UG472: *Bluetooth*® Mesh Stack and *Bluetooth*® Mesh Configurator User's Guide for SDK v2.x and Higher

This user guide describes the components, stack and DCD (Device Composition Data) configuration options for the Bluetooth Mesh SDK.

KEY POINTS

- Introduction to Simplicity Studio 5 Bluetooth Mesh components.
- Modifying the Device Composition Data, including device information, elements, and models.
- Setting stack configuration options to optimize the RAM and persistent storage usage.

Table of Contents

1 Introduction

A new way of configuring the Bluetooth Mesh stack, namely through components and the Bluetooth Mesh Configurator tool, was introduced beginning with Bluetooth Mesh SDK 2.0 and Simplicity Studio 5.

1.1 Terminology

The following table gathers the Bluetooth mesh-specific terms in use in this document. Those terms are defined in the SIG Bluetooth Mesh Profile specification.

Table 1-1. Terminology

Term	Definition
Address	The identity of one or more elements in one or more nodes.
Configuration Client	A node that implements the Configuration Client model.
Destination	The address to which a message is sent.
Device	An entity that is capable of being provisioned onto a mesh network.
Element	An addressable entity within a device. A device is required to have at least one element.
Message	A sequence of octets that is sent from a source to a destination.
Network	A group of nodes sharing a common address space.
Node	A provisioned device.
Provision	The process of authenticating and providing basic information (including unicast addresses and a network key) to a device. A device must be provisioned to become a node. Once provisioned, a node can transmit or receive messages in a mesh network.
Provisioner	A node that is capable of adding a device to a mesh network.
Relay	A node that receives and then retransmits messages.
Source	The address from which a message is sent.
State	A value representing a condition of an element that is exposed by an element of a node.
Subnet	A group of nodes that can communicate with each other.

2 Simplicity Studio 5 and Bluetooth Mesh

A number of new features and architecture changes were introduced beginning with Bluetooth Mesh SDK 2.0 and Simplicity Studio 5. The features supported are backward compatible with applications built with the Bluetooth Mesh SDK v1.x, although the API in use is different. Projects generated with the Bluetooth Mesh SDK version 2.0 and higher can be configured via the three following input parameter tools:

- Project file, using the slcp extension (silicon labs component project) (For the Mesh components)
- Bluetooth mesh configurator (for DCD configuration)
- Bluetooth GATT configurator (for Mesh services)

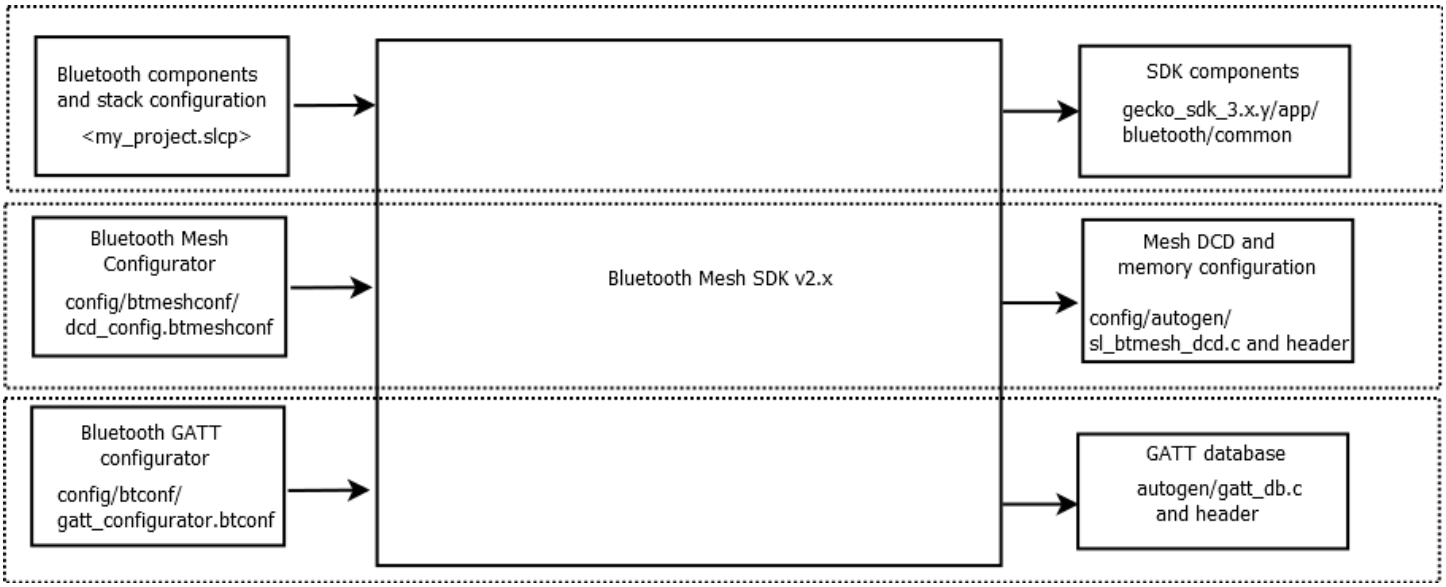

Figure 2-1. Bluetooth Configuration Overview

Note: Unlike versions 1.x and lower, the current version of the Bluetooth Mesh SDK does not have a Generate control. Project files are generated and updated as you make changes and save the updates in the Component Configurator.

2.1 Bluetooth Mesh Components

Upon creation of a Bluetooth Mesh project in Simplicity Studio 5, three tabs open automatically:

- The GATT Configurator (gatt_configuration.btconf)
- The slcp file or Project Configurator (<projectname>.slcp
- The Bluetooth Mesh Configurator (dcd_config.btmeshconf). If the example has documentation, the project opens on a readme tab.

Figure 2-2. Bluetooth Mesh Configuration Tools

The **GATT Configurator** is the same for both Bluetooth and Bluetooth mesh projects. *UG438: GATT Configurator User's Guide for Bluetooth SDK v3.x* describes how to configure the GATT database.

The **Project Configurator** and associated component editor allows you to install/uninstall and configure components.

Bluetooth Mesh components are organized in five categories:

