Qualcomm Technologies International, Ltd.

Qualcomm Technologies International, Ltd.
Churchill House
Cambridge Business Park
Cambridge, CB4 0WZ
United Kingdom

CSR

Push every boundary.™

CSR µEnergy™

CSRmesh™ 2.0 Light Application

Application Note

Issue 3

Document History

Revision	Date	History
1	30 SEP 2015	Original publication of this document.
2	27 OCT 2015	Editorial updates.
3	30 OCT 2015	Editorial updates.

Contacts

General information	www.csr.com
Information on this product	sales@csr.com
Customer support for this product	www.csrsupport.com
More detail on compliance and standards	product.compliance@csr.com
Help with this document	comments@csr.com

Performance and Conformance

Refer to www.csrsupport.com for compliance and conformance to standards information.

CSR

Contents

Tables and Figures

1. Introduction

This document describes the CSRmesh™ Light on-chip application built using the CSR µEnergy™ SDK.

The application has the following use cases:

- Light Control: The Light application implements the handler for the CSRmesh messages related to the Light Model, Power Model, Stream Model and Attention Model.
- CSRmesh GATT Bridge: The application also implements the CSR custom Mesh Control Service. This service allows a Bluetooth® Smart enabled phone to send and receive CSRmesh messages to many devices with the Light application acting as a bridge.

The CSRmesh Light application is part of the CSRmesh release and shows the CSRmesh light control use case. It uses the CSRmesh library provided as part of the CSRmesh release. For more information, see *CSRmesh API Guide*.

The CSRmesh Light application does not support bonding and is not compatible with previous CSRmesh releases.

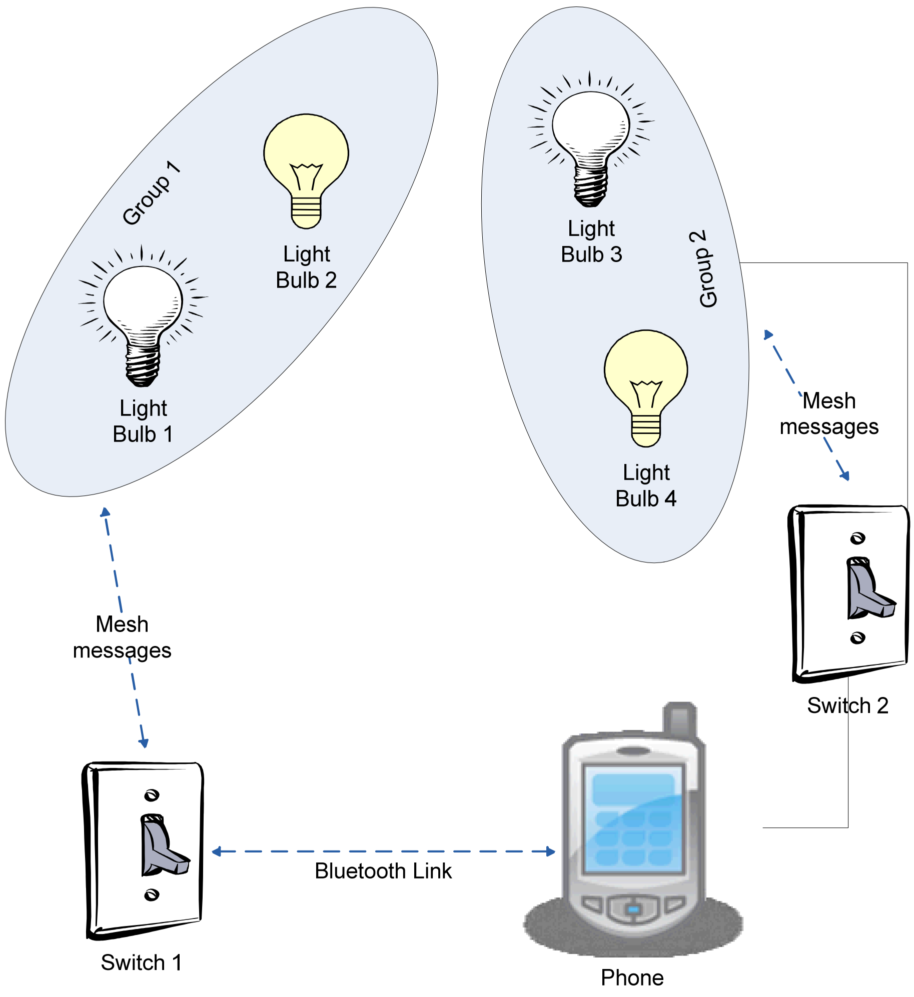


Figure 1.1: CSRmesh Use Case

1.1. Application Overview

The CSRmesh Light application uses the CSRmesh library API to communicate with other associated devices in the same CSRmesh network. Additionally it supports a custom GATT profile to allow the control of CSRmesh from a Bluetooth Smart enabled device.

1.1.1. Supported Profiles

The CSRmesh Light application implements the following CSR custom profiles to support the use cases.

1.1.1.1. CSRmesh Control Profile

The CSRmesh Control Profile defines the behaviour when:

- A network of devices such as lights and switches needs to be created.
- Controlling the device after a network is created, for example, switching the light on/off or changing the intensity or colour of a light.
- Reading the status of a device in the network, for example, on/off state, colour or intensity of a light.

Table 1.1 lists the two roles that the CSRmesh Control Profile defines.

Role	Description	Implementation
CSRmesh Bridge Device	Receives commands from the host and sends them over the CSRmesh network to associated devices. Receives responses from the associated devices over the CSRmesh and forwards them to the host over a Bluetooth Smart connection.	On the CSRmesh Light application
CSRmesh Control Device	Provides the interface to create a network of devices and to control the associated devices. The control commands are sent over a Bluetooth Smart connection to the CSRmesh devices.	On a Bluetooth Smart enabled phone or tablet

Table 1.1: CSRmesh Control Profile Roles

1.1.2. Application Topology

Table 1.2 and Table 1.3 lists the topology that the CSRmesh Bridge uses.

Role	Mesh Control Service	GAP Service	GATT Service	CSR OTA Update Application Service
GATT Role	GATT Server	GATT Server	GATT Server	GATT Server
GAP Role	Peripheral	Peripheral	Peripheral	Peripheral

Table 1.2: Application Topology

Role	Responsibility
GATT Server	Accepts incoming commands and requests from a client and sends responses, indications and notifications to the client.
GAP Peripheral	Accepts connections from the remote device and acts as a slave in the connection.

Table 1.3: Roles and Responsibilities

For more information about GATT server and GAP peripheral, see *Bluetooth Core Specification Version 4.1*.

1.1.3. Services Supported in GATT Server Role

The application exposes the following services:

- Mesh Control Service v2.0
- GATT Service
- GAP Service
- CSR OTA Update Application Service v7

The Mesh Control Service is mandated by the CSRmesh Control Profile. The GATT and GAP Services are mandated by the *Bluetooth Core Specification Version 4.1*.

Figure 1.2 shows the services supported in the GATT Server Role.

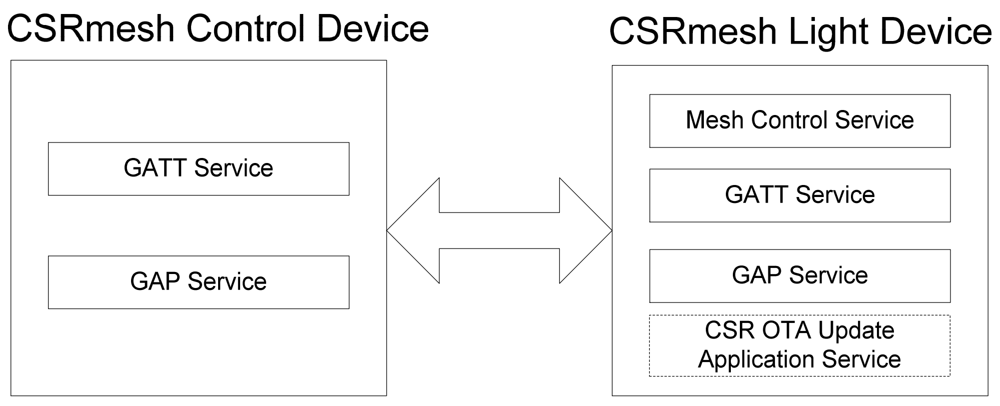


Figure 1.2: Primary Services

1.1.3.1. CSR OTA Update Application Service

The CSR OTA Update Application Service enables wireless update of the application software. A PC or mobile phone application provided by the device manufacturer enables the end-user to keep their device up-to-date with the latest features and bug fixes.

