Qualcomm Technologies International, Ltd.

CSRmesh 2.1

Home Application Notes

CS-348212-AN

October 28, 2016

Revision history

Revision	Date	Description
A	OCT 2016	Initial release of this document.

Contents

Figures

Tables

1 Introduction

This document describes the CSRmesh applications that can be built using the µEnergy SDK.

The main example application is the Light Application (see section 7). Other example applications are described in Appendix B.

The features common to all of these applications are described in the following chapters of this document. The applications themselves and their unique features are described in their relevant sections.

CSRmesh applications are part of the CSRmesh release and show the custom-defined mesh Control Service in GATT server role. They use the CSRmesh library provided as part of the CSRmesh release. For more information, refer to the *CSRmesh API Guide* documentation.

NOTE: CSRmesh applications do **not** support bonding and are **not** compatible with previous CSRmesh releases.

1.1 Applications overview

The CSRmesh applications use the CSRmesh library API to communicate with other associated devices in the same CSRmesh network. They also support a custom GATT profile, which is embedded in the library to provide an additional bearer for CSRmesh services. This allows devices that are not able to issue non-connectable adverts to communicate with the mesh through a connection.

1.1.1 Supported Profiles

CSRmesh applications implement the following custom profiles to support the different use cases.

CSRmesh Control Profile

The CSRmesh Control Profile defines the behavior when:

- A network of devices (such as lights and switches, temperature sensors and heaters, and so on) need to be created.
- Controlling the device after a network is created; for example, switching lights on/off or controlling heaters by sending the desired and current air temperature values.
- Reading the status of a device in the network; for example, the on/off state and color or intensity of a light.

Table 1-1 lists the two roles that the CSRmesh Control Profile defines.

Table 1-1 CSRmesh Control Profile roles

Role	Description	Implementation
CSRmesh Device	Receives commands from host and sends them over the CSRmesh network to associated devices. Receives responses from associated devices over the CSRmesh and forwards them to the host over a Bluetooth Smart connection.	On the CSRmesh 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

1.1.2 Application topology

Table 1-2 lists the topology that the CSRmesh application uses. Table 1-3 lists the roles and responsibilities.

Table 1-2 Application topology

Role	Mesh Control Service	GAP Service	GATT Service	CSR GAIA Service for OTA Update supported on CSR 102x platform	CSR GAIA Service for OTA Update over Relay supported on CSR 102x platform	CSR OTA Update Application Service supported on CSR 101x platform
GATT Role	GATT Server	GATT Server	GATT Server	GATT Server	GATT Client	GATT Server
GAP Role	Peripheral	Peripheral	Peripheral	Peripheral	Central	Peripheral

Table 1-3 Role and responsibilities

Role	Responsibility
GATT Server	Accepts incoming commands and requests from a client and sends responses, indications and notifications to the client.
GATT Client	Scans for advertisements and initiates connection requests to the server. (This functionality is supported by the node application only when OTA is supported over LOT Relay)
GAP Peripheral	Accepts connections from the remote device and acts as a slave in the connection.
GAP Central	Initiates a connection towards the peer device and acts as a Master device in the connection. (This functionality is supported by the Light application only when OTA is supported over LOT Relay).

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

1.1.3 Services Supported in GATT Server Role

The applications use the following services:

- Mesh Control Service v2.0
- GATT Service

- GAP Service
- CSR OTA Update Application Service v7 (Supported only on CSR101x platform)
- Generic Application Interface Architecture (GAIA) Service (Supported only on CSR102x platform)

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

Figure 1-1 shows the services supported in the GATT Server Role.