- Features
- Models
- Service
- Stack Classes
- Bluetooth Mesh Stack

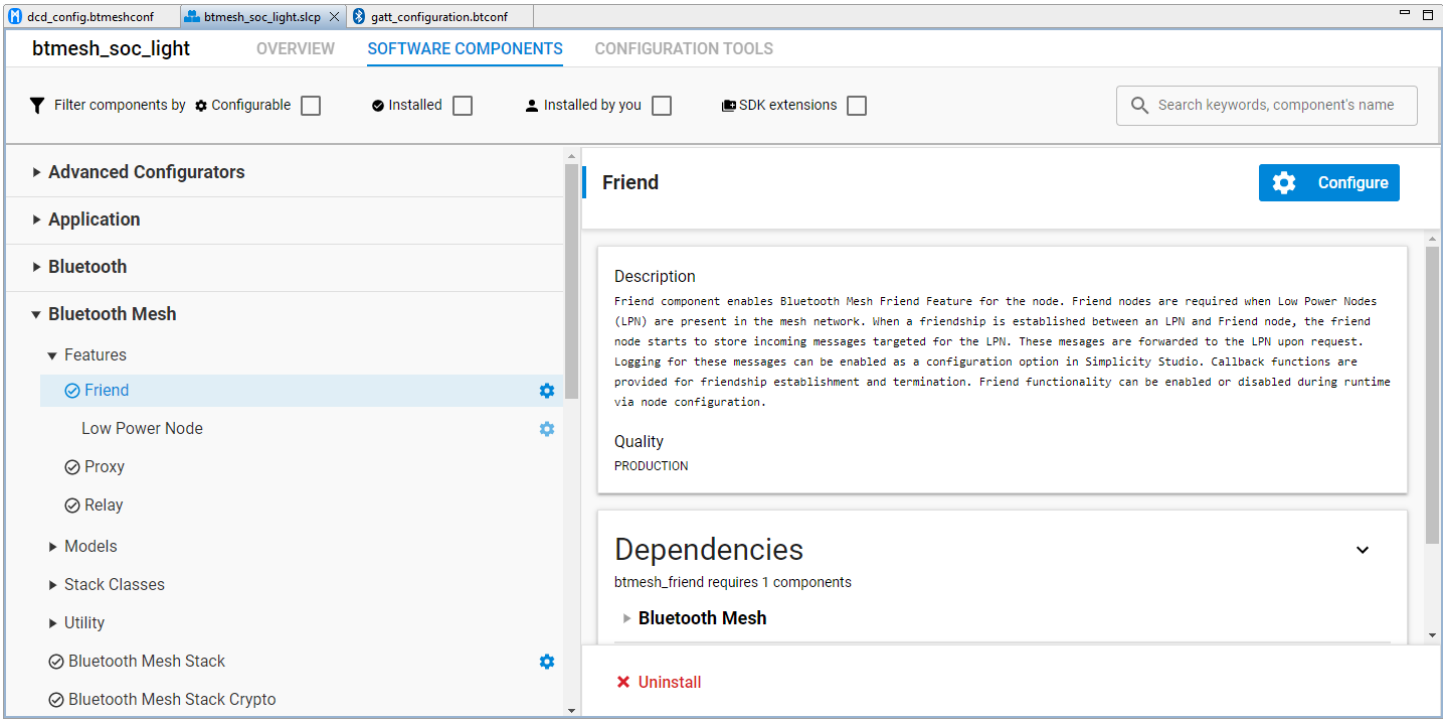

Figure 2-3. Project Configurator, Software Components Tab

The components in the Features group correspond to the Features field in the Device Composition Data Page 0. You can enable or disable a specific feature by installing or uninstalling the corresponding component.

The models components allow you to enable or disable a model API. If a model is needed in your project, make sure to install the corresponding component and configure it as needed.

The Service components provides a set of functionalities of possible use in project development, for example, factory reset and event logging.

Bluetooth Mesh APIs are organized into several categories by functionalities. You can choose what classes are required by the use case and initialize them. Uninitialized classes will not be built into the application, thereby reducing the flash and RAM usage.

The Bluetooth Mesh Stack component includes options that allow you to optimize memory usage. Several memory configuration options for a Mesh project are available. Some only affect the RAM consumption, and others affect both RAM and the persistent storage. Because the space available in RAM and persistent storage is limited, set the configuration option values in a suitable manner.

The **Bluetooth Mesh Configurator** provides access to Device Composition Data (DCD). This contains information about a Bluetooth mesh node, the elements it includes, and the supported models. DCD exposes the node information to a configuration client so that it knows the potential functionalities the node supports and, based on that, can configure the node.

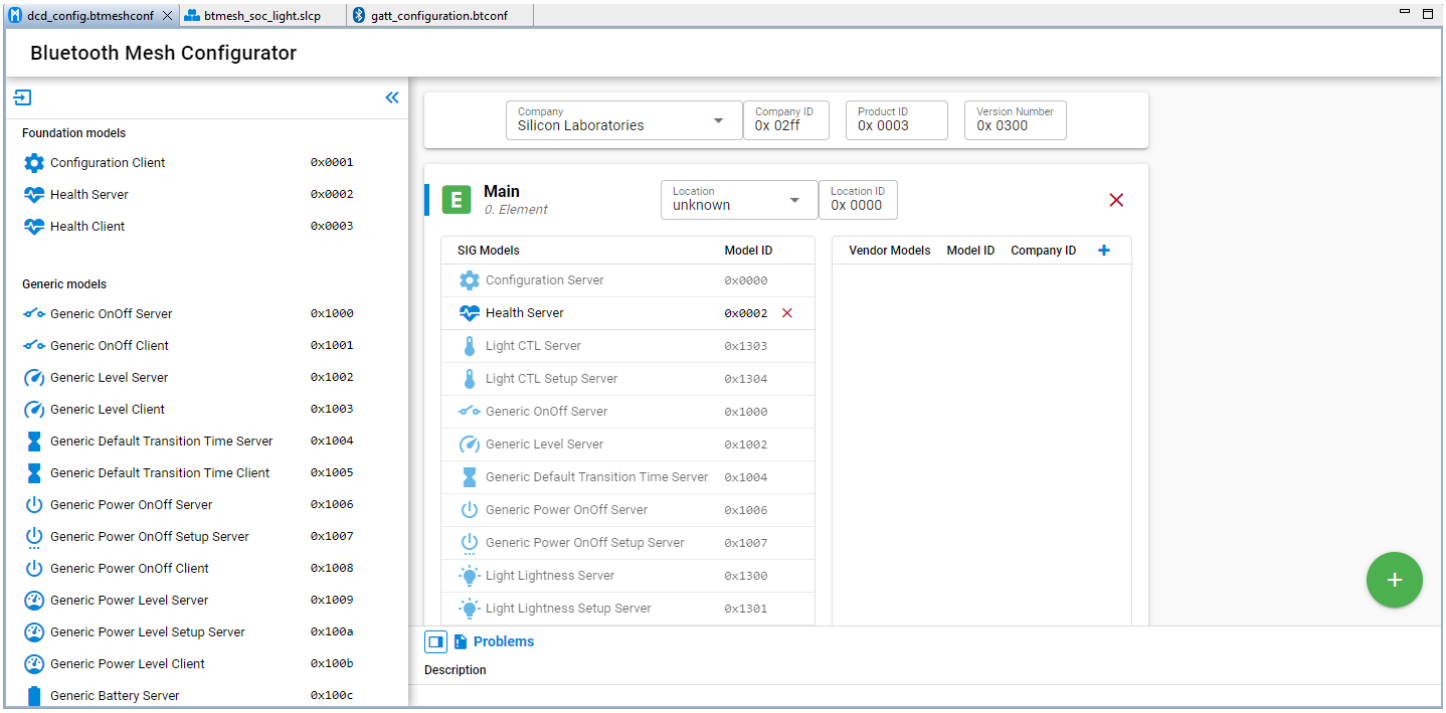

Figure 2-4. Bluetooth Mesh Configurator

2.2 Bluetooth Mesh Configurator

To access Device Composition Data, open the Bluetooth Mesh Configurator on the dcd_config.btmeshconf tab. The Device Composition data is presented in three areas: device information, elements, and models.

2.2.1 Device Information

The device information card contains four fields, shown in the following figure.

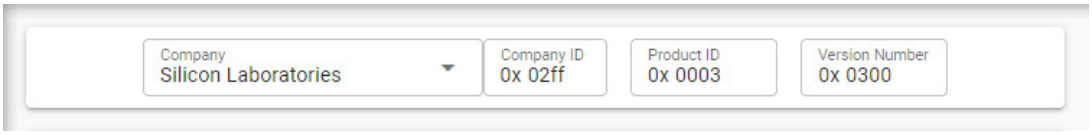

Figure 2-5. Device Information Card

The Company field is linked to the Company ID field, changing one will automatically change the other as a result. The meaning of each field is shown in the following table.

Table 2-1. Device Information Fields

Field Name	Notes
Company	The company name in the list containing all the registered companies in Bluetooth SIG
Company ID	16-bit company identifier assigned by the Bluetooth SIG
Product ID	16-bit vendor-assigned product identifier, vendor-specific
Version Number	16-bit vendor-assigned product version identifier, vendor-specific

A list of companies and their unique identifier can be found on the Bluetooth SIG site.

2.2.2 Elements

An element is an addressable entity within a node. Each node can have one or more elements, the first called the primary element and the others called secondary elements. Each element is assigned a unicast address during provisioning, so that it can be used to identify which node is transmitting or receiving a message. The primary element is addressed using the first unicast address assigned to the node, and the secondary elements are addressed using the subsequent addresses. Both primary and secondary elements have a dedicated card, such as that shown in the following figure, through which they can be configured. Click the green plus symbol to add an element or select an element and click the red X symbol to remove it.