To enable a device for future OTA updates, the application needs to:

- Add OTA Update functionality to the on-chip application.
- Add support for the CSR OTA Update Application Service and GATT Services to an application.
- Configure the on-chip bootloader.

The CSR OTA Update bootloader image must be present on the device and configured to contain the correct device name and optional shared authentication key.

When the device is enabled for OTA Update, the CSR µEnergy Over-the-Air Updater host application included in the SDK can update the device.

For more information about CSR OTA Update, see:

- *CSR µEnergy Over-the-Air (OTA) Update System Application Note*
- *CSR µEnergy Modifying an Application to Support OTA Update Application Note*
- *CSR µEnergy Over-the-Air (OTA) Update Application and Bootloader Services Specification*

For information about CSR OTA Update applications for iOS and Android, see www.csrsupport.com.

2. Using the Application

This section describes how to use the CSRmesh Light application with CSRmesh Control Application to control devices.

2.1. Demonstration Kit

Table 2.1 lists the components that the application can use for demonstration.

Component	Hardware	Application
Light Device	CSRmesh Development PCB (DB-CSR1010-10185-1A) OR Gunilamp	CSRmesh Light Application v2.0
CSRmesh Android Control Device	Android Bluetooth LE Device	CSRmesh Control Application v2.0
CSRmesh iOS Control Device	iOS Bluetooth LE Device	CSRmesh Control Application v2.0

Table 2.1: CSRmesh Components

2.1.1. CSRmesh Development Board

The CSR µEnergy SDK is used to download the CSRmesh Light application on the development boards. For more information, see *CSR µEnergy xIDE User Guide*.

Ensure that the development board is powered on using the Power Slider switch. Figure 2.1 shows the switch in the off position.

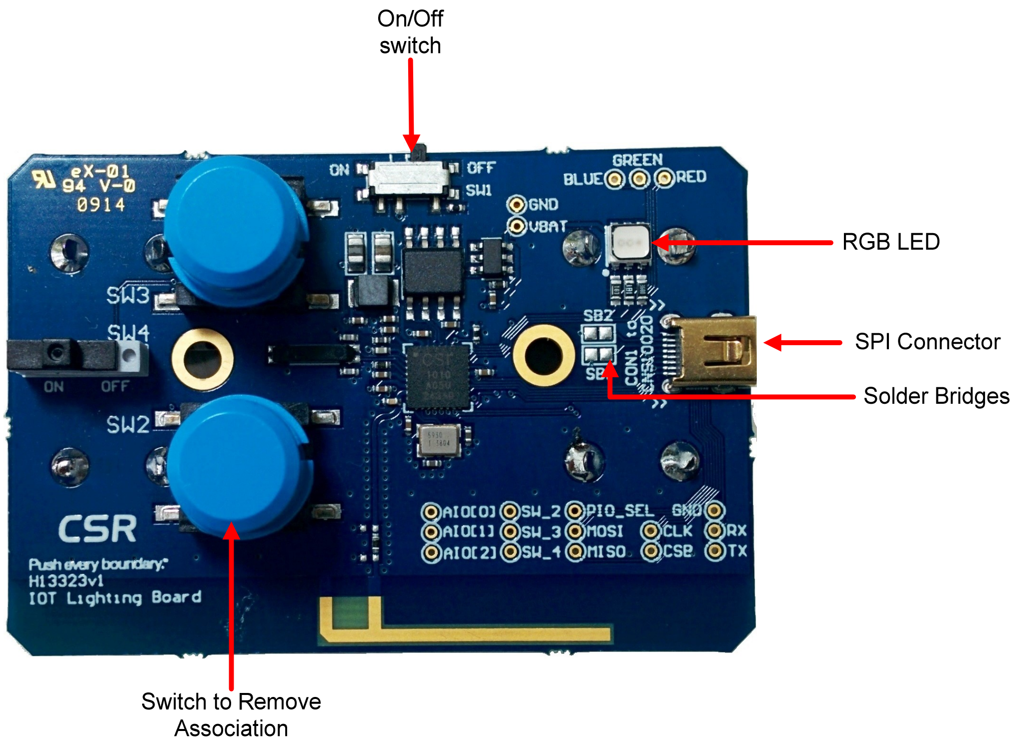


Figure 2.1: CSRmesh Development Board

Note:

When the development board is disconnected from the USB to SPI Adapter, wait at least 1 minute before powering the board on to allow dissipation of any residual charge received from the SPI connector.

Shorting the solder bridges exposes the UART through the CSR SPI connector. The PIOs connected to SW2 and SW3 are also mapped to UART Rx and Tx lines. Pressing any of these buttons shorts the UART lines to ground and corrupts data on the UART. CSR recommends not pressing these buttons during UART communication.

2.1.1.1. User Interface

The application uses the buttons available on the CSRmesh development board for the CSRmesh Sniffer application.

Table 2.2 lists the user interfaces of the CSRmesh development board.

User Interface Component	Function
Switch SW1	Power slider switch powering on/off the board.
Button SW2	Holding this button for more than 2 seconds removes the network association. This behaviour can be enabled or disabled by user configuration. For more information, see Section 6.8.
Button SW3	Unused
Switch SW4	Unused
RGB LED	▪ Blinks blue until it is not associated with any CSRmesh network. ▪ Blinks yellow when device association is in progress. ▪ Blinks red when device attention is requested. Light colour is set by light control messages after the device is associated.

Table 2.2: CSRmesh Development Board User Interface

2.1.1.2. CSRmesh Device Tag

The CSRmesh development board is supplied with a CSRmesh Device Tag sticker shown in Figure 2.2. The sticker contains:

- BT: Device Bluetooth Address
- SN: Serial Number
- XTAL TRIM: Crystal trim value to be set in the CS Config file
- UUID: CSRmesh Device UUID
- AC: CSRmesh Authorisation Code
- QR-Code: Encodes UUID and AC

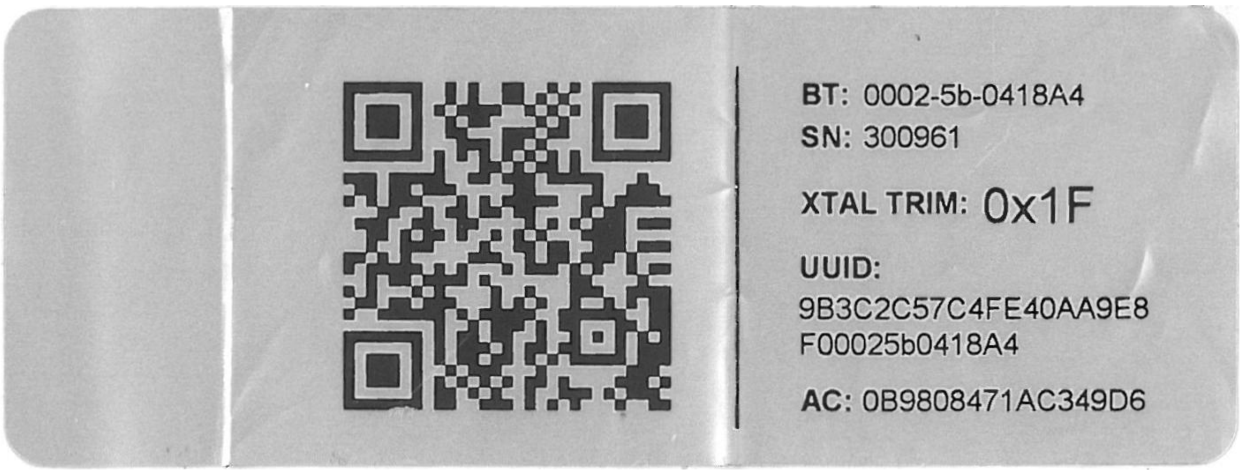


Figure 2.2: CSRmesh Device Tag Sticker

You can program the device with the Bluetooth address and XTAL Trim printed on the sticker by setting these values in the `CSRmesh_light_csr101x_A05.keyr` file.

The Device UUID and the Authorisation Code printed on the sticker can be programmed on the NVM at the offsets defined in Table 5.1 using the USB to SPI adapter.