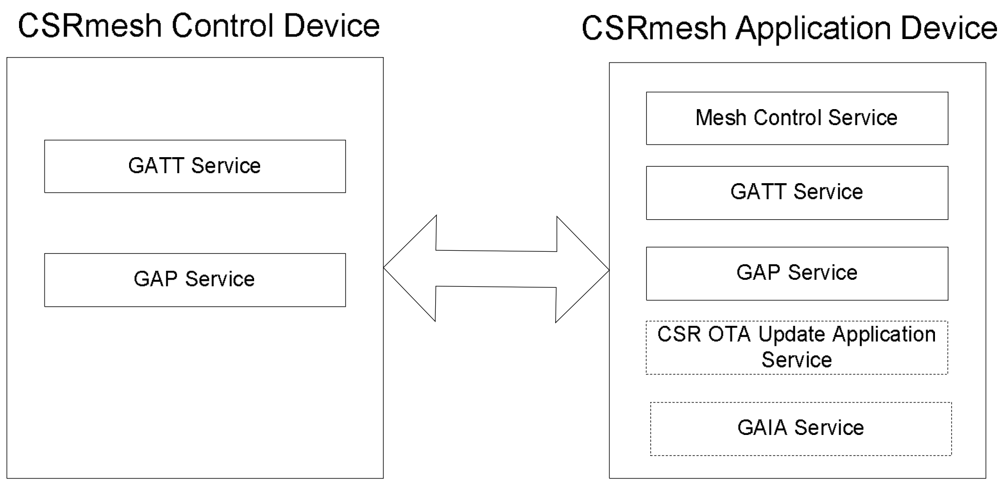


Figure 1-1 Primary services

OTA Update Application Service

The 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 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 µEnergy Over-the-Air Updater host application included in the SDK can update the device.

For more information about OTA Update, refer to the:

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

For information about OTA Update applications for iOS and Android, refer to www.csrsupport.com

GAIA Over-the-Air upgrade

The VM Upgrade Protocol over GAIA service enables wireless upgrade 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 GAIA OTA upgrade, the application needs to:

- Configure GAIA Service: GAIA service is enabled in the application by default. The user can disable GAIA service by removing the macro `GAIA_SUPPORT` from **Define Macros** in **Project Properties**.
- Configure GAIA OTA upgrade support: GAIA OTA Upgrade support is enabled in the application by default. The user can disable GAIA OTA Upgrade support by removing the macro `GAIA_OTAU_SUPPORT` from **Define Macros** in **Project Properties**.

The user can further specify the appearance and version of the new image to be downloaded using `mesh.upd` file provided in the project workspace, see section 6.10. When the device is enabled for GAIA OTAu, the µEnergy Over-the-Air Upgrader host application included in the SDK can be used to update the device. When the device gets updated, it can further relay the new image using Large Object Transfer (LOT) Model. See Section 4.8 for more information on this. Mesh is temporarily stopped when GAIA OTA upgrade is taking place. Mesh is restarted once the update gets over.

For information on GAIA OTA upgrade applications for iOS, see www.csrsupport.com.

2 Using the applications

This section describes how to use CSRmesh applications with CSRmesh Android and iOS applications to control devices.

2.1 Demonstration Kit

Table 2-1 lists the components that an application can use for demonstration.

Table 2-1 CSRmesh components

Component	Hardware	Application
Light Device	CSRmesh Development PCB (DB-CSR1010-10185-1A)	CSRmesh Application v2.1
Light Device	CSRmesh Development PCB (DK-CSR1025-10280-1A)	CSRmesh Application v2.1
CSRmesh Android Control Device	Android Bluetooth LE Device	CSRmesh Android Application v2.1
CSRmesh iOS Control Device	iOS Bluetooth LE Device	CSRmesh iOS Application v2.1

2.1.1 CSR101x CSRmesh development board

The µEnergy SDK is used to download the CSRmesh application on the development boards. Refer to the *µEnergy xIDE User Guide* for further information.

Ensure that the development board is switched on using the Power On/Off switch. Figure 2-3 shows the switch in the Off position.