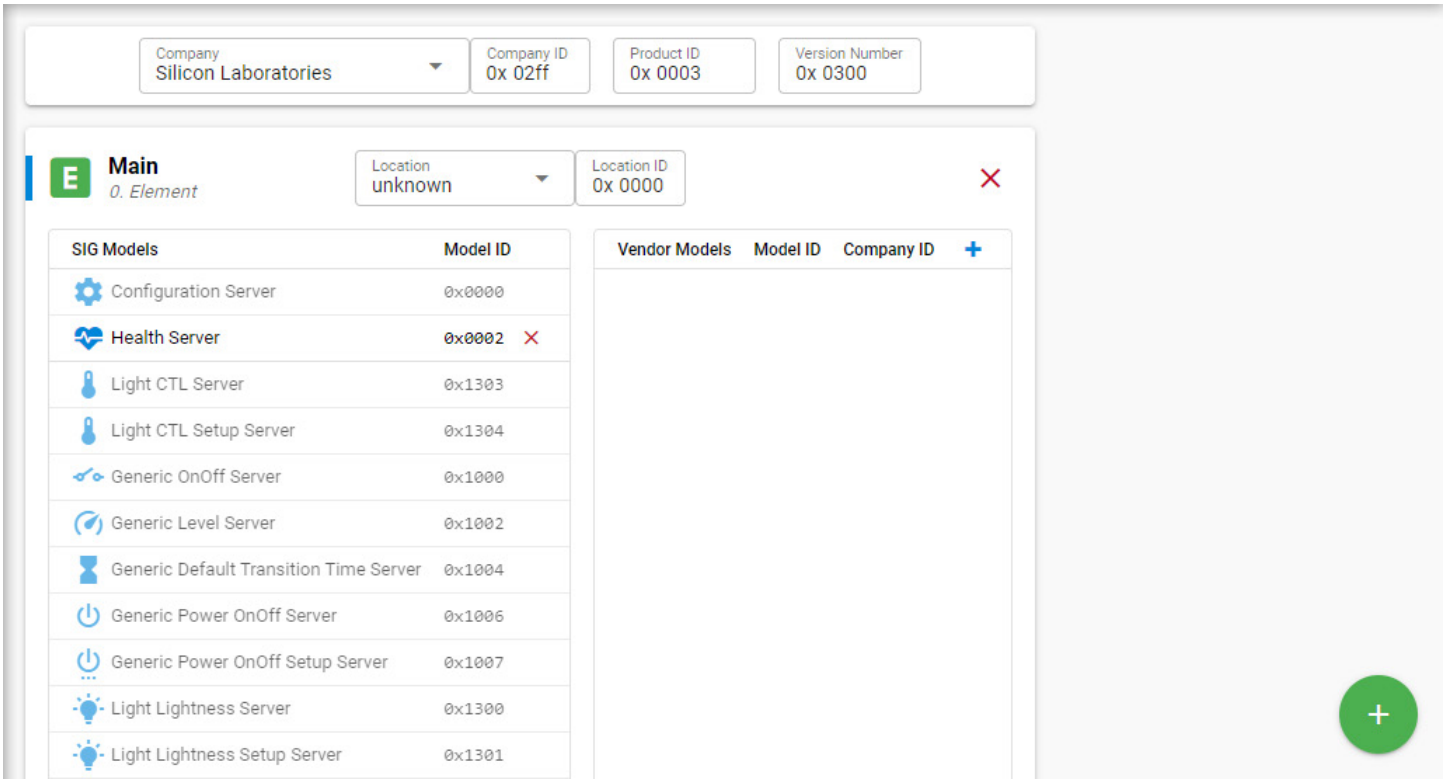

Figure 2-6. Primary Element Card

2.2.3 Models

A model defines the basic functionality of a node. A node may include multiple models. A model defines the required states, the messages that act upon those states, and any associated behaviors.

Models may be defined and adopted by the Bluetooth SIG and may also be defined by vendors. Models defined by the Bluetooth SIG are known as SIG-adopted models, and models defined by vendors are known as vendor models. SIG-adopted models are identified by a 16-bit model identifier and vendor models are identified by a 16-bit vendor identifier and a 16-bit model identifier.

The Bluetooth Mesh Configurator supports configuring both SIG-adopted models and vendor models through separate editors.

2.2.4 SIG-Adopted Model Editor

Add SIG Models via Components

If you are using the provided model components that automatically bring in the source/header files, libraries, and configurations to the project, and also contribute the model to the DCD, you cannot edit or delete the model from the DCD manually. The model is greyed out,

as shown in the following figure. In this case, all the model implementations will be generated to the project. You can modify the callbacks to adjust the application to your use case.

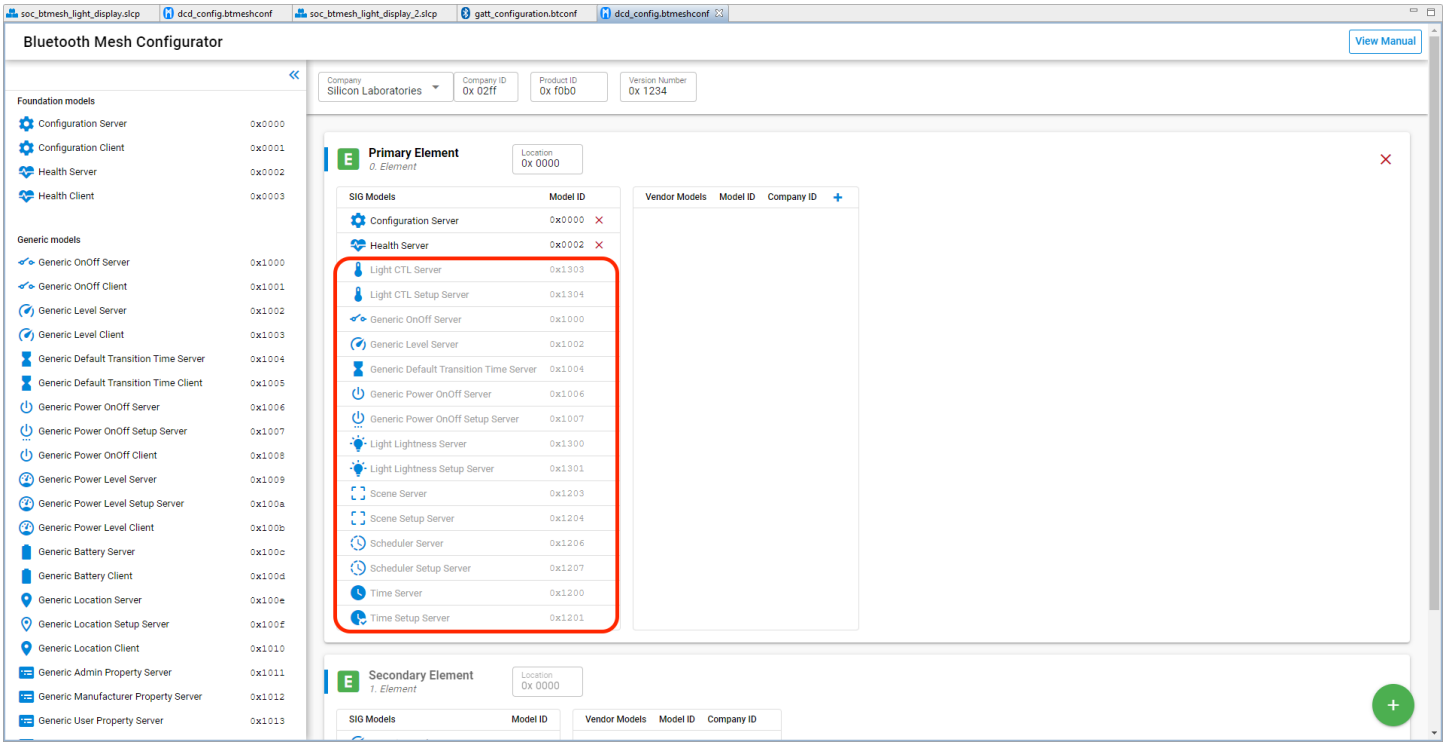

Figure 2-7. SIG-Adopted Model Editor with Components

Edit SIG Models Manually

The DCD editor displays the information list of available models in SSv5 on the left side. Because models are added and configured as components, they are not editable in the Mesh configurator. To build the DCD from a clean slate, for example when adding specific models to an element, uninstall all model components manually and then edit the DCD.