To program an example UUID `0x0123456789ABCDEFFEDCBA9876543210` and Authorisation Code `0x0123456789ABCDEF` on the NVM:

1. Open a command prompt.
2. Program the device UUID and Authorisation Code to the device EEPROM over the SPI link:
   - If base `NVM_START_ADDRESS` is defined in the `.keyr` file, program the device UUID and Authorisation Code to the device as follows:

     ```
     <CSR_uEnergy_Tools path>\uEnergyProdTest.exe -k
     CSRmesh_light_csr101x_A05.keyr -m1 0x02 0x3210 0x7654 0xBA98 0xFEDC
     0xCDEF 0x89AB 0x4567 0x0123 0xCDEF 0x89AB 0x4567 0x0123
     ```

     The first value following `-m1` is the NVM offset from `NVM_START_ADDRESS`. The command takes the NVM offset as the byte address. Word offset 1 for device UUID is byte offset 2, see Table 5.1.
   - If the `.keyr` file is not included in the command, program the device UUID and Authorisation Code to the device as follows:

     ```
     <CSR_uEnergy_Tools path>\uEnergyProdTest.exe -m1 0x4102 0x3210 0x7654
     0xBA98 0xFEDC  0xCDEF 0x89AB 0x4567 0x0123 0xCDEF 0x89AB 0x4567
     0x0123
     ```

     The first value following `-m1` is the NVM address obtained by adding base address `4100` and word offset 1, see Table 5.1 for the device UUID. The command takes the NVM address as the byte address. Word offset 1 is byte offset 2, so the effective address is `0x4102`.

The CSRmesh Control application reads the device Authorisation Code and the UUID from the QR-Code printed on the sticker during association. For more information, see *CSRmesh 2.0 Android Control Application Note* or *CSRmesh 2.0 iOS Control Application Note*. For more information about programming the NVM, see *CSRmesh 2.0 Production Test Tool User Guide*.

2.1.2. CSRmesh Control Application

The CSRmesh Control application runs on an Android or iOS device that supports BLE. It communicates with the CSRmesh devices by connecting to one of the devices that supports the CSR custom-defined CSRmesh Control Profile. The application is required for:

- Setting up a network by associating devices.
- Configuring and grouping the network devices.

For more information about using the CSRmesh Control application on an Android device, see *CSRmesh 2.0 Android Control Application Note*.

For more information about using the CSRmesh Control application on an iOS device, see *CSRmesh 2.0 iOS Control Application Note*.

For more information about the supported iOS and Android version, see *CSRmesh 2.0 Mobile Applications Release Note*.

3. Application Structure

This section describes the source files, head files and database files.

3.1. Source Files

Table 3.1 lists the source files.

File Name	Description
`app_data_stream.c`	Handles data stream model events. Implements a protocol to exchange large blocks of data with other CSRmesh devices. See Section 4.7 for more information.
`app_fw_event_handler.c`	Defines the handler functions for firmware events.
`app_mesh_event_handler.c`	Defines the handler functions for CSRmesh events.
`battery_hw.c`	Implements the routines to read battery level.
`csr_mesh_light.c`	Implements all the entry functions such as `AppInit()`, `AppProcessSystemEvent()`, `AppProcessCsrMeshEvent()` and `AppProcessLmEvent()`. Handles events received from the hardware, CSRmesh network and firmware first. Contains handling functions for all the LM and system events.
`csr_mesh_ight_gatt.c`	Implements routines for triggering advertisement procedures.
`csr_mesh_light_hw.c`	Implements the abstract hardware interface to configure and control the peripherals on specific hardware.
`csr_mesh_light_util.c`	Implements some utilities functions called by other handlers.
`csr_ota_service.c`	Implements the routines for handling the read/write access on the CSR custom-defined OTA Update Service.
`fast_pwm.c`	Implements the fast PWM output using the PIO controller.
`gap_service.c`	Implements routines for GAP Service such as handling read/write access indications on the GAP Service characteristics and reading/writing device name on NVM.
`gatt_service.c`	Defines routines for using GATT Service.
`iot_hw.c`	Implements the hardware interface to configure and control the peripherals on the CSRmesh development board.
`mesh_control_service.c`	Implements routines for handling read/write access on Mesh Control characteristics and for sending notifications of mesh device responses.
`nvm_access.c`	Implements the NVM read/write routines.
`pio_ctrlr_code.asm`	Implements the 8051 assembly code for fast PWM.

Table 3.1: Source Files

3.2. Header Files

Table 3.2 lists the header files.

File Name	Description
app_data_stream.h	Contains enumerations and function prototypes of the externally referenced functions defined in app_data_stream.c.
app_debug.h	Contains macro definitions for enabling debugging prints.
app_fw_event_handler.h	Contains prototypes of the externally referenced functions defined in app_fw_event_handler.c.
app_gatt.h	Contains macro definitions, user-defined data type definitions and function prototypes used across the application.
app_mesh_event_handler.h	Contains prototypes of the externally referenced functions defined in app_mesh_event_handler.c.
appearance.h	Contains the appearance value macro of the application.
battery_hw.h	Contains prototypes of the externally referenced functions defined in battery_hw.c.
csr_mesh_light.h	Contains the macros and definitions used in the csr_mesh_light.c application file.
csr_mesh_light_gatt.h	Contains macro definitions for enabling debugging prints.
csr_mesh_light_hw.h	Contains the function declarations to control the hardware interfaces.
csr_mesh_light_util.h	Contains prototypes of the externally referenced functions defined in the csr_mesh_light_util.c file.
csr_ota_service.h	Contains prototypes of the externally referenced functions defined in the csr_ota_service.c file.
csr_ota_uuids.h	Contains macros for UUID values for CSR OTA Update Service.
fast_pwm.h	Contains macro definitions and prototypes of the externally defined functions in fast_pwm.c.
gap_conn_params.h	Contains macro definitions for fast/slow advertising, preferred connection parameters, idle connection timeout values and so on.
gap_service.h	Contains prototypes of the externally referenced functions defined in the gap_service.c file.
gap_uuids.h	Contains macros for UUID values for GAP Service.
gatt_service.h	Contains prototypes of the externally referenced functions defined in the gatt_service.c file.
gatt_service_uuids.h	Contains macros for UUID values for GATT Service.
gunilamp_hw.h	Contains the function declarations to control the Gunilamp hardware interfaces.
iot_hw.h	Contains the function declarations to control the IoT hardware interfaces.
mesh_control_service.h	Contains prototypes of the externally referenced functions defined in the mesh_control_service.c file.

mesh_control_service_uuids.h	Contains macros for UUID values for Mesh Control Service.
nvm_access.h	Contains prototypes of the externally referenced NVM read/write functions defined in the nvm_access.c file.
ota_customisation.h	Contains definitions for the application data variables used by csr_ota_service.c.
user_config.h	Contains macros for customising the application.

Table 3.2: Header Files

3.3. Database Files

The SDK uses database files to generate the attribute database for the application. For instructions about how to write database files, see *GATT Database Generator User Guide*.

Table 3.3 lists the database files.

File Name	Description
app_gatt_db.db	Master database file including all service specific database files. Is imported by the GATT Database Generator.
csr_ota_db.db	Contains information related to CSR OTA Update service characteristics, their descriptors and values. For more information about CSR OTA Service characteristics, see Table A.3
gap_service_db.db	Contains information related to GAP Service characteristics, their descriptors and values. For more information about GAP Service characteristics, see Table A.2.
gatt_service_db.db	Contains information related to GATT Service characteristics, their descriptors and values. For more information about GATT Service characteristics, see Table A.1.
mesh_control_service_db.db	Contains information related to CSRmesh Control service characteristics, their descriptors and values. For more information about CSRmesh Control Service characteristics, see Table A.4.

Table 3.3: Database Files

4. Code Overview

This section describes significant functions of the application.

4.1. Application Entry Points

4.1.1. AppInit()

This function is invoked when the application is powered on or the chip resets. It performs the following initialisation functions:

- Initialises the application timers, hardware and application data structures.
- Configures GATT entity for server role.
- Configures the NVM manager to use I^{2}C EEPROM.
- Initialises all services.
- Reads the persistent storage. Sets a random device UUID based on Bit 3 of the `CS_USER_KEY 1` setting when the application runs for the first time.
- Initialises the CSRmesh stack:
  - Configures CSRmesh bearer parameters and initialises the CSRmesh Scheduler:

    `CSRSchedSetConfigParams(params)`

    `CSRSchedStart()`
  - Registers an application call-back function for handling core mesh events: `CSRmeshRegisterAppCallback(CSRmeshAppProcessMeshEvent)`
  - Initialises the Core Stack: `CSRmeshInit()`
  - Initialises the supported model server and client modules.
- Registers the attribute database with the firmware.