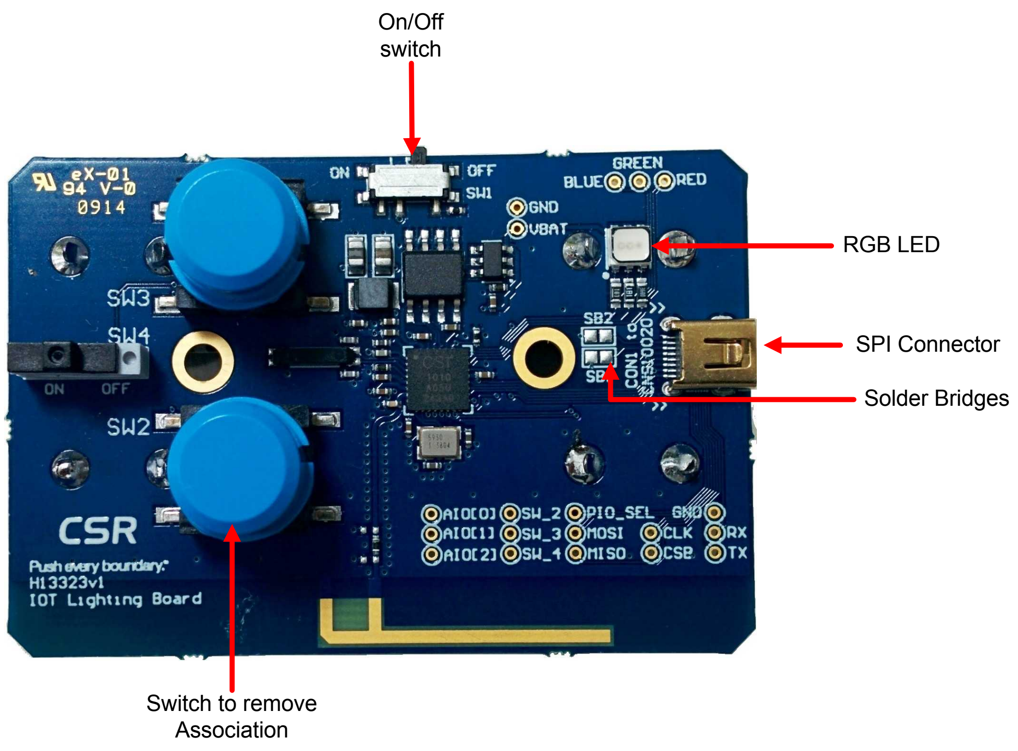


Figure 2-1 CSRmesh 101x development board

NOTE:

- When the development board is disconnected from the USB to SPI Adapter, wait at least on**e 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 USB to SPI Adapter. 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. QTIL recommends not pressing these buttons during UART communication.

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
- UUID :CSRmesh Device UUID
- AC: CSRmesh Authorization Code
- QR-Code : Encodes UUID and AC

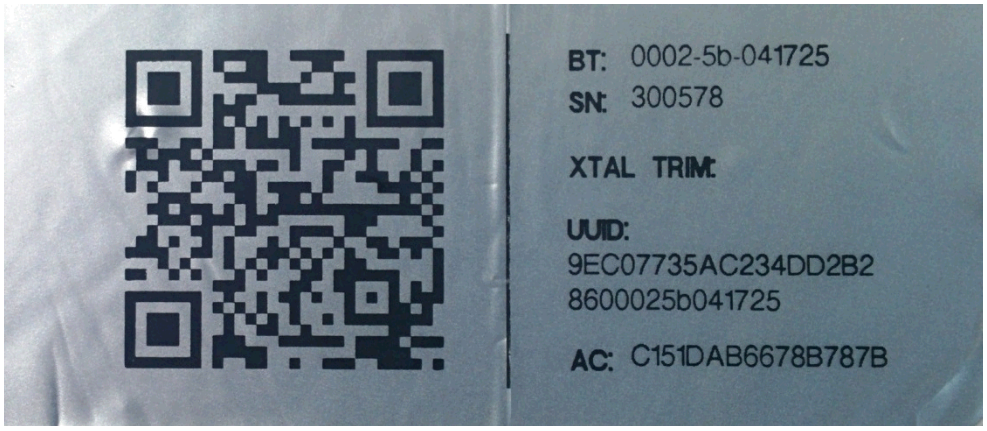


Figure 2-2 CSRmesh device tag sticker

The device can be programmed with the Bluetooth address and the XTAL Trim printed on the sticker by setting these values in the `bridge_CSR101x_A05.keyr` file.