To delete a model, select it and click the red X symbol. To add a SIG-adopted model, drag and drop the model from the left model pool to the SIG Models table in the correct element. A list of all the SIG-adopted models is displayed, and you can choose the one that is needed. Note that, although all the SIG-adopted models are listed, not all of them are currently supported by the Bluetooth Mesh SDK. For the information on the supported models, see the SDK release notes.

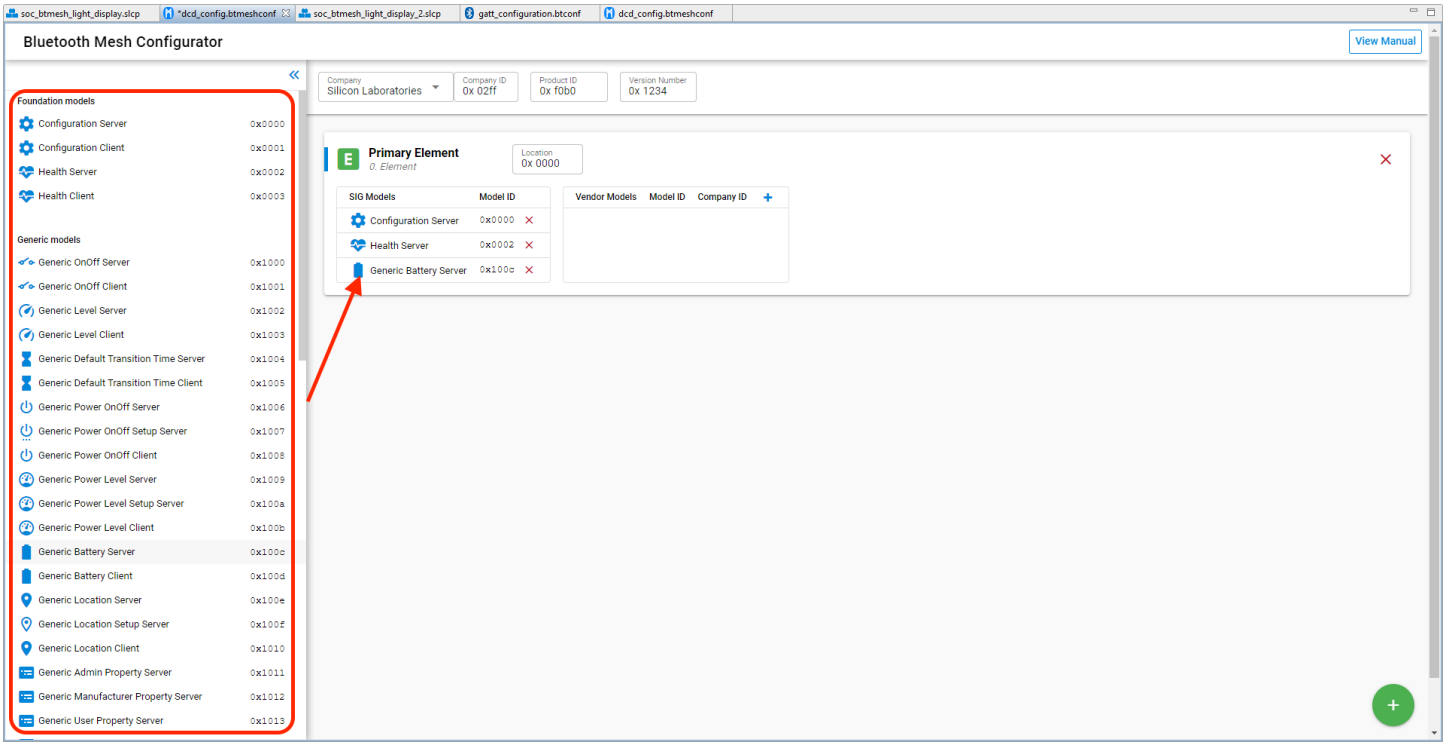

Figure 2-8. SIG-Adopted Model Editor

Due to the extension mechanism of models mentioned above, attention must be paid when adding models to your project:

1. When adding a model that is not a root model, in other words an extended model, all the models it extends from should also be added. The Bluetooth Mesh Model Specification has the detailed definition of the models' relationships. For example, to add a Light Lightness Server model, the Generic Power On/Off Server model and the Generic Level Server model must also be present in the settings, because the Light Lightness Server model is extended from them.
2. One element can only have one instance of a model. For example, if both model A and Model B are extended from Model C, you cannot add them both to a single element because it would require two Model C instances. The appropriate way to achieve this is to have two elements, and put model A and model C in one element and model B and model C in the other element. For example, the light example in the Bluetooth mesh SDK has two elements in order to have two Generic Level Server model instances.

In addition to the above points, follow the points below when editing the model setting of a node.

1. The configuration server model must be supported by a primary element and must not be supported by any secondary elements.
2. To develop a provisioner, add at least the configuration client model in your project, and it should be in the primary element.
3. The health server model must be supported by a primary element and may be supported by any secondary elements.
4. If the health client model is supported, it must be supported by a primary element and may be supported by any secondary elements.

2.2.5 Vendor Model Editor

The Vendor models give you more flexibility when developing products not covered by the SIG-adopted models. Vendors can define their own specification in these models, including states, messages, and the associated behaviors. The vendor model editor is shown in the following figure. The ID field contains the 32-bit vendor identifier and model identifier. The two least significant bytes of the ID are the vendor ID and the two most significant bytes are the model ID. In the following figure, 0x02FF is the vendor ID for Silicon Labs, and 0x0021 and 0x0022 is the model ID.

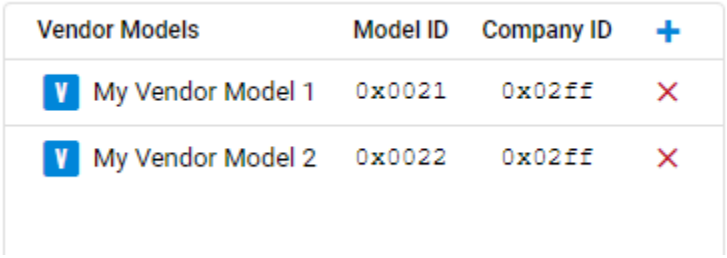

Figure 2-9. Vendor Model Editor

Click the plus symbol to add a vendor model, or select a model and click the red X symbol to remove it.

2.3 Bluetooth Mesh Stack

The Bluetooth mesh SDK provides several SRAM and internal flash consumption optimization options. Memory should be configured to allocate the appropriate amount of resource needed, so the space left for application usage is optimized. The stack memory configuration can be tuned by configuring the Bluetooth Mesh Stack component.

