UG366: *Bluetooth*® Mesh Node Configuration User's Guide for SDK v1.7.x

This users guide describes the DCD (Device Composition Data) Configurator and the Memory Configurator, both available in the Graphical User Interface of the Bluetooth mesh SDK.

KEY POINTS

- Modify the Device Composition Data, including device information, elements, and models.
- Set memory configuration options to optimize the RAM and persistent storage usage.

1 Introduction

The Silicon Labs Bluetooth mesh SDK supports configuring the Device Composition Data (DCD) and the memory use of a Bluetooth mesh project.

The Device Composition Data (DCD) 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.

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, developers should set the configuration option values in a suitable manner.

A graphic user interface is available in the .isc file, which is created automatically when you start a new project. The interface supports configuring the GATT database, the Device Composition Data, and the memory allocations for the project. UG365: Bluetooth GATT Configurator User's Guide describes how to configure the GATT database for a Bluetooth project. This document focuses on the other two parts – Device Composition Data and memory allocation. Tabs in the Bluetooth mesh Configurator allow you to change between the two, as shown in the following figure.

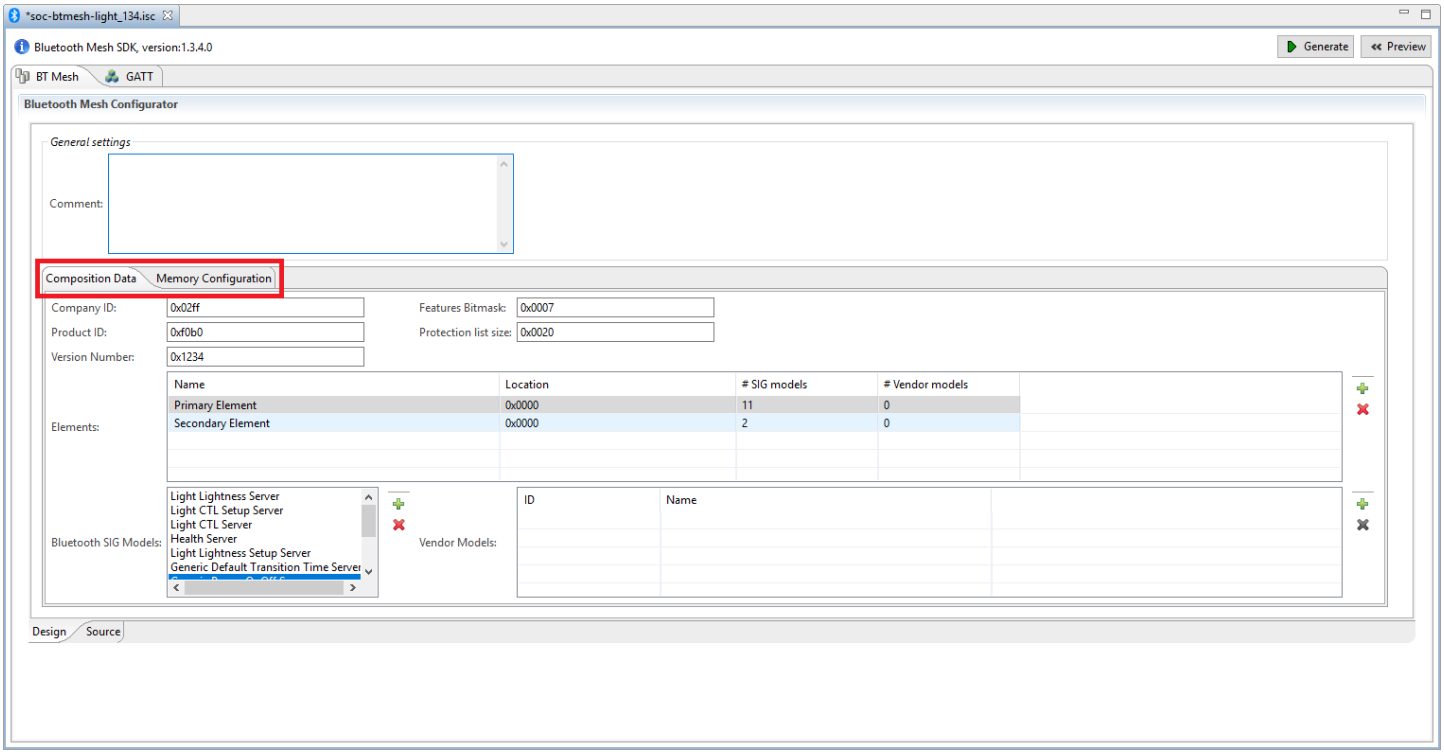

Figure 1-1. Graphic DCD and Memory Configurators

1.1 Terminology

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.

