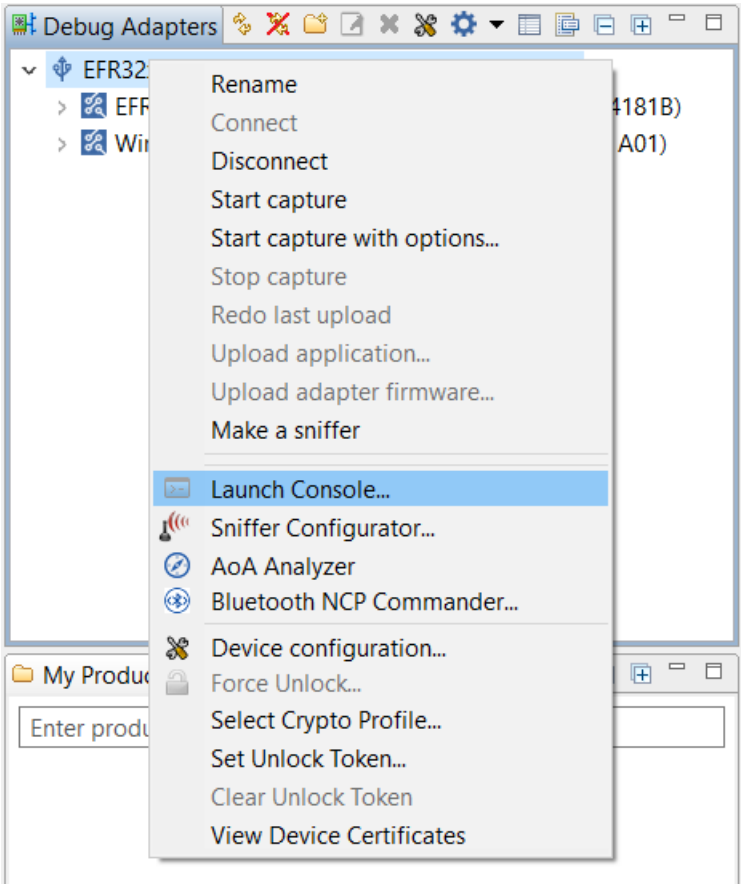
NCP Host Commands:

Command	Description
-t <tcp_address>	TCP/IP connection option. <tcp_address> is the TCP/IP address of the dev board.
-u <serial_port>	UART serial connection option. <serial_port> is the serial port assigned to the dev board by the host system. (COM# on Windows, /dev/tty# on POSIX)
-b <baud_rate>	Baud rate of the serial connection. <baud_rate> is the baud rate, default: 115200
-f	Disable flow control (RTS/CTS), default: enabled
-h	Print this help message

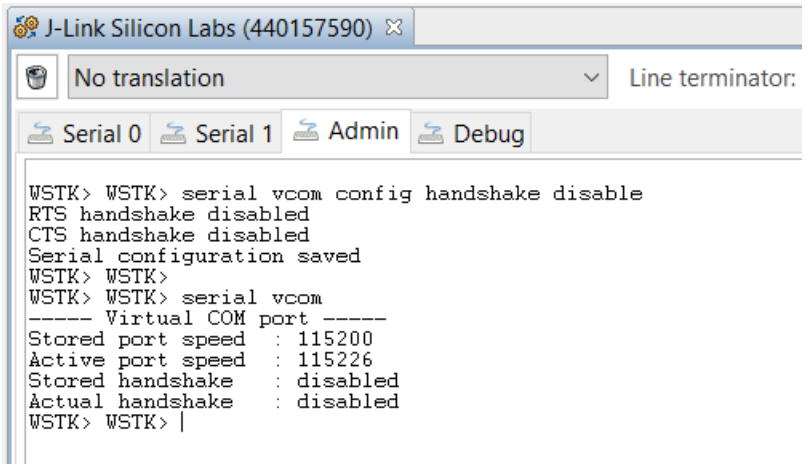
Host Provisioner Commands:

Command	Description
-i or --nodeinfo <UUID>	Print DCD information about a node in the network. <UUID> is the unique identifier of the node.
-l or --nodelist	List all nodes present in the provisioner's device database (DDB)
-p or --provision <UUID>	Provision a node. <UUID> is the UUID of the node to be provisioned. Can be acquired by --scan.
-r or --remove <UUID>	Remove the given node from the mesh network. <UUID> is the UUID of the node to be removed.
-k or --key-refresh <timeout>	Refresh the network key and app key. <timeout> is the phase timeout in seconds
-x or --key-export <filename>	Export the network, app, and device keys in JSON. <filename> is the output file name
-e or --reset	Factory reset the provisioner. Note: This command does not remove existing devices from the network.
-s or --scan	Scan and list unprovisioned beaconing nodes.
UUID shall be a string containing 16 separate octets, e.g. "00 11 22 33 44 55 66 77 88 99 aa bb cc dd ee ff" The separator can be any character, but in case of a whitespace character this example requires quotation marks around the string.	