The following figure shows the available configuration options:

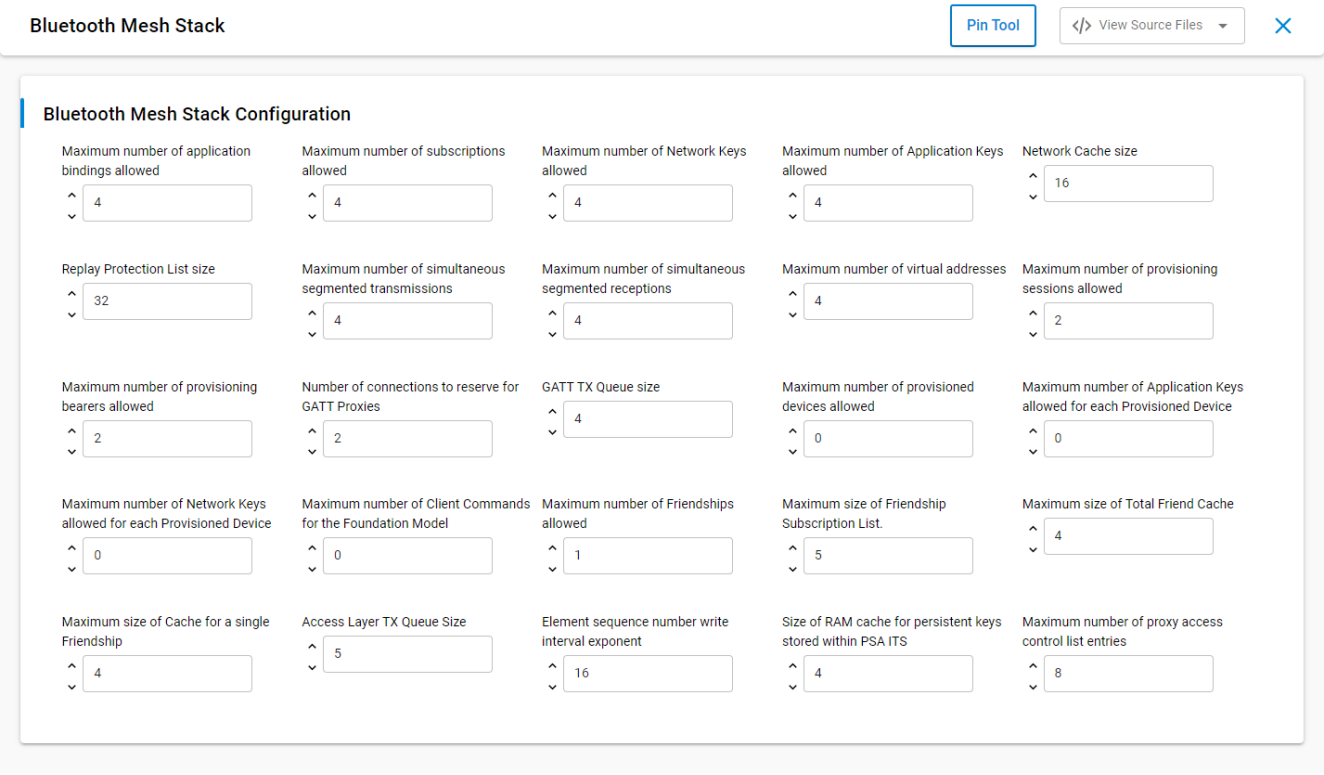

Figure 2-10. Bluetooth Mesh Stack Configuration

All configuration options affect RAM consumption. The stack allocates various structures at startup based on the values entered and uses the allocated memory during operation.

Some configuration options also affect consumption of persistent storage in internal flash. The stack allocates space in persistent storage based on the configuration option values at start up. The persistent storage implementation used by the Bluetooth Mesh stack is either NVM3 (recommended, default on series 2) or PS Store. For more details, see AN1135: Using Third Generation Non-Volatile Memory (NVM3) Data Storage.

The following table summarizes the configuration options:

Table 2-2. Summary of Configuration Options

Configuration Option	Description	Stored Persistently	Notes
Maximum number of Network Keys allowed	The maximum number of network keys that can be stored (see 2.2.1 Network and subnets and 2.3.9 Security of Mesh Profile 1.0.1).	Yes	No larger than 7.
Maximum number of Application Keys allowed	The maximum number of application keys that can be stored (see 2.2.1 Network and subnets and 2.3.9 Security of Mesh Profile 1.0.1).	Yes	No larger than 8.
Maximum number of application bindings allowed	The maximum number of application keys that can be bound to a model.	Yes	No larger than the smaller value of 'Maximum number of Application Keys allowed' and 255.
Maximum number of subscriptions allowed	The maximum number of addresses that the device can subscribe to (see 3.7.6.2 Subscribe of Mesh Profile 1.0.1).	Yes	No larger than 255.
Maximum number of provisioned devices allowed	The maximum number of devices that can be provisioned by this device.	Yes	Only applicable if the device is in provisioner role, no larger than 512. Set to 0 for node role.
Replay Protection List size	The replay protection list size (see 3.8.8 Message replay protection of Mesh Profile 1.0.1).	Yes	Set to equal or greater than the expected number of elements the device will communicate with. Otherwise, the node cannot receive a message from any new node if the list is already full. Must be no larger than 4096 and divisible by 16.
Maximum number of virtual addresses	The maximum number of virtual addresses the models on the device can publish or subscribe to (see 2.3.5 Addresses and 3.4.2.3 Virtual address of Mesh Profile 1.0.1).	Yes	Set to 0 if virtual address not used.
Maximum number of Network Keys allowed for each Provisioned Device	The maximum number of network keys on the peers provisioned by this device.	Yes	Only applicable if the device is in provisioner role.
Maximum number of Application Keys allowed for each Provisioned Device	The maximum number of application keys on the peers provisioned by this device	Yes	Only applicable if the device is in provisioner role.
Maximum number of simultaneous segmented receptions	The maximum number of segmented messages that can be received in parallel (see 3.5.3 Segmentation and reassembly of Mesh Profile 1.0.1).	No	Set to a low number if little segmentation is used.
Maximum number of simultaneous segmented transmissions	The maximum number of segmented messages that can be sent in parallel (see 3.5.3 Segmentation and reassembly of Mesh Profile 1.0.1).	No	Set to a low number if little reassembly is used.
Maximum number of provisioning sessions allowed	The maximum number of simultaneous provisioning sessions the device supports.	No	Set to 1 if the device is in node role. For the provisioner role, set to greater than 1 if provisioning multiple devices simultaneously.

Configuration Option	Description	Stored Persistently	Notes
Maximum number of Client Commands for the Foundation Model	The maximum number of commands that Configuration and Health client can send in parallel (see 4 Foundation models of Mesh Profile 1.0.1).	No	Only applicable if the device is in provisioner role.
Network Cache size	The network message cache size (see 3.4.6.5 Network Message Cache of Mesh Profile 1.0.1).	No	Network density-dependent.
Number of connections to reserve for GATT Proxies	The maximum number of GATT connections for PB-GATT and GATT bearers.	No	Can be 0 if PB-GATT and GATT bearers are not supported.
Maximum number of provisioning bearers allowed	Number of provisioning bearers (see 5.2 Provisioning bearer layer of Mesh Profile 1.0.1).	No	Number of supported provisioning bearers, PB-ADV, PB-GATT or both. Not greater than 2.
Maximum number of Friendships allowed	The maximum number of friendships that can be established (see 2.3.10 Friendship of Mesh Profile 1.0.1).	No	Only applicable for friend node.
Maximum size of Total Friend Cache	The maximum number of messages a friend node can cache. (see 3.5.5 Friend Queue of Mesh Profile 1.0.1).	No	Only applicable for friend node.
Maximum size of Cache for a single Friendship	The maximum number of messages a friend node can cache for a single low-power node (see 3.5.5 Friend Queue of Mesh Profile 1.0.1).	No	Only applicable for friend node.
Maximum size of Friendship Subscription List	The maximum number of addresses that can be stored in the Friend Subscription List.	No	Only applicable for friend node.
GATT TX Queue size	Queue size for messages over GATT bearer.	No	Connection interval-dependent.
Access Layer TX Queue Size	The maximum number of messages that can be queued in the Access layer (see 3.7.4.1 Transmitting an access message of Mesh Profile 1.0.1).	No	
Element sequence number write interval exponent	The latest Network PDU sequence numbers are stored into flash from time to time as defined by this setting for reset or power off situations.		From 0 to 23, default 16.
Size of RAM cache for persistent keys stored within PSA ITS	PSA ITS (internal trusted storage) Mesh encryption keys RAM cache to increase runtime performance.		From 0 to 544, default 16.
Maximum number of proxy access control list entries	Define the number of proxy access control list entries.		Default 8

To download Mesh Profile 1.0.1, go to https://www.bluetooth.org/docman/handlers/downloaddoc.ashx?doc_id=457092.