Term	Definition
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 Device Composition Data Configurator

All the items on the **Composition Data** tab compose the Device Composition Data. They are presented in three areas: device information, elements, and models.

2.1 Device Information

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

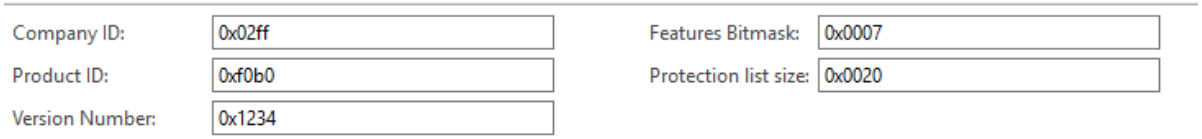

Figure 2-1. Device Information Section

The meaning of each field is shown in the following table.

Table 2-1. Device Information Fields

Field Name	Notes
Company ID	16-bit company identifier assigned by the Bluetooth SIG (available here)
Product ID	16-bit vendor-assigned product identifier, vendor-specific
Version Number	16-bit vendor-assigned product version identifier, vendor-specific
Features Bitmask	Bit field indicating the device features, as defined in the following table
Protection list size	16-bit value representing the minimum number of replay protection list entries in a device, refer to section 3.5 Rpl Size for more information

The Features Bitmask field indicates the capabilities of the node. The Feature Bitmask field format is shown in the following table.

Table 2-2. Features Bitmask Field Format

Bit	Feature	Notes
0	Relay	Relay feature support: 0 = False, 1 = True
1	Proxy	Proxy feature support: 0 = False, 1 = True
2	Friend	Friend feature support: 0 = False, 1 = True
3	Low Power	Low Power feature support: 0 = False, 1 = True
4-15	RFU	Reserved for Future Use

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. An elements editor, shown in the following figure, allows you to configure your project's elements. Click the green plus symbol (+) to add an element or select an element and click the red X symbol (x) to remove it.

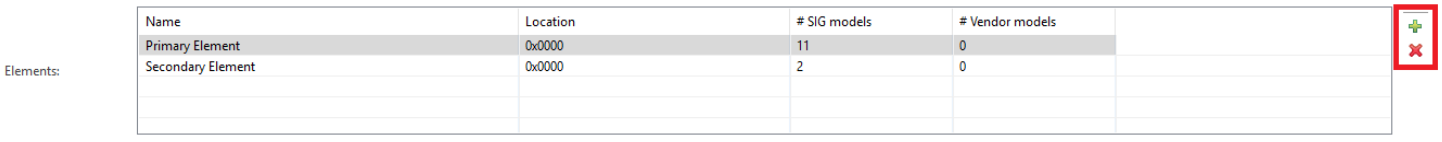

Figure 2-2. Elements Editor

2.3 Models

A model defines the basic functionality of a node, and 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.

Model specifications are designed to be very small and self-contained. At specification definition time, a model can require other models that must also be instantiated within the same node. This is called extending, which means a model adds functionality on top of other models.

Models that do not extend other models are referred to as root models. Model specifications are immutable. In other words, it is not possible to remove or add behavior to a model, whether the desired behavior is optional behavior or mandatory. Models are not versioned and have no feature bits. If additional behavior is required in a model, then a new extended model is defined that exposes the required behavior and can be implemented alongside the original model.

Therefore, knowledge of the models supported by an element determines the exact behavior exposed by that element.

The DCD configurator supports configuring both SIG-adopted models and vendor models through individual editors.