If the application doesn't respond, you may need to disable flow control by using the `-f` option. If the option does not help, please disable it in the WSTK console. Open Simplicity Studio 5, right-click on the part in Debug Adapters view, and select **Launch Console**.



Click the **Admin** tab in the right panel and type the command `serial vcom config handshake disable`.



The Host Provisioner example has two modes: CLI and UI.

- CLI mode
  - The CLI mode is one command per run.
  - The host database is not preserved between runs, but node identifiers (for example, UUID) remain the same provided the affected node has not been reset in the meantime. This means that a UUID found in a scanning session can be used in a provisioning session.
- UI mode
  - The UI mode is accessible by starting the program without any host provisioner commands, for example, `btmesh_host_provisioner.exe -u COM5 or btmesh_host_provisioner -u /dev/tty<X>`.
  - The user can choose from several commands in one session without exiting. In this case, the host example's database stores information about the nodes in the Bluetooth mesh network and those found during scanning.

- This database is updated while the program is running, but it is not guaranteed that it will be the same in the next run.
- On Windows using MSYS2, `winpty` may be required for user input handling, e.g. `winpty exe/btmesh_host_provisioner.exe -u COM5`. If you haven't installed the winpty package, run `pacman -S winpty` to install the package in a MSYS2 terminal.

3.2 Resetting the provisioner

Run the NCP host provisioner with the `-e or --reset` argument or input **e** in the UI mode to reset the NCP node.

```
% ./exe/btmesh_host_provisioner -u /dev/tty.usbmodem0004402253201 -e
[I] Factory reset
[I] Empty NCP-host initialised.
[I] Resetting NCP...
[I] Bluetooth stack booted: v5.1.2-b215
[I] Provisioner init
[I] Network initialized
[I] Networks: 0
[I] Address : 0
[I] IV Index: 0
[I] Initiating node reset
[I] Resetting hardware
```

3.3 Scanning unprovisioned nodes

Run the NCP host provisioner with the `-s or --scan` argument or input **s** in the UI mode to scan the nodes that are transmitting unprovisioned beacons.

```
% ./exe/btmesh_host_provisioner -u /dev/tty.usbmodem0004402253201 -s
[I] Scan
[I] Empty NCP-host initialised.
[I] Resetting NCP...
[I] Bluetooth stack booted: v5.1.2-b215
[I] Provisioner init
[I] Network initialized
[I] Networks: 0
[I] Address : 0
[I] IV Index: 0
[I] Scanning started

[I] Unprovisioned node
[I] ID:      0
[I] UUID:    fa e1 09 0e 2c 78 60 53 bf 90 1c 91 22 a1 5d bc
[I] OOB Capabilities: 0x0000

[I] Unprovisioned node
[I] ID:      1
[I] UUID:    52 58 b4 a4 03 a5 45 57 87 e2 dc 14 2e a0 c5 46
[I] OOB Capabilities: 0x0000

[I] Scanning stopped
```

3.4 Provisioning and configuring a node

To provision and configure a device, run the NCP host provisioner with the `-p or --provision` argument followed the UUID of the device, or input **p** in the UI mode and then select the device with its ID or UUID. Repeat the command until all devices are provisioned. Note that a scan is required prior to running the command in the UI mode. See section 3.3 Scanning unprovisioned nodes.

```
% ./exe/btmesh_host_provisioner -u /dev/tty.usbmodem0004402253201 -p "54 16 d3 83 64 c3 21 54 84 87 6a e4 9b 76 08
b2"
[I] Empty NCP-host initialised.
[I] Resetting NCP...
[I] Bluetooth stack booted: v5.1.2-b215
[I] Provisioner init
[I] Network initialized
[I] Networks: 0
[I] Address : 0
[I] IV Index: 0
[I] Starting provisioning session

[I] Device provisioned
[I] UUID:    54 16 d3 83 64 c3 21 54 84 87 6a e4 9b 76 08 b2
[I] Address: 0x2005

Configuration of node (netkey_idx=0,addr=0x2005) is started.
...
DCD query of node (netkey_idx=0,addr=0x2005) completed.
Node (netkey_idx=0,addr=0x2005) runs btmesh_soc_switch example.
...
Configuration of node (netkey_idx=0,addr=0x2005) is successful.

[I] Provisioning session finished
```