2.3.1 Maximum number of Network Keys allowed

The value determines the maximum number of network keys that can be stored in the device. For the Bluetooth Mesh SDK 1.7.x, the maximum is 7, which means a device can support no more than 7 subnets. A node should stay in the network(s) it was in after a power cycle, so the network keys and the related information should be stored persistently. Because of the key refresh procedure requirements, each network key will hold 2 values – the current network key and the old network key.

2.3.2 Maximum number of Application Keys allowed

The value determines the maximum number of application keys that can be stored in the device. For the Bluetooth Mesh SDK 1.7.x, the maximum number is 8, which means a device can support no more than 8 application keys no matter which network keys they are bound to. The application keys and the related information should be stored persistently. Because of key refresh procedure requirements, each application key will hold 2 values – the current application key and the old application key.

The maximum application key number should be set close to the expected number of application keys that will be used in a network.

2.3.3 Maximum number of application bindings allowed

If a message is successfully decrypted by the upper transport layer with an application key, the decrypted message and the application key information will be delivered to the access layer. The access layer will check if the message is used by the model on the node, and then check if the application key is bound to the model. This value decides the maximum number of application keys that can be bound to a single model. The binding information will be stored persistently. Because the bindings are model-specific, the total amount of flash usage is multiplied by the number of models on the node.

The number of bindings should not be set larger than the number of application keys that can be stored on the device. It can be set to a smaller number if it is expected that each model will be bound to only one or a few keys.

2.3.4 Maximum number of subscriptions allowed

Each model can have a separate or a shared subscription list, if it supports subscription. This value determines the subscription list size, in other words, how many addresses can be subscribed by a model. All the subscription lists will be stored persistently, because the extended models share the same subscription list with their root model. The real amount of space depends on what models are on the device.

The number of subscriptions should be set to the maximum expected number of subscriptions to be made to each model, or slightly larger.

2.3.5 Maximum number of provisioned devices allowed

This setting is applicable only when the device is in the provisioner role. The value determines the maximum number of devices the provisioner can provision to the network. The best number for this setting should be the maximum expected network size. For Bluetooth Mesh SDK 1.7.x, the maximum value is 512, which means the maximum network size supported by the stack is 512 nodes.

Because a node cannot provision any devices into the network and doesn’t have the device database, set to 0 for devices in node role.

2.3.6 Replay Protection List size

A message sent by a legitimate originating element can be passively received by an attacker and then replayed later without modification. This is called a replay attack. Because the originating element has encrypted and authenticated the message using the correct keys, the receiver cannot determine whether it is under a replay attack solely by performing the message integrity checks.

To increase protection against replay attacks, each element increases the sequence number for each new message that it sends, and the receivers keep track of the largest sequence number they have received from each originating element. This bookkeeping is called the replay protection list. If a valid message has been received from an originating element with a specific sequence number, any future messages from the same originating element containing sequence numbers less than or equal to the last valid sequence number are very likely replayed messages and should be discarded. Therefore, messages are delivered to the access layer in sequence number order.

Due to security concerns, entries in the replay protection list cannot be reused, which means there is no way to delete or clear an entry if it has already been used. No Least Recently Used algorithm is used in this list because it brings potential security risks. It is explicitly specified in the Mesh Profile 1.0.1 - If a node does not have enough resources to perform replay protection for a given source address, then the node shall discard the message immediately upon reception.

Furthermore, because nodes could be removed from the network and new devices will be added to the network, the replay protection list should be set to the maximum number of elements the device will communicate with, which could be larger than the maximum network size in this case. For example, assume the network size is 5, contains devices 1-5, and they have already communicated with each other. The replay protection list on device 1 should contains device 2-5, which occupies 4 entries. Then, if devices 2-5 are removed from the network and devices 6-9 are added to the network, device 1 will need 4 more entries in the replay protection list to be able to communicate with devices 6-9. In this scenario, device 1 needs 8 replay protection entries, which is larger than the network size of 5.

The replay protection list is stored persistently. The number of replay protection list entries should be set to the number of peers a node is expected to communicate with, rounded up to the nearest number divisible by 16.

2.3.7 Maximum number of virtual addresses

This setting determines the maximum number of virtual addresses the models on the device can publish or subscribe to. A virtual address is a multicast address and can represent multiple elements on one or more nodes. Each virtual address logically represents a Label UUID, which is a 128-bit value that does not have to be managed centrally. Each message sent to a Label UUID includes a message integrity check value containing the full Label UUID that is used to authenticate the message. To reduce the overhead of checking every known Label UUID, a hash of the Label UUID is used. Although the virtual address is 2 bytes, the 128-bit label UUID value should be stored persistently because the full data needs to be used for decrypting the messages sent to virtual addresses.

The number of virtual addresses should be set to as small a number as possible, or zero if it is expected that virtual addresses are not used. The current Mesh Model specification does not require the use of virtual addresses, so at the moment they are used only in vendor-specific contexts.

2.3.8 Maximum number of Network Keys allowed for each provisioned device

This setting is only used during the key refresh procedure. The provisioner should persistently store the states of network keys of the nodes participating in the key refresh procedure. This value determines the maximum number of network keys that can be included in the key refresh procedure, in other words, how many network keys can be refreshed one time.

This only applies to a provisioner. Set to 1 if you only want to refresh one network key at a time, or a higher value if the use case needs to refresh more than one network key at once. Set to 0 for devices in node role.

2.3.9 Maximum number of Application Keys allowed for each provisioned device

This setting is only used during the key refresh procedure. The provisioner should persistently store the states of application keys of the nodes participating in the key refresh procedure. This value determines the maximum number of application keys that can be included in the key refresh procedure, in other words, how many application keys can be refreshed one time.

This only applies to a provisioner. Set to 1 if you only want to refresh one application key at a time, or a higher value if the use case needs to refresh more than one application key at once. Set to 0 for devices in node role.

2.3.10 Maximum number segments allowed for received packets

Due to the packet size limitation in Bluetooth Mesh, a message may be sent unsegmented or segmented, depending on the message payload size. For the transport layer to receive the segmented message, it has to cache the received segments before all the segments are received successfully. This setting determines the maximum segmented messages that can be received concurrently. Note, this does not define how many segmented packets in a message, but how many segmented messages no matter how many packets the message is segmented into. For example, if the setting is 3, the node is able to receive segmented message A, B, and C simultaneously, no matter how messages A, B, and C are segmented into A1, B1, C1 … An, Bn, Cn. The maximum number that a message can be segmented into is defined in the Mesh Profile specification.

A device with standard models rarely receives segmented messages (encryption key deployment being one example) so a low number can be used if none of the vendor models need segmentation.

2.3.11 Maximum number segments allowed for transmitted packets

Due to the packet size limitation in Bluetooth Mesh, a message may be sent unsegmented or segmented, depending on the message payload size. For the transport layer to send the segmented message, it has to cache the whole message before all the segments are sent and acknowledged successfully. This setting determines the maximum segmented messages that can be sent concurrently. Note, this does not define how many segmented packets are in a message, but how many segmented messages no matter how many packets the message is segmented into. For example, if the setting is 3, the node is able to send segmented message A, B, and C simultaneously, no matter how messages A, B, and C are segmented into A1, B1, C1 … An, Bn, Cn. The maximum number that a message can be segmented into is defined in the Mesh Profile specification.