2.3.1 SIG-Adopted Model Editor

The SIG-adopted model editor is on the bottom left of the configurator. Once you click any element, the models included in the element are listed in the editor as shown in the following figure.

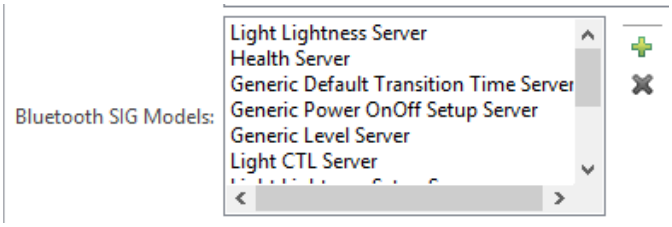

Figure 2-3. SIG-Adopted Model Editor

To delete a model, select it and click the red X symbol (x). To add a SIG-adopted model, click the green plus symbol (+). A list of all the SIG-adopted models is displayed, and you can choose the one you want to, as shown in the following figure. 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, please check the SDK release notes.

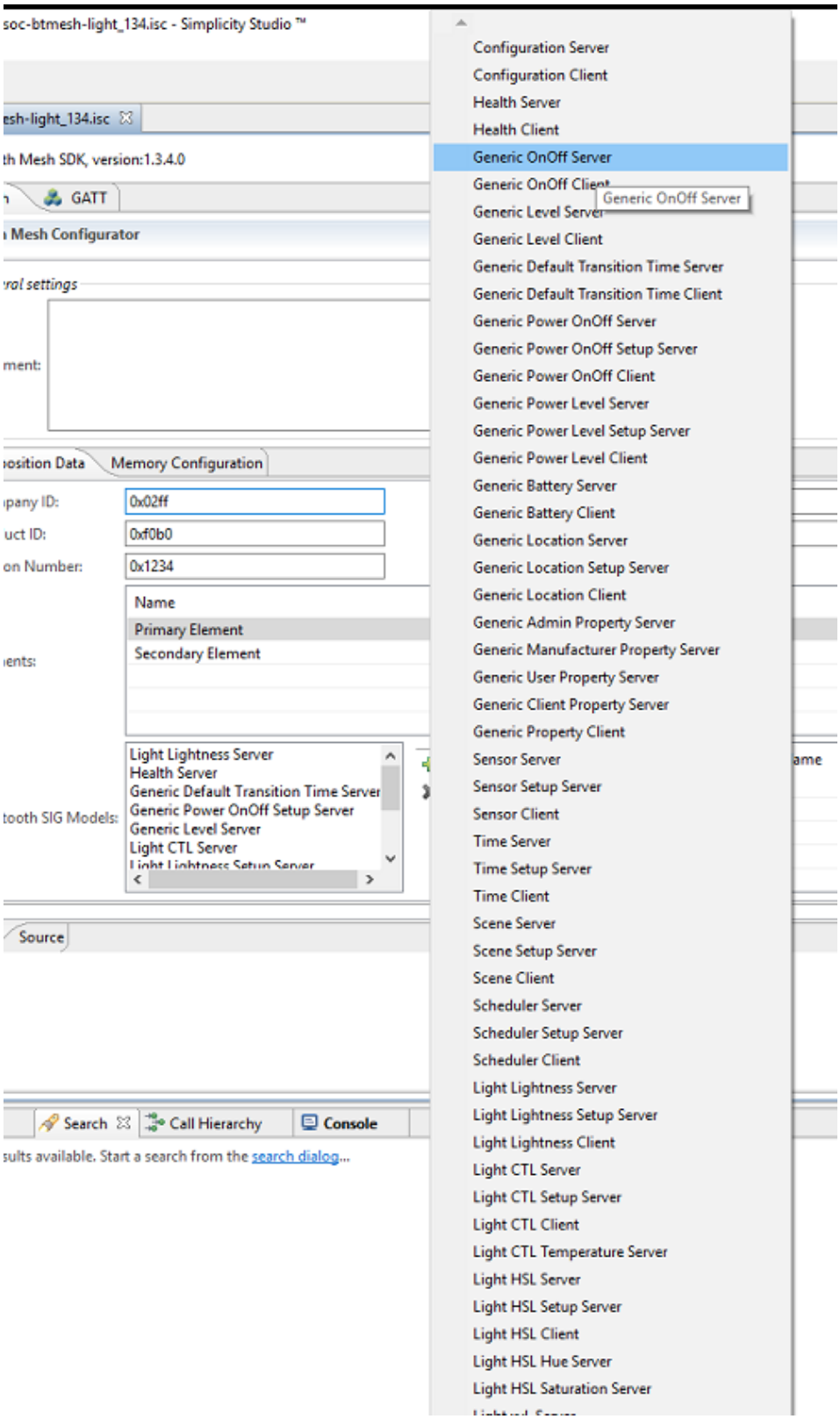

Figure 2-4. All SIG-Adopted Models

Due to the extension mechanism of models mentioned above, attention is needed 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 be also added. The Bluetooth Mesh Model Specification has the detailed definition of the models' relationship. For example, to add a Light Lightness Server model, you need to make sure that the Generic Power OnOff Server model and the Generic Level Server model are both also present in the settings, because the Light Lightness Server model is extended from them.