3.5 Listing all nodes

Run the NCP host provisioner with the `-l or --nodelist` argument or input **l** in the UI mode to list all nodes that have been provisioned and configured.

```
% ./exe/btmesh_host_provisioner -u /dev/tty.usbmodem0004402253201 -l
[I] Nodelist
[I] Empty NCP-host initialised.
[I] Resetting NCP...
[I] Bluetooth stack booted: v5.1.2-b215
[I] Provisioner init
[I] Network initialized
[I] Networks: 1
[I] Address : 2001
[I] IV Index: 0
[I] Querying DDB list

[I] Address: 0x2005
[I] Element count: 1
[I] UUID:    54 16 d3 83 64 c3 21 54 84 87 6a e4 9b 76 08 b2
```

3.6 Displaying DCD information

To display the DCD information of a node, run the NCP host provisioner with the `-i or --nodeinfo` argument followed by the UUID of the node, or input **i** in the UI mode and then select the node with its ID, UUID, or Address.

```
% ./exe/btmesh_host_provisioner -u /dev/tty.usbmodem0004402253201 -i "54 16 d3 83 64 c3 21 54 84 87 6a e4 9b 76 08
b2"
[I] Nodeinfo: 54 16 d3 83 64 c3 21 54 84 87 6a e4 9b 76 08 b2
[I] Empty NCP-host initialised.
[I] Resetting NCP...
[W] Failed to add node to network in database!
[I] Bluetooth stack booted: v5.1.2-b215
[I] Provisioner init
[I] Network initialized
[I] Networks: 1
[I] Address : 2001
[I] IV Index: 0
[I] Querying node information

[I] Task DCD get (page=0) request to node (netkey_idx=0,addr=0x2005,handle=0x00000001) is sent.
[W] Task DCD get (page=0) request (retry) to node (netkey_idx=0,addr=0x2005,handle=0x00000002) is sent.
[I] Node (netkey_idx=0,addr=0x2005) DCD company id: 0x02ff
[I] Node (netkey_idx=0,addr=0x2005) DCD product id: 0x0006
[I] Node (netkey_idx=0,addr=0x2005) DCD version id: 0x0421
[I] Node (netkey_idx=0,addr=0x2005) DCD min replay prot list length: 32
[I] Node (netkey_idx=0,addr=0x2005) DCD feature relay:  1
[I] Node (netkey_idx=0,addr=0x2005) DCD feature proxy:  1
[I] Node (netkey_idx=0,addr=0x2005) DCD feature friend: 0
[I] Node (netkey_idx=0,addr=0x2005) DCD feature lpn:    1
[I] Node (netkey_idx=0,addr=0x2005) DCD element index 0 with location 0x0000 (sig_models=8,vendor_models=0)
[I] Node (netkey_idx=0,addr=0x2005) DCD SIG model 0x0000-Configuration Server (elem=0)
[I] Node (netkey_idx=0,addr=0x2005) DCD SIG model 0x0002-Health Server (elem=0)
[I] Node (netkey_idx=0,addr=0x2005) DCD SIG model 0x1001-Generic OnOff Client (elem=0)
[I] Node (netkey_idx=0,addr=0x2005) DCD SIG model 0x1302-Light Lightness Client (elem=0)
[I] Node (netkey_idx=0,addr=0x2005) DCD SIG model 0x1205-Scene Client (elem=0)
[I] Node (netkey_idx=0,addr=0x2005) DCD SIG model 0x1402-UnknownSigModel (elem=0)
[I] Node (netkey_idx=0,addr=0x2005) DCD SIG model 0x1305-Light CTL Client (elem=0)
[I] Node (netkey_idx=0,addr=0x2005) DCD SIG model 0x1400-UnknownSigModel (elem=0)
[I] Task DCD get (page=0) of node (netkey_idx=0,addr=0x2005,handle=0x00000002) is completed successfully.
Node 0x2005 information:
Company id: 0x02ff-Silicon Labs
Product id: 0x0006-btmesh_soc_switch
Version id: 0x0421
Min replay prot list length: 32
Features:
  -Relay:  1
  -Proxy:  1
  -Friend: 0
  -LPN:    1
Elements: (count=1)
  -Element 0:
    -Address: 0x2005
    -Location: 0
    -Models: (count=8)
      -SIG model: 0x0000-Configuration Server
      -SIG model: 0x0002-Health Server
      -SIG model: 0x1001-Generic OnOff Client
      -SIG model: 0x1302-Light Lightness Client
      -SIG model: 0x1205-Scene Client
      -SIG model: 0x1402-UnknownSigModel
      -SIG model: 0x1305-Light CTL Client
      -SIG model: 0x1400-UnknownSigModel
```