4.1.2. AppProcessLmEvent()

This function is invoked whenever an LM-specific event is received by the system. Section 4.1.2.1 to Section 4.1.2.5 describe the events handled by this function.

4.1.2.1. Database Access

- `GATT_ADD_DB_CFM`: This confirmation event marks the completion of database registration with the firmware. On receiving this event, the application starts advertising.
- `GATT_ACCESS_IND`: This indication event is received when the CSRmesh Control device tries to access an ATT characteristic managed by the application.

4.1.2.2. LS Events

- `LS_CONNECTION_PARAM_UPDATE_CFM`: This confirmation event is received in response to the connection parameter update request by the application. The connection parameter update request from the application triggers the L2CAP connection parameter update signalling procedure. See Volume 3, Part A, Section 4.20 of *Bluetooth Core Specification Version 4.1*.
- `LS_CONNECTION_PARAM_UPDATE_IND`: This indication event is received when the remote central device updates the connection parameters. On receiving this event, the application validates the new connection parameters against the preferred connection parameters and triggers a connection parameter update request if the new connection parameters do not comply with the preferred connection parameters.
- `LS_RADIO_EVENT_IND`: This radio event indication is received when the chip firmware receives an acknowledgement for the Tx data sent by the application. On receiving this event, the application aligns the timer wakeup, which sends data periodically to the collector with the latent connection interval.

4.1.2.3. LM Events

- `LM_EV_ADVERTISING_REPORT`: This event is received when an advertisement packet is received. This can be a CSRmesh advertisement. The application passes the event data to the CSRmesh library to process the packet by calling the `CSRSchedHandleIncomingData()` API.
- `LM_EV_CONNECTION_COMPLETE`: This event is received when the connection with the master is considered to be complete and includes the new connection parameters.
- `LM_EV_DISCONNECT_COMPLETE`: This event is received on link disconnection. Disconnection can be locally triggered or triggered by the remote connected device due to link loss.
- `LM_EV_ENCRYPTION_CHANGE`: This event indicates a change in the link encryption.
- `LM_EV_CONNECTION_UPDATE`: This event indicates that the connection parameters are updated to a new set of values and is generated when the connection parameter update procedure is either initiated by the master or slave. These new values are stored by the application for comparison against the preferred connection parameter, see Section 6.2.

4.1.2.4. SMP Events

- `SM_SIMPLE_PAIRING_COMPLETE_IND`: This indication event indicates that pairing completes successfully or otherwise. See *Volume 3, Part H, Section 2.4 and Section 3.6 of Bluetooth Core Specification Version 4.1 Connection Events*.

4.1.2.5. GATT Events

- `GATT_CONNECT_CFM`: This confirmation event indicates that the connection procedure completes. If it does not successfully complete, the application goes to the idle state and waits for a user action. For more information about application states, see Section 4.2.
- `GATT_CANCEL_CONNECT_CFM`: This confirmation event confirms cancellation of the connection procedure. When the application stops advertisements to change advertising parameters or to save power, this signal confirms the successful stopping of advertisements by the CSRmesh application.

4.1.3. AppProcessSystemEvent()

This function handles the system events such as a low battery notification or a PIO change. CSRmesh applications currently handle the following system events:

- `sys_event_pio_changed`: This event indicates a change in the PIO value. Whenever the user presses or releases the button, the corresponding PIO value changes and the application receives a PIO changed event and takes the appropriate action.

4.1.4. CSRmeshAppProcessMeshEvent ()

This function handles the CSRmesh core stack events received from the CSRmesh network or caused by internal state change.

4.1.4.1. Network Association Messages

- `CSR_MESH_ASSOC_STARTED_EVENT`: This event is received when a CSRmesh Control application sends an association request to a light that is ready for association. The application starts blinking yellow to indicate that the association is in progress.
- `CSR_MESH_KEY_DISTRIBUTION`: This event is received when the CSRmesh Control device provides the network key used for all future messages to communicate on the network. The application switches the state to associate, switches the LED to indicate association completion and stores the association status on the NVM.
- `CSR_MESH_ASSOC_COMPLETE_EVENT`/`CSR_MESH_SEND_ASSOC_COMPLETE_EVENT`: One of these events is received when the association completes. The application updates the network association state and stops the ready for association LED display. It disables the promiscuous state so that the device relays only known network messages.
- `CSR_MESH_ASSOCIATION_ATTENTION_EVENT`: This event is received when a CSRmesh Control application seeks the attention of the device. The application starts blinking green to display attention. The attention display is continued till timeout or the association state changes.

- `CSR_MESH_CONFIG_RESET_DEVICE_EVENT`: This event is received when a configuring device removes the device. The association with the network is removed and the application uses this event to clean up the model data and sets the device in ready for association state.
- `CSR_MESH_BEARER_STATE_EVENT`: This message is received when a configuring device sends a bearer state change message.

4.1.4.2. Device Configuration Messages

- `CSR_MESH_CONFIG_RESET_DEVICE_EVENT`: This message is received when a configuring device wants to remove all CSRmesh network information from the device. The application resets all the assigned model group IDs.

4.1.4.3. Device Information Messages

- `CSR_MESH_OPERATION_REQUEST_FOR_INFO`: is received by the application for the following messages. It has sub events related to device information.
- `CSR_MESH_GET_VID_PID_VERSTION_EVENT`: This message is received when configuring device requests for Vendor Identifier, Product Identifier and Version number information from the device. The application sends this information to the library for transmission.
- `CSR_MESH_GET_DEVICE_APPEARANCE_EVENT`: This message is received when configuring device requests for device appearance information from the device. The application sends this information to the library for transmission.

4.1.4.4. Group Model Messages

- `CSR_MESH_GROUP_SET_MODEL_GROUPID_EVENT`: This message is received when the control device sets a new group ID to a supported model. The CSRmesh Light application stores the assigned Group IDs in the Group ID list for the Model and saves it on the NVM.

4.1.4.5. Bearer Model Messages

- `CSR_MESH_BEARER_STATE_EVENT`: The application enables or disables the relay and promiscuous mode set in the message when this message is received.

4.1.5. AppLightEventHandler

This function handles the CSRmesh Light model events received from the CSRmesh network, such as setting the light state, status or any other responses for Light model commands sent over CSRmesh:

- `CSRMESH_LIGHT_SET_RGB`: This event is received when a CSRmesh Switch or the Control application sends a command to set the light colour.
- `CSRMESH_LIGHT_SET_RGB_NO_ACK`: The handling is the same as that of `CSRMESH_LIGHT_SET_RGB` but has no response sent to the initiating device.
- `CSRMESH_LIGHT_SET_LEVEL`: This event is received when a CSRmesh Switch or the Control application sends a command to set the brightness level. The application sets the PWM duty cycle on the PIOs controlling the RGB LED corresponding to the RGB values received.
- `CSRMESH_LIGHT_SET_LEVEL_NO_ACK`: The handling is the same as that of `CSR_MESH_LIGHT_SET_LEVEL` but has no responses sent to the initiating device.
- `CSRMESH_LIGHT_SET_POWER_LEVEL/CSRMESH_LIGHT_SET_POWER_LEVEL_NO_ACK`: The application sets the LED to the new power level.
- `CSRMESH_LIGHT_SET_COLOR_TEMP`: This event is received when the Control application sends a command to set the colour temperature.
- `CSRMESH_LIGHT_GET_STATE`: The application returns the latest values of the light state parameters.

4.1.6. AppPowerEventHandler

This function handles the CSRmesh Power model events received from the CSRmesh network, such as setting the power state or any other responses for Power model commands sent over CSRmesh.

- `CSRMESH_POWER_TOGGLE_STATE`: This event is received when the control device sends a toggle power state command.

- `CSRMESH_POWER_TOGGLE_STATE_NO_ACK`: This event is received when the control device sends a toggle power state command, except that no response is sent to the initiating device.
- `CSRMESH_POWER_SET_STATE`: This event is received when the control device sends a power set state command.
- `CSRMESH_POWER_SET_STATE_NO_ACK`: This event is received when the control device sends a power set state command, except that no response is sent to the initiating device.

4.1.7. AppBatteryEventHandler

This function handles the CSRmesh Battery model events received from the CSRmesh network, such as getting the battery state or any other responses for Battery model commands sent over CSRmesh:

- `CSRMESH_BATTERY_GET_STATE`: This message is received when the control device queries the battery status. The application returns the current battery status.