The Device UUID and the Authorization Code printed on the sticker can be programmed on the NVM at the offsets defined in

Table 5 using the USB to SPI adapter.

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

1. Open a command prompt.
2. Program the device UUID and Authorization 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 Authorization Code to the device as follows:

     ```
     <CSR_uEnergy_Tools path>\uEnergyProdTest.exe -k
     CSRmeshbridge_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 the `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 Authorization 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 effective address is `0x4102`.

The CSRmesh Control application reads the device Authorization Code and the UUID from the QR-Code printed on the sticker during association.

NOTE: This is an example application. Refer to the *CSRmesh 2.0 Android Control Application Note* or the *CSRmesh 2.0 iOS Control Application Note* for details. For further information about programming the NVM, refer to the CSRmesh 2.0 Production Test Tool User Guide.

User interface

The application uses the buttons available on the CSRmesh development board. Table 2-2 lists the user interfaces of the CSRmesh development board.

Table 2-2 CSRmesh development board user interface

User Interface Component	Function
Switch SW1	Power slider switch powering on/off the board.
Button SW2	Holding this button for more than two seconds removes the network association. This behaviour can be enabled or disabled by user configuration.
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.

2.1.2 CSR102x CSRmesh Development Board

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

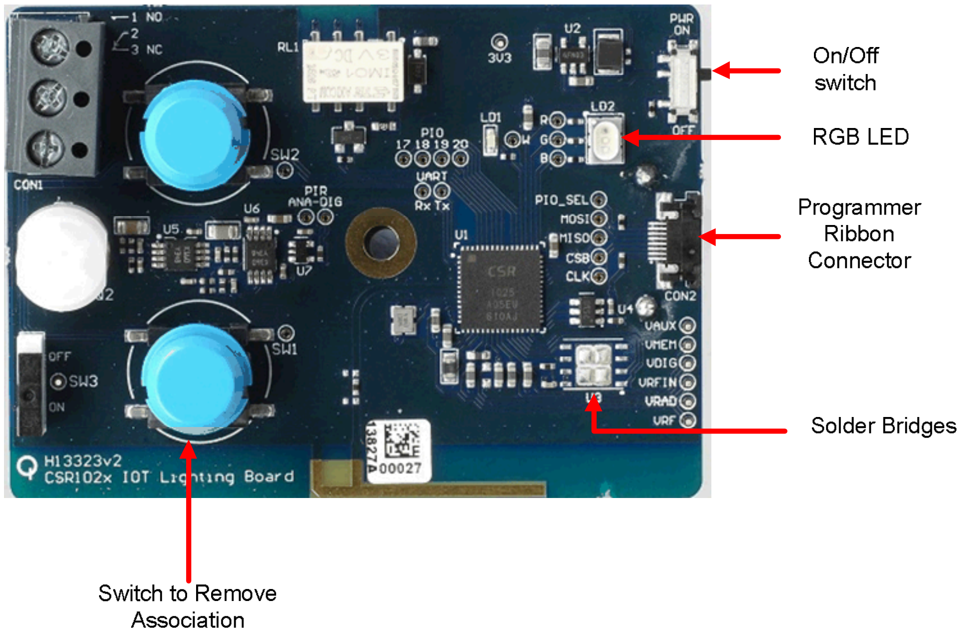


Figure 2-3 CSRmesh 102x development board

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

Table 2-3 lists the user interfaces of the CSRmesh development board.

Table 2-3 CSRmesh Development Board User Interface

User Interface Component	Function
Switch	Power slider switch powering on/off the board.
Button SW1	Holding this button for more than 2 seconds removes the network association. This behaviour can be enabled or disabled by user configuration.
Button SW2	Holding this button for more than 2 seconds removes the network association. This behaviour can be enabled or disabled by user configuration.
Switch SW3	Unused
RGB LED	Blinks green on attracting attention before association. 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.

2.1.3 CSRmesh Control Application

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

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

NOTE: This is an example application.