2. One element can only have one instance of any model. For example, if you want to add model A and Model B which are both extended from Model C to your project, you cannot add all of them to a single element because it requires 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, make sure to 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.

2.3.2 Vendor Models Editor

Vendor models give you more flexibility when developing products not covered by the SIG-adopted models. Vendors can define their own specification in these vendor 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 0x0001 is the model ID.

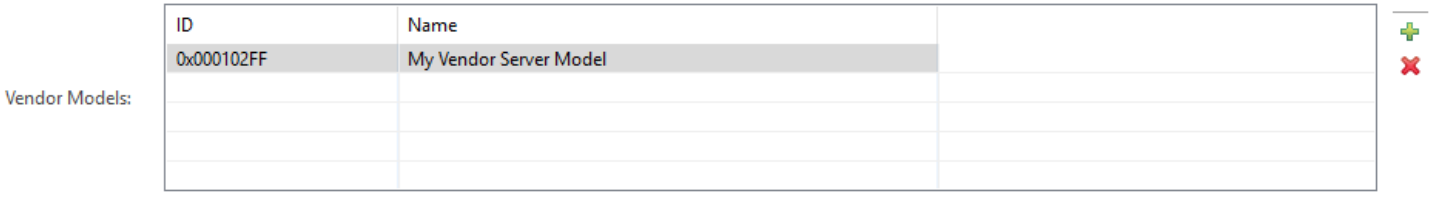

Figure 2-5. Vendor Model Editor

Click the green plus symbol (+) to add a vendor model, or select a model and click the red X symbol (x) to remove it.

3 Memory Configurator

The Bluetooth mesh SDK provides several configuration options for a mesh project to optimize the RAM and flash consumption. Memory configuration is needed because RAM and flash are usually limited in an embedded system. The appropriate configuration values avoids extra RAM and flash usage so that there is more space for the application. The following figure contains all the fields in the memory configuration tab.