4.1.8. AppAttentionEventHandler

This function handles the CSRmesh Attention model events received from the CSRmesh network, such as setting the attention state or any other responses for Attention model commands sent over CSRmesh:

- `CSRMESH_ATTENTION_SET_STATE`: This message is received when a control device requests the attention of a device in the network. The application blinks red when Attention state is set and resumes the last set light colour when the Attention state is reset.

4.1.9. AppDataServerHandler

This function handles the CSRmesh Data Stream model events received from the CSRmesh network when the device is in server role:

- `CSRMESH_DATA_STREAM_FLUSH`: This message is received when a Data stream Client wants to start sending a data stream or to indicate complete transmission of data in the current stream.
- `CSRMESH_DATA_STREAM_SEND`: This message is received when the next stream data block is received from the data stream client. The application verifies the type of message being streamed and stores it in the device info string.
- `CSRMESH_DATA_BLOCK_SEND`: This message is received when a data stream client sends a single block of data.

4.1.10. AppDataClientHandler

This function handles the CSRmesh Data Stream model events received from the CSRmesh network when the device is in client role:

- `CSRMESH_DATA_STREAM_RECEIVED`: This message is received when the data stream server acknowledges the reception of a stream data block sent by the device using the `StreamSendData` function.

4.2. Internal State Machine for GATT Connection

Figure 4.1 shows the different state transitions in the application during connection.

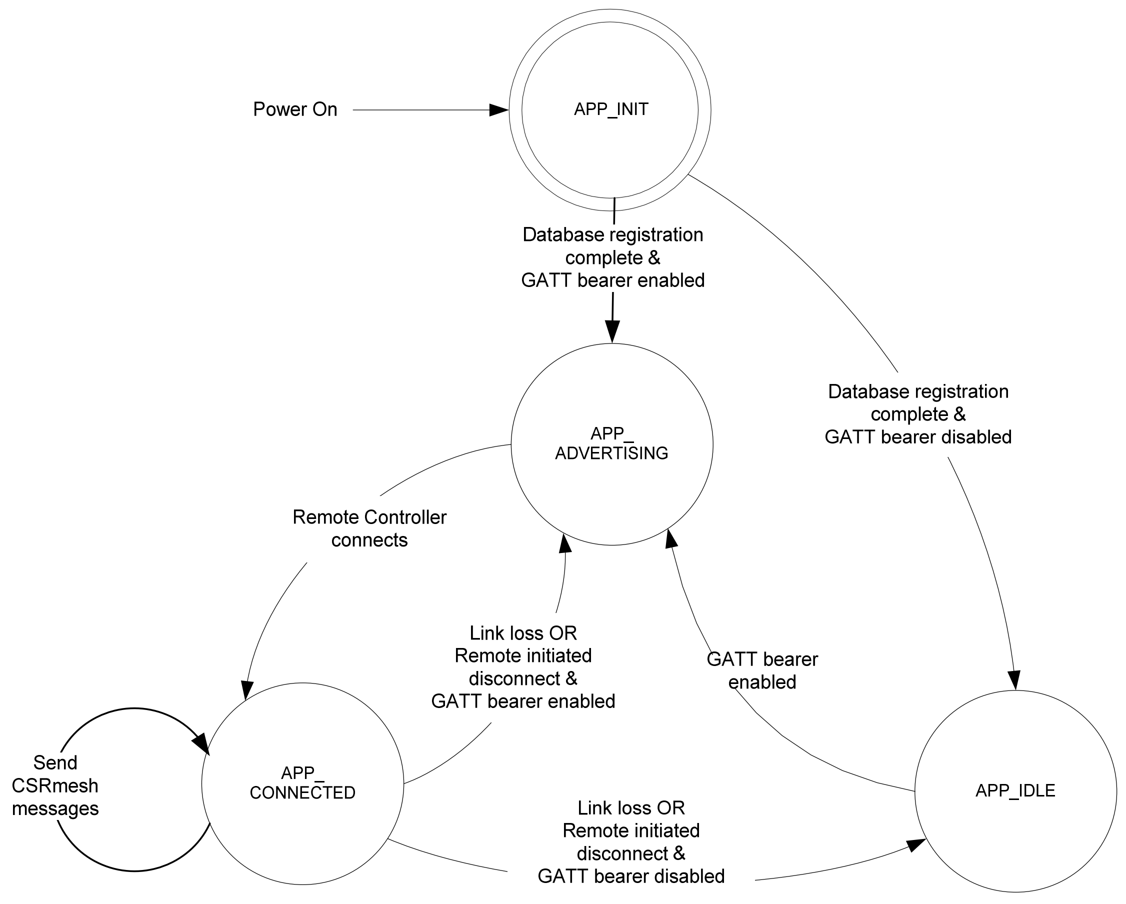

Power On
APP_INIT
Database registration complete & GATT bearer enabled
Database registration complete & GATT bearer disabled
APP_ADVERTISING
Remote Controller connects
Link loss OR Remote initiated disconnect & GATT bearer enabled
GATT bearer enabled
Send CSRmesh messages
APP_CONNECTED
Link loss OR Remote initiated disconnect & GATT bearer disabled
APP_IDLE

Figure 4.1: Internal GATT State Machine

4.2.1. app_state_init

When the application is powered on or the chip resets, it enters the `app_state_init` state. The application registers the service database with the firmware and waits for confirmation. On a successful database registration the application starts advertising if the GATT bearer is enabled. If GATT bearer is disabled, the application switches to the `app_state_idle` state.

4.2.2. app_state_advertising

The application starts in the `app_state_advertising` state and transmits connectable advertising events at an interval defined by `ADVERT_INTERVAL`. When a central device connects to it, the advertisements are stopped and the application enters the `app_state_connected` state. For more information about advertisement timers, see Section 6.1. If the GATT bearer is disabled in this state, the application switches to the `app_state_idle` state.

4.2.3. app_state_idle

The CSRmesh application enters the `app_state_idle` state when the GATT bearer is disabled and the application is not in the `app_state_connected` state.

4.2.4. app_state_connected

In the `app_state_connected` state, the CSRmesh application is connected to a CSRmesh Control device using connection intervals specified in Table 6.2. It can receive commands from the Control device or send responses received over CSRmesh to control the device.

If link loss occurs and the GATT bearer is enabled, the application switches to the `app_state_advertising` state, otherwise it switches to `app_state_idle` state.

In the case of a remote triggered disconnection, the application switches to the `app_state_advertising` state if the GATT bearer is enabled, otherwise it changes to the `app_state_idle` state.

4.2.5. app_state_disconnecting

The CSRmesh application never triggers a disconnection on its own:

- If a disconnection is triggered, the application enters the `app_state_advertising` state.

4.3. CSRmesh Association

The device needs to be associated with a CSRmesh network to communicate with other devices in the network. In CSRmesh 2.0, the application does not send device identification messages periodically by default. The UUID and AC of the device can be programmed to the device using the CSR uEnergy Production Test Tool. The CSRmeshQRCodeScanner application generates the QR code corresponding to the known UUID and Authorization Code, see Figure 4.3. For more information about device association, see *CSRmesh 2.0 Android Control Application Note* or *CSRmesh 2.0 iOS Control Application Note.*

Figure 4.2 shows the application association state machine.