- For details about using the CSRmesh Control application on an Android device, refer to the *CSRmesh 2.1 Android Control Application Note.*
- For details about using the CSRmesh Control Application on an iOS device, refer to the *CSRmesh 2.1 iOS Control Application Note*.
- For details about the supported iOS and Android version, refer to the *CSRmesh 2.1 Mobile Applications Release Note*.

3 Application structure

This section describes the common source files, head files and database files to CSRmesh applications. Unique files for each application are described in the relevant application section.

3.1.1 Common source files

Table 3-1 lists the source files common to all CSRmesh applications.

Table 3-1 Common source files

File Name	Description
`..\mesh_common\mesh\handlers\advertisement\advertisement_handler.c`	Implements routines for triggering advertisement procedures.
`..\mesh_common\mesh\handlers\connection\connection_handler.c`	Defines the handler functions for handling the connection manager events.
`..\mesh_common\mesh\handlers\core_mesh\core_mesh_handler.c`	Defines the handler functions for CSRmesh events.
`..\mesh_common\mesh\handlers\data_model\data_model_handler.c`	Handles data stream model events. Implements a protocol to exchange large blocks of data with other CSRmesh devices. See Section 4.6 for more information.
`..\mesh_common\mesh\handlers\light_model\light_model_handler.c`	Implements the routines to initialise and handle the light model events.
`..\mesh_common\mesh\handlers\power_model\power_model_handler.c`	Implements the routines to initialise and handle the power model events.
`..\mesh_common\mesh\handlers\battery_model\battery_model_handler.c`	Implements the routines to initialise and handle the battery model events.
`..\mesh_common\mesh\handlers\attention_model\attention_model_handler.c`	Implements the routines to initialise and handle the attention model events.
`..\mesh_common\mesh\handlers\largeobjecttransfer_model\largeobjecttransfer_model_handler.c`	Implements the routines to initialise and handle the large object transfer model events.
`..\mesh_common\mesh\handlers\action_model\action_model_handler.c`	Implements the routines to initialise and handle the action model events.
`..\mesh_common\mesh\handlers\actuator_model\actuator_model_handler.c`	Implements the routines to initialise and handle the actuator model events.
`..\mesh_common\mesh\handlers\asset_model\asset_model_handler.c`	Implements the routines to initialise and handle the asset model events.
`..\mesh_common\mesh\handlers\beacon_model\beacon_model_handler.c`	Implements the routines to initialise and handle the beacon model events.
`..\mesh_common\mesh\handlers\beacon__proxymodel\beacon_proxy_model_handler.c`	Implements the routines to initialise and handle the beacon proxy model events.

File Name	Description
`..\mesh_common\mesh\handlers\diagnostic_model\diagnostic_model_handler.c`	Implements the routines to initialise and handle the diagnostic model events.
`..\mesh_common\mesh\handlers\extension_model\extension_model_handler.c`	Implements the routines to initialise and handle the extension model events.
`..\mesh_common\mesh\handlers\firmware_model\firmware_model_handler.c`	Implements the routines to initialise and handle the firmware model events.
`..\mesh_common\mesh\handlers\otau\otau_handler.c`	Implements the routines to work for the OTAU.
`..\mesh_common\mesh\handlers\scan\scan_handler.c`	Implements the routines for the scan functionality when the device is in central role used while updating the OTA over relay.
`..\mesh_common\mesh\handlers\sensor_model\sensor_model_handler.c`	Implements the routines to initialise and handle the sensor model events.
`..\mesh_common\mesh\handlers\time_model\time_model_handler.c`	Implements the routines to initialise and handle the time model events.
`..\mesh_common\mesh\handlers\tracker_model\tracker_model_handler.c`	Implements the routines to initialise and handle the tracker model events.
`..\mesh_common\mesh\handlers\watchdog_model\watchdog_model_handler.c`	Implements the routines to initialise and handle the watchdog model events.
`..\mesh_common\mesh\handlers\common\app_debug.c`	Implements the routines for debug utility functions.
`..\mesh_common\mesh\handlers\common\app_util.c`	Implements the routines for utility functions.
`..\mesh_common\mesh\drivers\ battery_hw.c`	Implements the routines to read battery level.
`..\mesh_common\mesh\drivers\ iot_hw.c`	Implements the hardware interface to configure and control the peripherals on the CSRmesh development board.
`..\mesh_common\mesh\drivers\csr_mesh_ps_ifc.c`	Implements the routines for reading and writing the core mesh stack NVM.
`..\mesh_common\mesh\drivers\fast_pwm.c`	Implements the routines for the fast pwm module..
`..\mesh_common\mesh\drivers\stts751_temperature_sensor_hw.c`	Implements the driver to read the temperature from the stts751 temperature sensor.
`..\mesh_common\components\connection_manager\cm_common.c`	Implements the common code of the connection manager.
`..\mesh_common\components\connection_manager\cm_hal.c`	Implements the hardware abstraction layer code in the connection manager.
`..\mesh_common\components\connection_manager\cm_observer.c`	Implements the LE observer functionality in the connection manager.
`..\mesh_common\components\connection_manager\cm_peripheral.c`	Implements the LE peripheral functionality in the connection manager.
`..\mesh_common\components\connection_manager\cm_private.c`	Implements the private code of the connection manager.
`..\mesh_common\components\connection_manager\cm_security.c`	Implements the LE Security functionality in the connection manager.