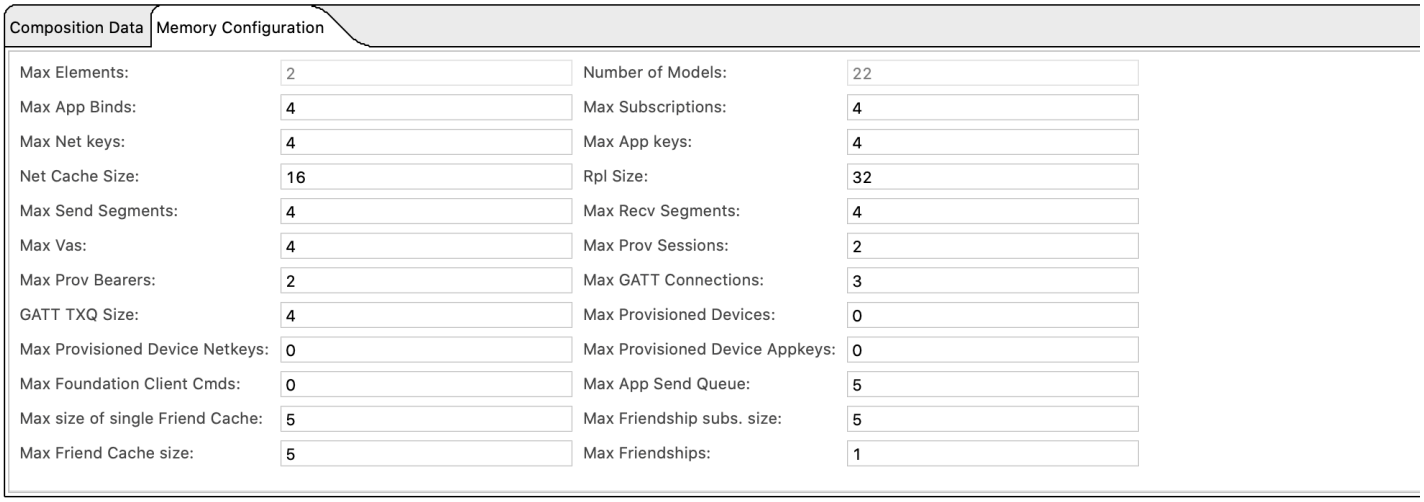

Figure 3-1. Memory Configurator

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

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

The following table summarizes the configuration options. Each option is then discussed in more detail.

Table 3-1. Summary of Configuration Options

Configuration Option	Description	Stored Persistently	Notes
Max Net keys	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.
Max App Keys	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.
Max App Binds	The maximum number of application keys that can be bound to a model.	Yes	No larger than the smaller value of *Max App Keys* and 255.
Max Subscriptions	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.
Max Provisioned Devices	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.

Configuration Option	Description	Stored Persistently	Notes
Rpl 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.
Max Vas	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.
Max Provisioned Device Netkeys	The maximum number of network keys on the peers provisioned by this device.	Yes	Only applicable if the device is in provisioner role.
Max Provisioned Device Appkeys	The maximum number of application keys on the peers provisioned by this device	Yes	Only applicable if the device is in provisioner role.
Max Recv Segments	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.
Max Send Segments	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.
Max Prov Sessions	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.
Max Foundation Client Cmds	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.
Net 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.
Max GATT Connections	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.
Max Prov Bearers	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.
Max Friendships	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.
Max Friend Cache size	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.
Max size of single Friend Cache	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.
Max Friendship subs.size	The maximum number of addresses that can be stored in the Friend Subscription List.	No	Only applicable for friend node.
GATT TXQ Size	Queue size for messages over GATT bearer.	No	Connection interval-dependent.
Max App Send Queue	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	

3.1 Max Net Keys

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.

3.2 Max App Keys

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.

3.3 Max App Binds

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.

3.4 Max Subscriptions

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.

3.5 Max Provisioned Devices

This setting is appliable only when the device is in 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.

3.6 Rpl 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. Since 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 blacklisted or 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 blacklisted 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.

3.7 Max Vas

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.

3.8 Max Provisioned Device Netkeys

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.

3.9 Max Provisioned Device Appkeys

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.

3.10 Max Recv Segments

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.

3.11 Max Send Segments

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 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.

3.12 Max Prov Sessions

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.

3.13 Max Foundation Client Cmds

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.

3.14 Net 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.

3.15 Max GATT Connections

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.

3.16 Max Prov 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.

3.17 Max Friendships

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.

3.18 Max Friend Cache Size

As mentioned in Section 3.17 Max Friendships, 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.

3.19 Max Size of Single Friend Cache

The ***Max Friend Cache Size*** setting (section 3.17 Max Friendships) 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 ***Max Friend Cache Size*** 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 ***Max Friend Cache Size*** setting. Set to 0 if the node does not support the friend feature.

3.20 Max Friendship Subs.size

As mentioned in section 3.17 Max Friendships, 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.

3.21 GATT TXQ 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.

3.22 Max App Send Queue

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.

4 After Configuration

After editing the configurations in the .isc file, click **Generate** in the upper right corner of the interface to bring all the changes and configurations into effect, as shown in the following figure.