A device with standard models rarely sends segmented messages (encryption key deployment being one example) so a low number can be used if none of the vendor models need segmentation.

2.3.12 Maximum number of provisioning sessions allowed

Provisioning is session-based. A provisioner does not necessarily need to provision the devices serially, but instead can provision the devices in parallel. This setting determines the maximum number of provisioning sessions that can happen concurrently. For example, in an ideal scenario, if the value is 1 and each provisioning takes 3 seconds, then provisioning 100 devices takes 300 seconds. If you set this value to 5, then it takes 60 seconds in total to provision all 100 nodes. Although in practice the time will be affected by packet collision, it should still be much less than 300 seconds.

This does not need to be over 1 for devices in node role because a device cannot be provisioned by multiple provisioners at the same time. It may be over 1 for a provisioner that provisions devices concurrently and it significantly reduces the time for provisioning a large network.

2.3.13 Maximum number of client Ccmmands for the Foundation Model

After provisioning a device into a network, the first step is probably to configure the node because an unconfigured node is not functional. The configuration starts by the configuration client model sending a command to the configuration server on the node, followed by a reverse status message if applicable. The current Mesh Profile 1.0.1 specification only defines 2 foundation client models – configuration client model and heath client model. This setting determines how many commands can be sent by the foundation client models concurrently before the status message is received. For example, in an ideal scenario, if the value is 1 and the configuration to add an application key to a node takes 3 seconds, then adding the application key to 100 devices takes 300 seconds. If you set this value to 5, then it takes 60 seconds in total to add the application key to all 100 nodes. Although in practice the time will be affected by packet collision, it should still be much less than 300 seconds.

This is not applicable and should be set to 0 for devices that do not have configuration client or health client models. It may be over 1 for a provisioner that configures devices concurrently and it significantly reduces the time for configuring a large network.

2.3.14 Network cache size

The network message cache is used to reduce unnecessary security checks and excessive relaying. It is a list of all the information about recently seen network packets. When a network PDU is received and already in the network message cache, for example because of network retransmission, it will be discarded immediately without processing. If a received network PDU is not in the network message cache, its information should be added to the network message cache and be further processed.

Note, this network message cache is different from the replay protection list. It is not for security. When the Network Message Cache is full and an incoming new Network PDU needs to be cached, an incoming new Network PDU should replace the oldest Network PDU that is already in the Network Message Cache.

The suitable value is dependent on expected network density, node configuration, and traffic frequency. For example, if network layer repetition is configured on, with an interval that allows multiple messages to be injected to the network by other nodes during the interval, the cache should be large enough to handle this and not flush the previous Tx before the repetition.

2.3.15 Number of connections to reserve for GATT proxies

This setting determines the maximum number of GATT connections the device can establish concurrently. For devices in an unprovisioned state, the GATT connections can be used for establishing the PB-GATT bearer so that provisioning can be done over GATT. For devices in a provisioned state and supporting Proxy, the GATT connections can be used for establishing the proxy connections so that all the communication can go over GATT bearers.

If devices do not support the Proxy feature or provisioning over a GATT connection, it can be set to 0. The number of proxy filters is limited to 16 per connection.

2.3.16 Maximum provisioning bearers

Two provisioning bearers are defined in the Mesh Profile 1.0.1 – PB-ADV and PB-GATT. This setting should be consistent with the number of provisioning bearers supported by the device. An unprovisioned device may support PB-ADV and may support PB-GATT. Supporting both PB-ADV and PB-GATT is strongly recommended. A Provisioner must support at least one of PB-ADV or PB-GATT. Supporting PB-ADV is strongly recommended.

The value should not be set to larger than 2 as only 2 available provisioning bearers are defined in Mesh Profile 1.0.1. It is also not recommended to set it to 0, as provisioning is mandatory for devices to be added to a network.

2.3.17 Maximum number of Friendships allowed

In principle, all the nodes in the Bluetooth Mesh network should listen for incoming packets at the highest possible duty cycle to avoid losing packets. But a battery-powered device must sleep to save power, so it must be associated with an always-on device that stores and relays messages on its behalf. The relationship between the always-on node and the battery-powered nodes is friendship.

This setting is only applicable for nodes that support the friend feature. It determines the maximum number of friendships it can establish, in other words, the maximum number of low-power nodes it can establish the friendship with concurrently. Set to 0 if the node does not support the friend feature.

2.3.18 Maximum size of Total Friend Cache

As mentioned in Section 2.3.17 Maximum number of Friendships allowed, a low-power node needs to establish a friendship with a neighboring friend node. The friend node will cache the message targeted to the low-power node and the low-power node can periodically poll the friend node for messages. All the cached messages are stored in the Friend Queue. This setting determines the Friend Queue size, that is, how many messages in total it can store for all the low-power nodes it established friendship with. It is only applicable for nodes which support the friend feature. Set to 0 if the node does not support friend feature.

2.3.19 Maximum size of cache for a single friendship

The ***Maximum size of Total Friend Cache*** setting (section 2.3.17 Maximum number of Friendships allowed) determines the maximum number of messages that can be stored in the Friend Queue in total. This setting determines the maximum number of messages that can be stored in the Friend Queue for a single Friendship. Because of the difference among use cases, the requirement for the number of messages to be stored could vary from one low-power node to another. This setting and the ***Maximum size of Total Friend Cache*** setting make the allocation of the Friend Queue dynamic when establishing friendships, and the use of the Friend Queue more efficient. This setting is only applicable for nodes that support the friend feature and must be set equal to or less than the ***Maximum size of Total Friend Cache*** setting. Set to 0 if the node does not support the friend feature.

2.3.20 Maximum size of Friendship Subscription List

As mentioned in section 2.3.17 Maximum number of Friendships allowed, the friend node needs to cache the message targeted to the low-power node that it established the friendship with. The low-power node would like to receive two types of messages. One is the message with the destination address to be the unicast address(s) on the low-power node. The other is the addresses that the low-power node subscribes to. In order to cache the messages designated for the addresses that the low-power node subscribes to, the friend node needs to maintain a list of the low-power node's subscription addresses. A low-power node could update the subscription list by adding addresses to the list or removing addresses from the list. This setting determines the maximum addresses that can be stored in the subscription list on the friend node for a single low-power node.

2.3.21 GATT TX Queue size

This setting determines the number of PDUs that may be pending transmit on the GATT bearer. The value may be small for a node that is only configured over a GATT proxy bearer (default is 4). It may need to be larger for a node that acts as a GATT proxy between the network and a legacy device, and where the connection interval for the GATT connection is long.

2.3.22 Access Layer TX Queue Size

As defined in 3.7.4.1 Transmitting an access message of Mesh Profile 1.0.1, the message in response to a received message should be sent after a random delay. Those messages need to be queued in the stack waiting for the timing to be sent. This setting determines the maximum number of access layer messages that can be queued, in other words, how many replies can be queued for sending concurrently. For nodes that do not have server models, this value can be set to a low value as messages will only be cached during the configuration phase. For server model nodes, this value should be set according to the requirements of the actual use case.

2.3.23 Element sequence number write interval exponent

Each network PDU originating from a device must be sent with an increasing sequence number. To maintain this when the device is reset or powered off, the latest sequence numbers are stored in flash with a frequency defined by this setting. The setting defines the sequence number writing interval as a power of two exponent. For example, a value of 10 would mean 1024 (2 to the 10th power). To avoid

excessive flash wear, the interval should be relatively high on a device that generates a lot of traffic, and it can be set relatively low on a device that generates little traffic. Range from 0 to 23, the default is 16.

2.3.24 Size of RAM cache for Persistent Keys stored within PSA ITS

When PSA internal trusted storage (ITS) is used to store the Mesh encryption keys, a RAM cache should be set up to increase runtime performance. The size of the cache should be set according to the expected use of application and device keys. For a node, it can be set to the number of application keys times two (to accommodate both key variants during a key refresh); for a Provisioner, it should be set to the number of application keys times two (to accommodate both key variants during a key refresh) plus a fraction of the expected number of device keys that will be stored. For devices that do not use PSA ITS, the setting is ignored. Range from 0 to 544, the default is 16.