File Name	Description
`..\mesh_common\components\connection_manager\cm_server.c`	Implements the LE Server functionality in the connection manager.
`..\mesh_common\components\connection_manager\cm_central.c`	Implements the LE Central functionality in the connection manager.
`..\mesh_common\components\connection_manager\cm_client.c`	Implements the LE Client functionality in the connection manager.
`..\mesh_common\components\nvm_manager\nvm_access.c`	Implements the NVM read/write routines.
`..\mesh_common\peripheral\conn_param_update.c`	Implements the functions responsible for the connection parameter update on the peripheral device.
`..\mesh_common\server\gap\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.
`..\mesh_common\server\gatt\gatt_service.c`	Defines routines for using GATT Service.
`..\mesh_common\server\mesh_control\mesh_control.c`	Implements routines for handling read/write access on Mesh Control characteristics and for sending notifications of mesh device responses.
`..\mesh_common\server\gaia\byte_utils.c`	Defines the utilities used for frame parsing and creation for GAIA
`..\mesh_common\server\gaia\gaia.c`	Implements the GAIA protocol for a server connection
`..\mesh_common\server\gaia\gaia_otau.c`	Implements the GAIA OTA Upgrade protocol for server side
`..\mesh_common\server\gaia\gaia_service.c`	Implements the routines for GAIA server service
`..\mesh_common\client\gaia\gaia_client.c`	Implements the GAIA protocol for a client connection
`..\mesh_common\client\gaia\gaia_client_service.c`	Implements the routines for GAIA client service
`..\mesh_common\client\gaia\gaia_otau_client.c`	Implements the GAIA OTA Upgrade protocol for client side

As well as the common source files, each application has its own unique source files. Please refer to the appropriate table for the application source files you require:

- Table 7-2 Light application source files
- Table B-3 Bridge application source files
- Table B-7 Heater application source files
- Table B-10 Switch application source files
- Table B-15 Temperature Sensor application source files

3.1.2 Common header Files

Table 3-2 lists the common header files.