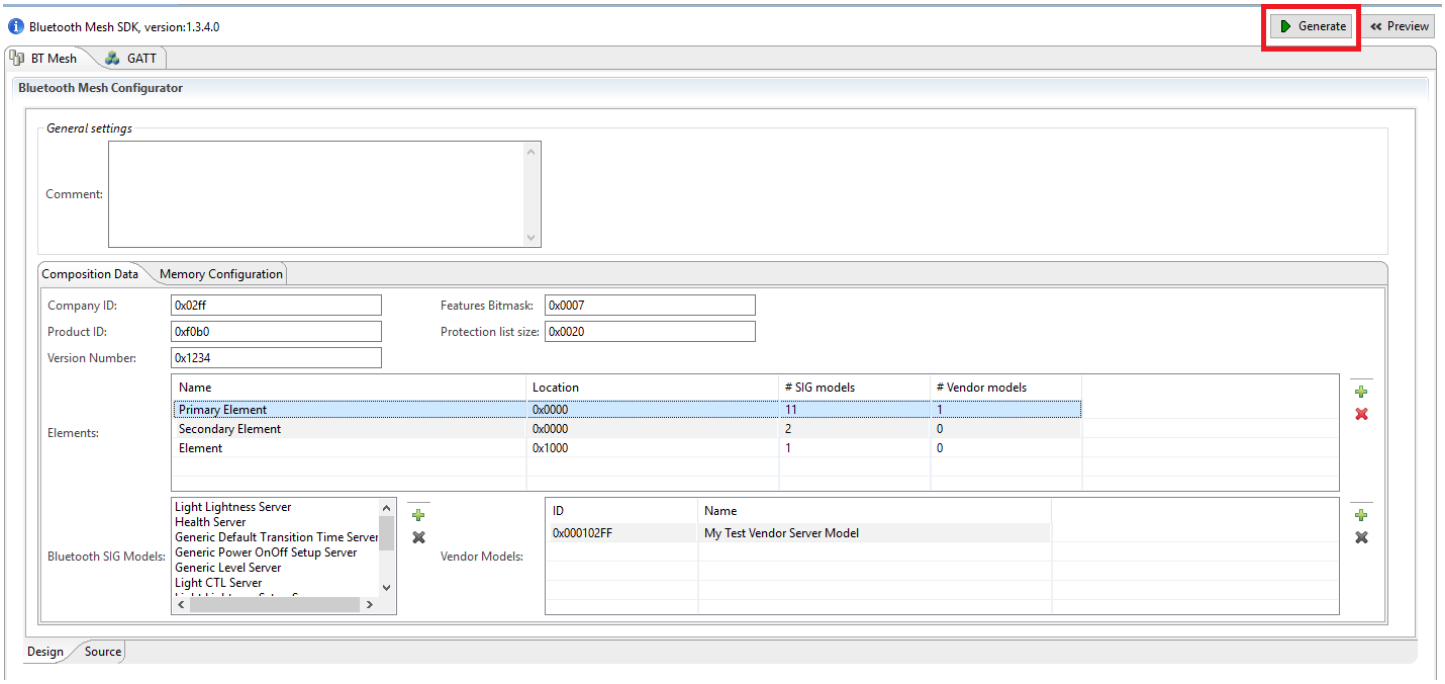

Figure 4-1. Generate Button

When you click **Generate**, Simplicity Studio will:

- Generate dcd.c and mesh_app_memory_config.h files, which are used to pass the configuration data to the Bluetooth mesh stack.
- Back up the current files, which will be overwritten by the new settings, if needed, as shown in the following figure.