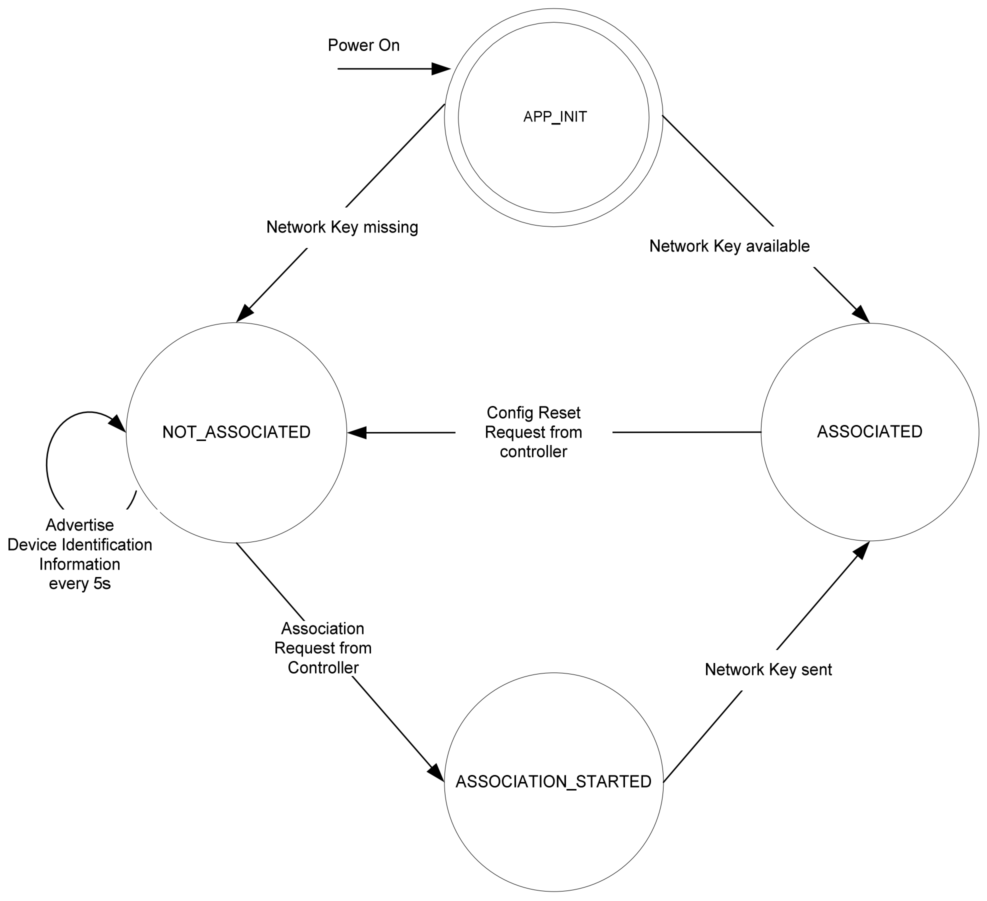
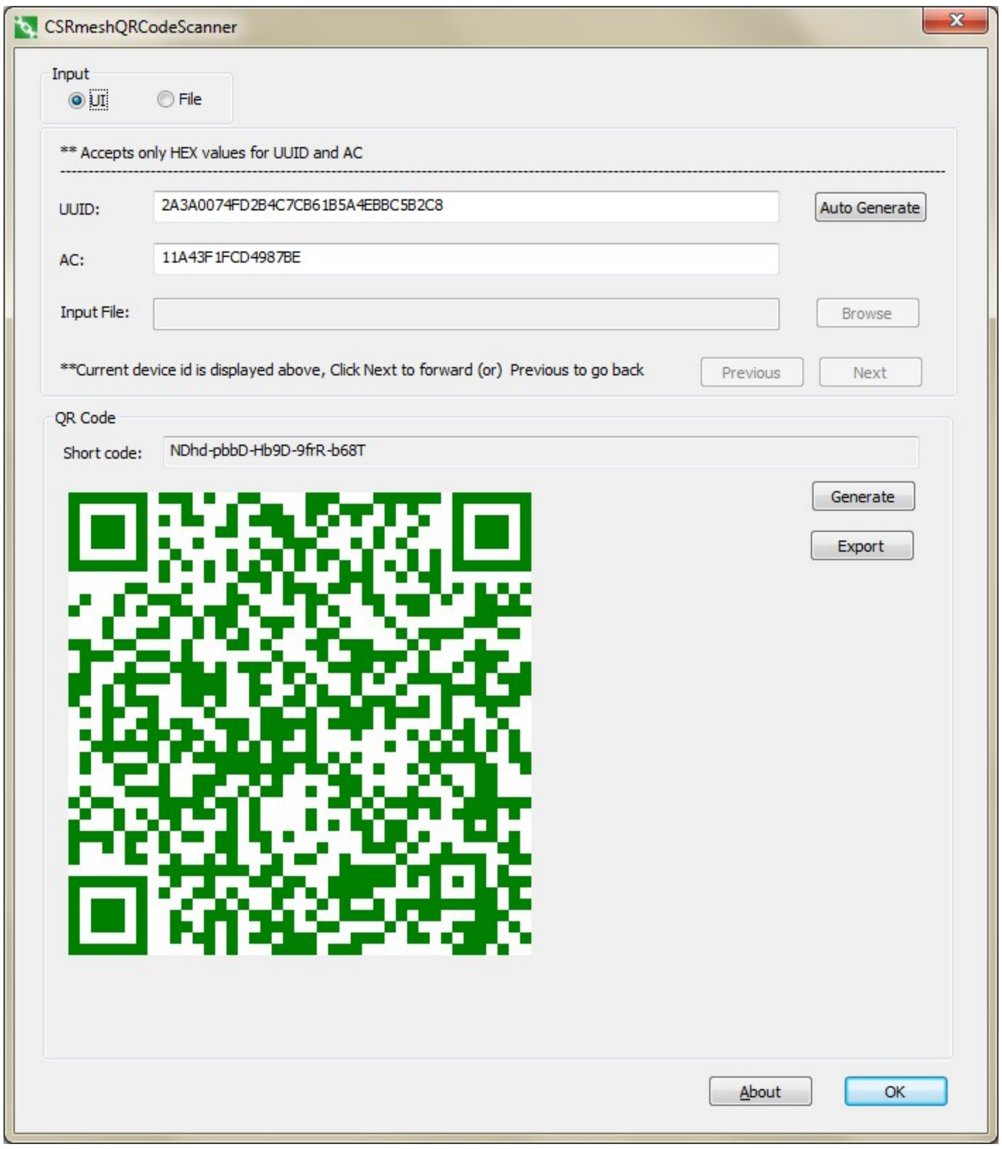

Power On
APP_INIT
Network Key missing
Network Key available
NOT_ASSOCIATED
Config Reset Request from controller
ASSOCIATED
Advertise Device Identification Information every 5s
Association Request from Controller
Network Key sent
ASSOCIATION_STARTED

Figure 4.2: CSRmesh Association State Machine



Input: [✓ UI] File

** Accepts only HEX values for UUID and AC

UUID: 2A3A0074FD2B4C7CB61B5A4EBBC5B2C8

AC: 11A43F1FCD4987BE

Input File:

**Current device id is displayed above, Click Next to forward (or) Previous to go back

Short code: NDhd-pbbD-Hb9D-9frR-b68T

Figure 4.3: CSRmesh QR Code Scanner Application

4.3.1. app_state_not_associated

The first time the application is flashed on the device, it is in the `app_state_not_associated` state. The application is ready to associate with a CSRmesh network.

4.3.2. app_state_association_started

The application enters `app_state_association_started` state when receiving an association request from the control device.

4.3.3. app_state_associated

The application enters the `app_state_associated` state when association completes. The application saves the association state on the NVM and continues to be associated even after power cycle. The application switches to `app_state_not_associated` when receiving a `CSR_MESH_CONFIG_RESET_DEVICE` or on a 2-second long press of the SW2 button.

4.4. CSRmesh Models Supported

Table 4.1 lists the CSRmesh models that the Light application supports.

CSRmesh Model	Application Action
Light Model	Sets the light colour and level using light set messages. Responds to get the state message with the current light state.
Power Model	Handles the power control messages. Responds to get the state with the current power state of the light.
Attention Model	Handles the `CSR_MESH_ATTENTION_SET_STATE_EVENT` command. When the attract attention is set, the application blinks red.
Battery Model	Upon receiving the `CSR_MESH_GET_BATTERY_STATE` message, the application reads the battery level and responds with the current battery level and state. The application is implemented with reference to the CSRmesh development boards that run on AA Batteries. So the application sets `BATTERY_MODEL_STATE_POWERING_DEVICE` indicating that the device is battery powered. The `BATTERY_MODEL_STATE_NEEDS_REPLACEMENT` bit is also set if the battery level is below the threshold.
Data Stream Model	Enables the application to send and receive a stream of bytes from another CSRmesh device.

Table 4.1: CSRmesh Models Supported

Note:

The CSRmesh Light application supports groups for Light, Power, Data Stream and Attention models. It statically allocates memory to store the assigned group IDs and saves them on the NVM.

4.5. Synchronising with CSRmesh Activity

The CSRmesh Light application can connect to the CSRmesh Control application in a bridge device role. The application must synchronise with the CSRmesh library to avoid collision of the advertisements and connection events with the CSRmesh activity. The application calls the CSRmesh APIs to synchronise connection radio events and connectable advertisements with the CSRmesh library.