Table 3-2 Common header files

`..\mesh_common\mesh\handlers\connection\connection_handler.h`	Contains prototypes of the externally referenced functions defined in `connection_handler.c`.
`..\mesh_common\mesh\handlers\core_mesh\core_mesh_handler.h`	Contains prototypes of the externally referenced functions defined in `core_mesh_handler.c`.
`..\mesh_common\mesh\handlers\advertisement\advertisement_handler.h`	Contains prototypes for the functions defined in `advertisement_handler.h` file.
`..\mesh_common\mesh\handlers\data_model\data_model_handler.h`	Contains enumerations and function prototypes of the externally referenced functions defined in `data_model_handler.c`.
`..\mesh_common\mesh\handlers\light_model\light_model_handler.h`	Contains function prototypes of the externally referenced functions defined in `light_model_handler.c`.
`..\mesh_common\mesh\handlers\power_model\power_model_handler.h`	Contains function prototypes of the externally referenced functions defined in `power_model_handler.c`.
`..\mesh_common\mesh\handlers\attention_model\attention_model_handler.h`	Contains function prototypes of the externally referenced functions defined in `attention_model_handler.c`.
`..\mesh_common\mesh\handlers\battery_model\battery_model_handler.h`	Contains function prototypes of the externally referenced functions defined in `battery_model_handler.c`.
`..\mesh_common\mesh\handlers\largeobjecttransfer_model\largeobjecttransfer_model_handler.h`	Contains function prototypes of the externally referenced functions defined in `largeobjecttransfer_model_handler.c`.
`..\mesh_common\mesh\handlers\action_model\action_model_handler.h`	Contains enumerations and function prototypes of the externally referenced functions defined in `action_model_handler.c`.
`..\mesh_common\mesh\handlers\asset_model\asset_model_handler.h`	Contains function prototypes of the externally referenced functions defined in `asset_model_handler.c`.
`..\mesh_common\mesh\handlers\beacon_model\beacon_model_handler.h`	Contains function prototypes of the externally referenced functions defined in `beacon_model_handler.c`.
`..\mesh_common\mesh\handlers\beacon_proxy_model\beacon_proxy_model_handler.h`	Contains function prototypes of the externally referenced functions defined in `beacon_proxy_model_handler.c`.
`..\mesh_common\mesh\handlers\actuator_model\actuator_model_handler.h`	Contains function prototypes of the externally referenced functions defined in `actuator_model_handler.c`.

`..\mesh_common\mesh\handlers\diagnostic_model\diagnostc_model_handler.h`	Contains function prototypes of the externally referenced functions defined in `diagnostic_model_handler.c`.
`..\mesh_common\mesh\handlers\firmware_model\firmware_model_handler.h`	Contains enumerations and function prototypes of the externally referenced functions defined in `firmware_model_handler.c`.
`..\mesh_common\mesh\handlers\otau\app_otau_handler.h`	Contains function prototypes of the externally referenced functions defined in `app_otau_handler.c`.
`..\mesh_common\mesh\handlers\scan\scan_handler.h`	Contains function prototypes of the externally referenced functions defined in `scan_handler.c`.
`..\mesh_common\mesh\handlers\sensor_model\sensor_model_handler.h`	Contains function prototypes of the externally referenced functions defined in `sensor_model_handler.c`.
`..\mesh_common\mesh\handlers\time_model\time_model_handler.h`	Contains function prototypes of the externally referenced functions defined in `time_model_handler.c`.
`..\mesh_common\mesh\handlers\tracker_model\tracker_model_handler.h`	Contains function prototypes of the externally referenced functions defined in `tracker_model_handler.c`.
`..\mesh_common\mesh\handlers\watchdog_model\watchdog_model_handler.h`	Contains function prototypes of the externally referenced functions defined in `watchdog_model_handler.c`.
`..\mesh_common\mesh\handlers\common\app_debug.h`	Contains function prototypes to be used by the application for debugging purpose.
`..\mesh_common\mesh\handlers\common\app_gatt.h`	Contains GATT specific error codes and definitions.
`..\mesh_common\mesh\handlers\common\app_util.h`	Contains function prototypes of the externally referenced functions defined in `app_util.c`.
`..\mesh_common\mesh\handlers\common\appearance.h`	Contains the appearance values to be used in various CSRmesh node applications.
`..\mesh_common\mesh\drivers\battery_hw.h`	Contains prototypes of the externally referenced functions defined in `battery_hw.c`.
`..\mesh_common\mesh\drivers\iot_hw.h`	Contains the function declarations to control the IoT hardware interfaces.
`..\mesh_common\peripheral\conn_param_update.h`	Contains the function definitions defined in the `conn_param_update.c`
`..\mesh_common\components\connection_manager\cm_api.h`	Contains the public API's for the connection manager.
`..\mesh_common\components\connection_manager\cm_types.h`	Contains the public data structure definitions for the connection manager.