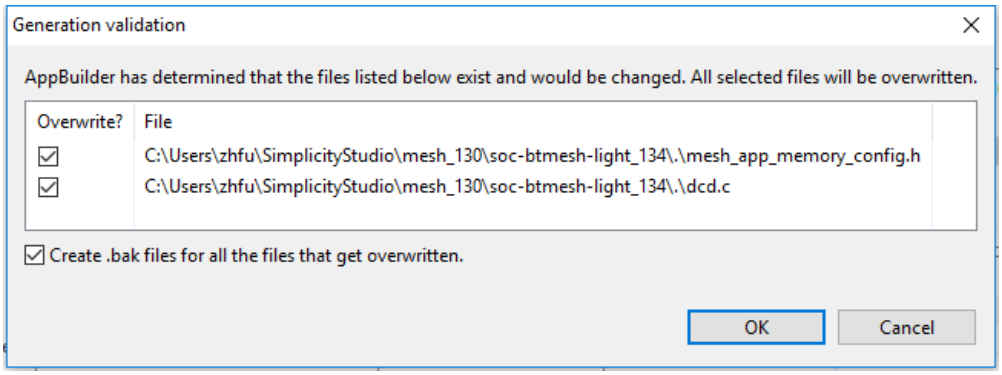

Figure 4-2. Generation Validation

When you click **OK**, you see the modification details, as shown in the following figure.

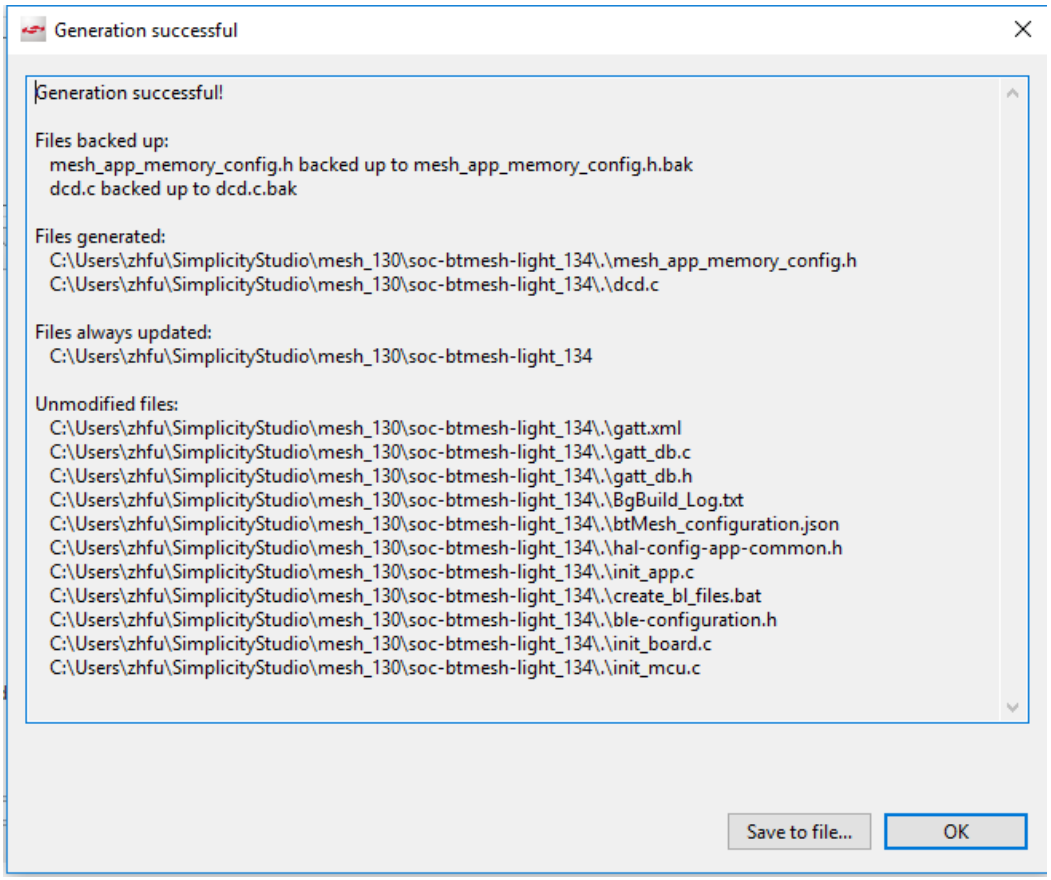

Figure 4-3. File Modification Details

Now when you compile the project, the new settings will take effect. However, if devices already have settings in persistent storage, the Bluetooth mesh stack will not overwrite those settings. You must completely erase the persistent storage for the new configuration to take effect.

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

http://www.silabs.com