4.5.1. Application Connectable Advertisements

The application sends connectable advertisements at regular intervals when the device is powered not connected. The interval at which the advertising events are sent is defined by `ADVERT_INTERVAL`.

The application calls `CSRSchedSendUserAdv()` rather than the firmware API to send connectable adverts. This function schedules an application advertising event when called.

4.5.2. Connection Events

The application must notify the CSRmesh stack of the GATT connection events to schedule the CSRmesh activity:

- The application calls `CSRSchedNotifyGattEvent(ucid, conn_interval)` to synchronise the CSRmesh library with the connection events. It is called in the following event handlers:
  - `LM_EV_CONNECTION_COMPLETE`
  - `LS_CONNECTION_PARAM_UPDATE_IND`
  - `LM_EV_DISCONNECT_COMPLETE`

4.6. Bearer State Management

The CSRmesh light application supports 2 bearers:

- GATT Bearer
- BLE Advertising Bearer

The application adopts the following policy with regard to the supported bearer state:

- Relay is always enabled on both bearers unless it is disabled by the `CSR_MESH_BEARER_STATE_EVENT` message or by the CS User Key configuration, see Section 6.8.
- When the device is not associated, the promiscuous mode is enabled on both bearers. This helps relaying the messages authenticated by the network key that are addressed to other associated devices.
- Both relay and promiscuous modes are enabled when the device is connected as a GATT bridge. This means any message sent from the control application over a GATT connection is relayed on the mesh promiscuously. This is useful because any device in the vicinity that supports bridge role, regardless of the association status and the network to which it is associated to, can be used as a bridge by the control device.
- The last configured bearer state is restored when the connection is terminated.

4.7. Application Data Stream Protocol

The application implements a protocol to transfer large blocks of data with another CSRmesh device on the network. Table 4.2 lists the message format that the application uses to exchange messages. Table 4.3 lists application data stream messages.

Message Code (1 octet)	Length (1 or 2 octets)	Data (0 – 32767 octets)
Identifies the type of the message. See Table 4.3 for supported messages.	Length of the data: ▪ If the MSB of the first octet is `0` then the length is a 7-bit value. ▪ If the MSB of the first octet is `1` then the length is a 15-bit value.	Data associated with the message.

Table 4.2: Application Data Stream Message Format

Message	Code	Length	Data	Description
`CSR_DEVICE_INFO_REQ`	`0x01`	`0`	None	Requests device information. Can be sent as a stream or a datagram message.
`CSR_DEVICE_INFO_RSP`	`0x02`	Variable (`1` to `32767`)	Device info	A text string containing information about the device. Is sent over a stream. The application sends a text string containing information about supported CSRmesh features.
`CSR_DEVICE_INFO_SET`	`0x03`	Variable (`1` to `32767`)	Device info	Text/binary data stored on the device as device info. The application stores the data associated with this message as the device information and responds with this data for any further device info requests.
`CSR_DEVICE_INFO_RESET`	`0x04`	`0`	None	Resets the device info. The application returns the default string to any further device info requests.
-	`0x05` to`0xFF`	-	-	Undefined

Table 4.3: Application Data Stream Messages

5. NVM Map

Table 5.1 lists the parameters that the application stores in the NVM to prevent loss in the event of powering off or a chip panic.

Entity Name	Type	Size of Entity (Words)	NVM Offset (Words)
CSRmesh Stack NVM	uint16 array	32	0
CSRmesh Device UUID (part of 31 words of Stack NVM)	uint16 array	8	1
CSRmesh Device Authorisation Code (part of 31 words of Stack NVM)	uint16 array	4	9
Sanity Word	uint16	1	32
Application NVM Version	uint16	1	33
CSRmesh Association State	boolean	1	34
Light R G B and Power values	structure	2	35
Light Model Group IDs	uint16 array	4	37
Power Model Group IDs	uint16 array	4	41
Attention Model Group IDs	uint16 array	4	45
Data Model Group IDs	uint16 array	4	49

Table 5.1: NVM Map for Application

Entity Name	Type	Size of Entity (Words)	NVM Offset (Words)
GAP Device Name Length	uint16	1	53
GAP Device Name	uint8 array	20	54

Table 5.2: NVM Map for GAP Service

Note:

The application does not pack data before writing it to the NVM. This means that writing a `uint8` takes one word of NVM memory.

5.1. Application NVM Version

Before reading the parameters in Table 5.1 and Table 5.2 from the NVM, the application reads the NVM version at word offset `1`. In the case of an application update when the version number stored on the NVM does not match the defined `APP_NVM_VERSION`, the application re-initialises the NVM starting from Light Model Group Id's and stores the defined `APP_NVM_VERSION` at word offset `1`.

Note:

The Device UUID, Authorisation Code, CSRmesh library data and association information are not reset, ensuring that the contents of the NVM are valid for the NVM offsets defined in the application. In case the NVM offsets of the parameters change on an application update, this value can be incremented to invalidate the contents of the NVM.

6. Customising the Application

This section describes how to customise some parameters of the CSRmesh applications.

The developer can customise the application by modifying the following parameter values.

6.1. Advertisement Timers

The CSRmesh Light application sends connectable advertisements when powered on and not connected. The application uses a timer to send a connectable advertising event at regular intervals. The advertising interval is defined in the `csr_mesh_light_gatt.h` file.

Timer Name	Timer Value
`ADVERT_INTERVAL`	1250 ms ± (0~10) ms

Table 6.1: Advertisement Timers

6.2. Connection Parameters

The CSRmesh Light application uses the connection parameters in Table 6.2 by default. The macros for these values are defined in the `gap_conn_params.h` file. These values are chosen by considering the overall current consumption of the device and optimum performance of the device in the CSRmesh network. CSR recommends not modifying these parameters. If the connection interval is set to less than 13 ms, the device is not able to scan CSRmesh messages from the associated network but only messages received from the GATT connection. See *Bluetooth Core Specification Version 4.1* for the connection parameter range.

Parameter Name	Parameter Value
Minimum Connection Interval	90 ms
Maximum Connection Interval	120 ms
Slave Latency	0 intervals
Supervision Timeout	6000 ms

Table 6.2: Connection Parameters

6.3. Number of Supported Model Groups

The CSRmesh application supports Group ID assignment for Light, Power, Stream and Attention models of the CSRmesh. The maximum number of model groups supported by default is set to `4` in the `user_config.h`.

Parameter Name	Parameter value
`MAX_MODEL_GROUPS`	`4`

Table 6.3: Number of Supported Model Groups

Note:

Each group supported per model requires one word in the RAM and one word in the NVM. The maximum number of groups supported is limited by the available RAM and NVM space.

6.4. Device Name

You can change the device name for the application. The default name is `CSRmesh` in the file `gap_service.c`. The maximum length of the device name is 20 octets.

6.5. Device Address

The application uses a public address by default. The `USE_STATIC_RANDOM_ADDRESS` macro in the `user_config.h` file enables the support for static random addresses.

6.6. Non-Volatile Memory

The application uses one of the following macros to store and retrieve persistent data in either the EEPROM or Flash-based memory:

- `NVM_TYPE_EEPROM` for I^{2}C EEPROM
- `NVM_TYPE_FLASH` for SPI Flash

Note:

The macros are enabled by selecting the NVM type using the Project Properties in xIDE. The macro is defined during compilation to let the application know for which NVM type it builds. If EEPROM is selected, `NVM_TYPE_EEPROM` is defined; If SPI Flash is selected, `NVM_TYPE_FLASH` is defined. Follow the comments in the `.keyr` file.

6.7. Application Features

Table 6.4 lists how to customise some parameters on the CSRmesh Light application. The application can be configured by uncommenting the required definition in `user_config.h`.

Configuration	Description
`USE_AUTHORISATION_CODE`	Enforces Authorisation code checking on a device during association. If this feature is enabled, the Associating control device must have the same authorisation code to associate the device with the network.
`ENABLE_DEVICE_UUID_ADVERTS`	If this feature is enabled, the application sends UUID adverts periodically with an interval defined in `DEVICE_ID_ADVERT_TIME`. Otherwise the device UUID adverts are not sent. Disabled by default.
`ENABLE_DATA_MODEL`	Enables the data stream model to send/receive streams of octets to/from another CSRmesh device. If this feature is enabled, the application supports streaming of the device information string over the data model. The application uses a simple protocol to send and receive messages. See Section 4.7 for more information.
`COLOUR_TEMP_ENABLED`	Enables support for colour temperature setting.
`DEBUG_ENABLE`	Enables application debug logging on UART. The debug messages can be viewed on a HyperTerminal or any other terminal application by connecting the device to the PC and opening the corresponding COM Port with a 2400-8-N-1 configuration.
`LONG_KEYPRESS_TIME`	Configures the time for which the SW2/SW3 buttons should be held pressed to remove the network association locally. Is used only if `USE_ASSOCIATION_REMOVAL_KEY` is defined.
`USE_ASSOCIATION_REMOVAL_KEY`	Enables removing of the association of a light with a network locally. If this feature is enabled, a 2-second long press of the SW2 button removes the association with the network. In this mode UART is disabled. CSR recommends not defining `DEBUG_ENABLE` when this feature is enabled.