3.7 Removing a node

To remove a node from the mesh network, run the NCP host provisioner with the `-r or --remove` argument followed by the UUID of the node, or input **r** in the UI mode and then select the node with its ID, UUID, or Address.

```
% ./exe/btmesh_host_provisioner -u /dev/tty.usbmodem0004402253201 -r "54 16 d3 83 64 c3 21 54 84 87 6a e4 9b 76 08
b2"
[I] Remove: 54 16 d3 83 64 c3 21 54 84 87 6a e4 9b 76 08 b2
[I] Empty NCP-host initialised.
[I] Resetting NCP...
[I] Bluetooth stack booted: v5.1.2-b215
[I] Provisioner init
[I] Network initialized
[I] Networks: 1
[I] Address : 2001
[I] IV Index: 0
[I] Unprovisioning...
[I] Task node reset request to node (netkey_idx=0,addr=0x2005,handle=0x00000001) is sent.
[W] Task node reset request (retry) to node (netkey_idx=0,addr=0x2005,handle=0x00000002) is sent.
[I] Task node reset of node (netkey_idx=0,addr=0x2005,handle=0x00000002) is completed successfully.
[I] Node removed from network
[I] Refresh keys to prevent trashcan attacks.
```

3.8 Refreshing the keys

To refresh the network and app keys, run the NCP host provisioner with the `-k or --key-refresh` argument followed by the timeout of the command, or input **k** in the UI mode (timeout defaults to the stack's value or to the previously set).

```
% ./exe/btmesh_host_provisioner -u /dev/tty.usbmodem0004402253201 -k 30
[I] Key refresh with phase timeout: 30
[I] Empty NCP-host initialised.
[I] Resetting NCP...
[I] Bluetooth stack booted: v5.1.2-b215
[I] Provisioner init
[I] Network initialized
[I] Networks: 1
[I] Address : 2001
[I] IV Index: 0
[I] Key refresh started
[I]   Phase 1 succeed: 54 16 d3 83 64 c3 21 54 84 87 6a e4 9b 76 08 b2
[I] Key refresh phase 1
[I]   Phase 2 succeed: 54 16 d3 83 64 c3 21 54 84 87 6a e4 9b 76 08 b2
[I] Key refresh phase 2
[I]   Phase 0 succeed: 54 16 d3 83 64 c3 21 54 84 87 6a e4 9b 76 08 b2
[I] Key refresh succeed
```

3.9 Exporting the keys

To export the network, app and device keys, run the NCP host provisioner with the `-x or --key-export` argument followed by the desired filename, or input **x** in the UI mode (filename defaults to `BtMeshKeys.json`).

```
% ./exe/btmesh_host_provisioner -u /dev/tty.usbmodem0004402253201 -x ./keys.json
[I] Export keys to: ./keys.json
[I] Empty NCP-host initialised.
[I] Resetting NCP...
[I] Bluetooth stack booted: v5.1.2-b215
[I] Provisioner init
[I] Network initialized
[I] Networks: 1
[I] Address : 2001
[I] IV Index: 0
```

Simplicity Studio

One-click access to MCU and wireless tools, documentation, software, source code libraries & more. Available for Windows, Mac and Linux!

IoT Portfolio
www.silabs.com/IoT

SW/HW
www.silabs.com/simplicity

Quality
www.silabs.com/quality

Support & Community
www.silabs.com/community

SILICON LABS

Silicon Laboratories Inc.
400 West Cesar Chavez
Austin, TX 78701
USA

www.silabs.com