2.3.25 Maximum number or proxy access control list entries

Define the number of proxy access control list entries. The default is 8.

2.4 Bluetooth GATT Configurator

The Bluetooth Mesh technology is primarily based on BLE advertisements that use a specific Mesh Message AD (Advertising Data) type. Some nodes, though, might not be able to advertise using the Mesh Message AD type, and instead require a GATT connection to send and receive network packets, provisioning data and so on.

The Bluetooth Mesh specification defines two GATT services dedicated to mesh networks for good operation in a connected context:

- The Mesh Provisioning service (0x1827)
- The Mesh Proxy service (0x1828)

A device may support the Mesh Provisioning Service or the Mesh Proxy Service or both. If both are supported, only one of these services should be exposed in the GATT database at a time. For more details on those services, refer to the Bluetooth Mesh profile specification (section 7).

In the current Silicon Labs' Bluetooth Mesh SDK, both services are present in the GATT database by default when creating a project.

The services have their capabilities disabled by default because they are Bluetooth Mesh-specific.

In certain cases, typically when provisioning through a gateway running a third party or open source stack like BlueZ or Zephyr OS, it is necessary to have the services advertised. This simply indicates to the provisioner that the Mesh services are supported by the node.

Note that, by default, when working with the current Bluetooth Mesh SDK, this is not necessary.

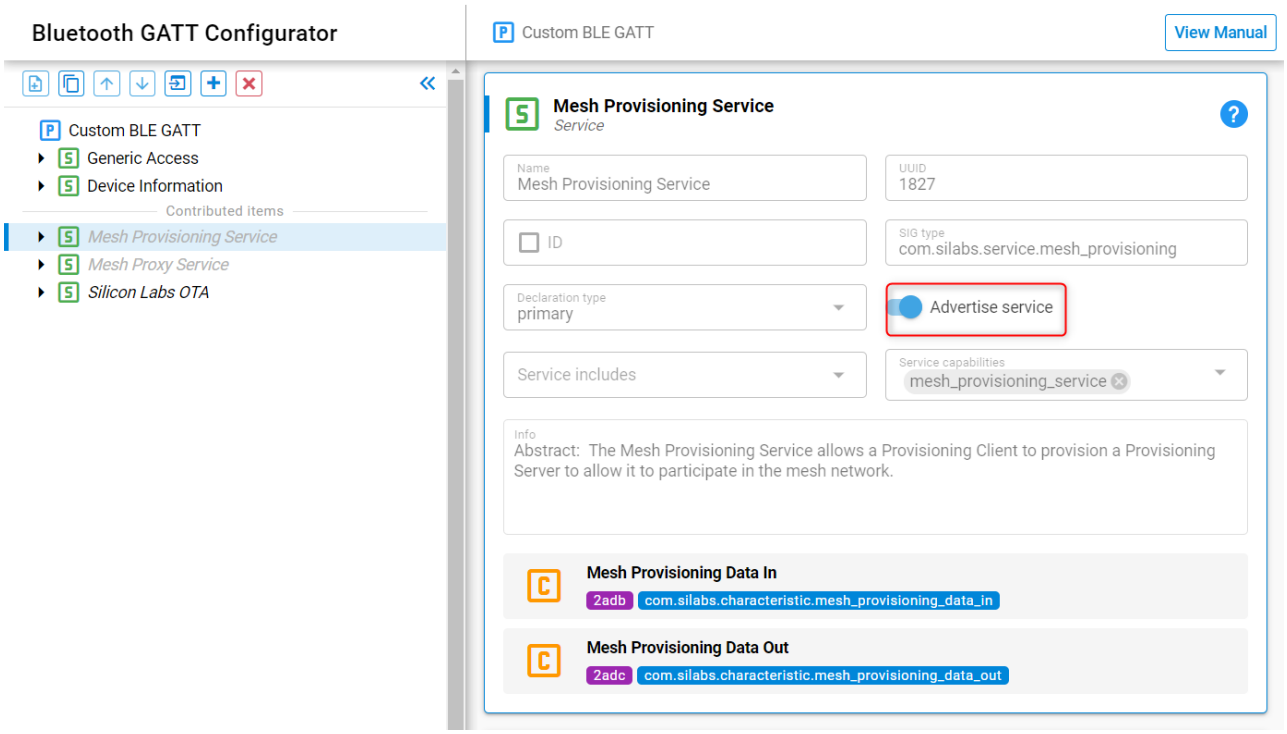

Figure 2-11. Bluetooth GATT Configurator

Enabling and disabling service advertisement can be done through the service xml definition file in your project:

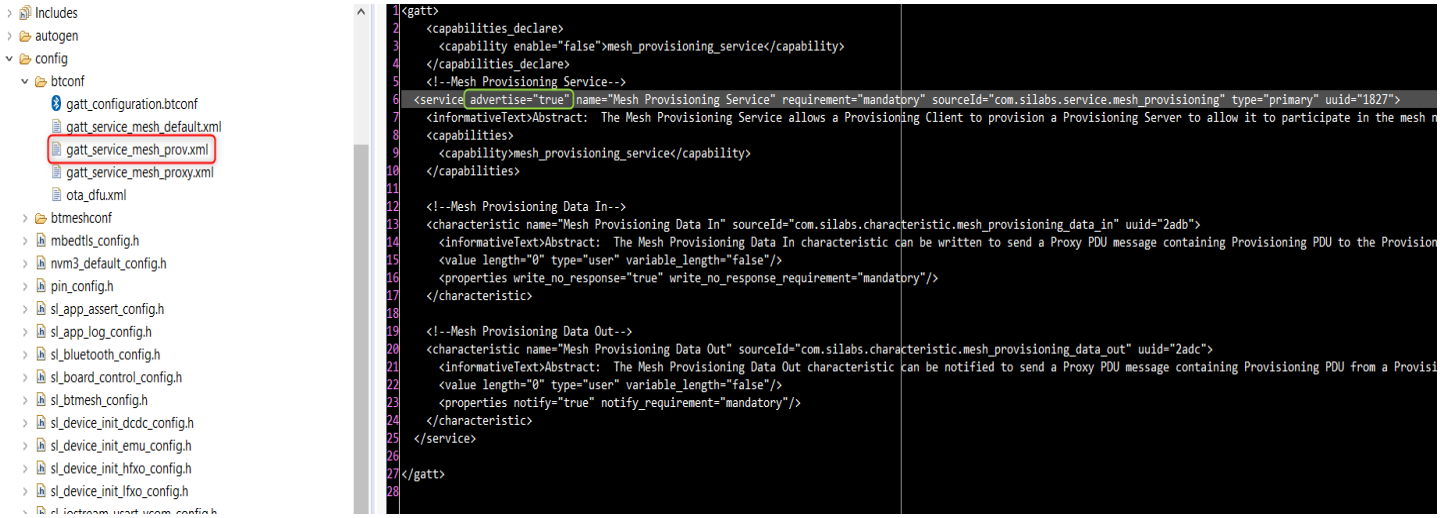

Figure 2-12. Editing the xml Service Parameter File

3 Bluetooth Mesh SDK and EFR32BG Series 1 and 2

The Bluetooth Mesh SDK v2.0 and higher supports both EFR32BG series 1 and series 2 products. Some products may not support all features.

3.1.1 Series 1 Support

Only EFR32xG13 and EFR32xG12 products support the Bluetooth Mesh stack.

3.1.2 Series 2 Support

Bluetooth Mesh is fully supported by EFR32xG21 and EFR32xG24 products. Due to memory considerations, EFR32xG22 chips have limited support for the Bluetooth Mesh stack (LPN and Proxy features available only).

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