Configuration	Description
`ENABLE_FAST_PWM`	If this feature is enabled, the RGB LEDs are controlled by the PWM implemented on the 8051 PIO controller for faster control. If this feature is disabled, the PWM is controlled using the hardware PWM. It is enabled by default in `user_config.h`
`USE_STATIC_RANDOM_ADDRESS`	If this features is enabled, the application uses a static random address to send connectable adverts.

Table 6.4: Application Configuration

6.8. Configuring the CSRmesh Parameters

The CS user keys can be defined in the `.keyr` file to override the default CSRmesh parameters. The CSRmesh library sets the parameters based on the CS user key values. Table 6.5 lists the recommended values for optimal performance of the devices over the CSRmesh network.

CS User Key Index	Parameter	Recommended Value	Description
0	CSRmesh Configuration Bitmask	`0000` to `0007`	Bit 0: CSRmesh Relay Enable `1`: Enables relay of CSRmesh messages. `0`: Disables relay of CSRmesh messages. Bit 1: CSRmesh Bridge Enable `0`: Disables connectable advertisements. `1`: Enables connectable advertisements. Disabling bridge leaves the device non connectable on any service. The connectable adverts can be enabled by sending a `CSR_MESH_BEARER_SET_STATE` message with the LE GATT server bearer bit set. Bit 2: CSRmesh Random Device UUID Enable `1`: Enables generation of random device UUID. `0`: Reads the UUID from NVM. If this bit is set to `1`, the application generates a random UUID and stores it in the NVM when running for the first time. This can be used to avoid having the same UUIDs on multiple devices without explicitly programming a UUID on each device. Bits 3 to 15 are ignored.

Table 6.5: Configuring CSRmesh Parameters

Appendix A CSRmesh Application GATT Database

A.1 GATT Service Characteristics

Characteristic Name	Database Handle	Access Permissions	Managed By	Security Permissions	Value
Service Changed	`0x0003`	Indicate	Application	Security Mode 1 and Security Level 1	Service Changed handle value
Service Changed Client Characteristic Configuration Descriptor	`0x0004`	Read Write	Application	Security Mode 1 and Security Level 1	Current client configuration for Service Changed characteristic

Table A.1: GATT Service Characteristics

A.2 GAP Service Characteristics

Characteristic Name	Database Handle	Access Permissions	Managed By	Security Permissions	Value
Device Name	`0x0007`	Read Write	Application	Security Mode 1 and Security Level 1	Device name Default value : `CSRmesh`
Appearance	`0x0009`	Read	Firmware	Security Mode 1 and Security Level 1	Unknown: `0x0000`
Peripheral Preferred Connection Parameters	`0x000b`	Read	Firmware	Security Mode 1 and Security Level 1	Connection interval: Minimum: 90 ms Maximum: 120 ms Slave latency: 0 Connection timeout: 6 s

Table A.2: GAP Service Characteristics

For more information about GAP service and security permissions, see *Bluetooth Core Specification Version 4.1*.

A.3 CSR OTA Update Application Service Characteristics

Characteristic Name	Database Handle	Access Permissions	Managed By	Security Permissions	Value
Current Application	`0x000e`	Read Write	Application	Security Mode 1 and Security Level 2	Current live application `0x0`: OTA Update Bootloader `0x1`: Identifies application 1 `0x2`: Identifies application 2
Read CS Block	`0x0010`	Write	Application	Security Mode 1 and Security Level 2	Format: uint16[2] Index 0: An offset in 16-bit words in the CS defined in the SDK documentation. Index 1: Size of the CS block expected in octets.
Data Transfer	`0x0012`	Read Notify	Application	Security Mode 1 and Security Level 2	This characteristic is ATT_MTU-3 (20)-bytes long. The format of the 20-bytes is defined by the message context.
Data Transfer Client Characteristic Configuration	`0x0013`	Read Write	Application	Security Mode 1 and Security Level 2	Current client configuration for Data Transfer characteristic
Version	`0x0015`	Read	Firmware	Security Mode 1 and Security Level 2	Service version Format: uint8

Table A.3: CSR OTA Update Application Service Characteristics

A.4 Mesh Control Service Characteristics

Characteristic Name	Database Handle	Access Permissions	Managed By	Security Permissions	Value
Network Key	`0x0018`	Write	Application	Security Mode 1 and Security Level 1	0
Device UUID	`0x001a`	Read	Application	Security Mode 1 and Security Level 1	`22e4-b12c-5042-11e3-9618-ce3f-5508-acd9`
Device ID	`0x001c`	Read Write	Application	Security Mode 1 and Security Level 1	`0x8001`
MTL Continuation Control Point	`0x001e`	Write	Application	Security Mode 1 and Security Level 1	Dynamic
MTL Continuation Control Point Client Characteristic Configuration	`0x001f`	Read Write	Application	Security Mode 1 and Security Level 1	Current client configuration for MTL Continuation Control Point characteristic
MTL Complete Control Point	`0x0021`	Write Notify	Application	Security Mode 1 and Security Level 1	Dynamic
MTL Complete Control Point Client Characteristic Configuration	`0x0022`	Read Write	Application	Security Mode 1 and Security Level 1	Current client configuration for MTL Complete Control Point characteristic
MTL TTL	`0x0024`	Read Write	Application	Security Mode 1 and Security Level 1	50
MESH Appearance	`0x0026`	Read Write	Application	Security Mode 1 and Security Level 1	0

Table A.4: Mesh Control Service Characteristics

Document References

Document	Reference
Bluetooth Core Specification Version 4.1	www.bluetooth.org/
CSR µEnergy Modifying an Application to Support OTA Update Application Note	CS-304564-AN
CSR µEnergy Over-the-Air (OTA) Update System Application Note	CS-316019-AN
CSR µEnergy xIDE User Guide	CS-212742-UG
CSRmesh 2.0 Android Controller Application Note	CS-337680-AN
CSRmesh 2.0 Gateway SB User Guide	CS-332701-UG
CSRmesh 2.0 iOS Controller Application Note	CS-337682-AN
CSRmesh 2.0 Mobile Application User Guide	CS-337051-UG
CSRmesh 2.0 Production Test Tool User Guide	CS-335123-UG
CSRmesh 2.0 Node API Guide	www.csrsupport.com
CSRmesh 2.0 Node Release Note	CS-339050-RN
CSRmesh Application 1.x to 2.0 Porting Guide	CS-335300-DC
GATT Database Generator	CS-219225-UG
Installing the CSR Driver for the Profile Demonstrator Application	CS-235358-UG
Over-the-Air Update Application and Bootloader Services Specification	CS-316220-SP
Service Characteristics And Descriptions	developer.bluetooth.org

Terms and Definitions

API	Application Programmer's Interface
BLE	Bluetooth Low Energy (now known as Bluetooth Smart)
Bluetooth®	Set of technologies providing audio and data transfer over short-range radio connections
CS	Configuration Store
CSR	Cambridge Silicon Radio
CSRmesh™	A CSR protocol that enables peer-to-peer-like networking of Bluetooth Smart devices
EEPROM	Electrically Erasable Programmable Read Only Memory
GAP	Generic Access Profile
GATT	Generic Attribute Profile
I^2C	Inter-Integrated Circuit
IoT	Internet of Things
IRK	Identity Resolving Key
LED	Light Emitting Diode
LM	Link Manager
MTL	Message Transport Layer
NVM	Non Volatile Memory
OTA	Over The Air
PC	Personal Computer
PCB	Printed Circuit Board
PIO	Programmable Input Output
PWM	Pulse Width Modulation
Rx	Receive
SDK	Software Development Kit
SPI	Serial Peripheral Interface
Tx	Transmit
USB	Universal Serial Bus
UUID	Universally Unique Identifier