`..\mesh_common\components\connection_manager\cm_hal.h`	Contains the function definitions defined in the `cm_hal.c`
`..\mesh_common\components\connection_manager\cm_observer.h`	Contains the function definitions defined in the `cm_observer.c`
`..\mesh_common\components\connection_manager\cm_peripheral.h`	Contains the function definitions defined in the `cm_peripheral.c`
`..\mesh_common\components\connection_manager\cm_private.h`	Contains the function definitions defined in the `cm_private.c`
`..\mesh_common\components\connection_manager\cm_security.h`	Contains the function definitions defined in the `cm_security.c`
`..\mesh_common\components\connection_manager\cm_server.h`	Contains the function definitions defined in the `cm_server.c`
`..\mesh_common\components\nvm_manager\nvm_access.h`	Contains prototypes of the externally referenced NVM read/write functions defined in the `nvm_access.c` file.
`..\mesh_common\server\gap\gap_service.h`	Contains prototypes of the externally referenced functions defined in the `gap_service.c` file.
`..\mesh_common\server\gap\gap_uuids.h`	Contains macros for UUID values for GAP Service.
`..\mesh_common\server\gatt\gatt_service.h`	Contains prototypes of the externally referenced functions defined in the `gatt_service.c` file.
`..\mesh_common\server\gatt\gatt_service_uuids.h`	Contains macros for UUID values for GATT Service.
`..\mesh_common\server\mesh_control\mesh_control_service.h`	Contains prototypes of the externally referenced functions defined in the `mesh_control_service.c` file.
`..\mesh_common\server\mesh_control\mesh_control_service_uuids.h`	Contains macros for UUID values for Mesh Control Service.
`..\mesh_common\server\gaia\gaia.h`	Contains prototypes of the externally referenced functions defined in the `gaia.c` file.
`..\mesh_common\server\gaia\gaia_otau_api.h`	Contains public defines for the GAIA OTAu server module
`..\mesh_common\server\gaia\gaia_otau_private.h`	Contains private defines for the GAIA OTAu server module
`..\mesh_common\server\gaia\gaia_service.h`	Contains prototypes of the externally referenced functions defined in the `gaia_service.c` file.
`..\mesh_common\server\gaia\gaia_uuids.h`	Contains UUID macros for GAIA service

`..\mesh_common\server\gaia\byte_utils.h`	Contains prototypes of the externally referenced functions defined in the `byte_utils.c` file.
`..\mesh_common\client\gaia\gaia_client.h`	Contains prototypes of the externally referenced functions defined in the `gaia_client.c` file.
`..\mesh_common\client\gaia\gaia_client_service.h`	Contains prototypes of the externally referenced functions defined in the `gaia_client_service.c` file.
`..\mesh_common\client\gaia\gaia_otau_client_api.h`	Contains public defines for the GAIA OTAu client module
`..\mesh_common\client\gaia\gaia_otau_client_private.h`	Contains private defines for the GAIA OTAu client module
`..\mesh_common\client\gaia\gaia_uuids.h`	Contains UUID macros for GAIA service

As well as the common header files, each application has its own unique header files. Please refer to the appropriate table for the application source files you require:

- Table 7-2 Light application source files
- Table B-3 Bridge application source files
- Table B-7 Heater application source files
- Table B-10 Switch application source files
- Table B-15 Temperature Sensor application source files

3.1.3 Common database files

All CSRmesh applications use the same database files. Table 3-3 lists the common database files.

Table 3-3 Common database files

File Name	Description
`app_gatt_db.db`	Master database files 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.
`Gap_service_db.db`	Contains information related to GAP Service characteristics, their descriptors and values. See Table C for GAP characteristics.
`Gatt_service_db.db`	Contains information related to GATT Service characteristics, their descriptors and values.
`Mesh_control_service_db.db`	Contains information related to CSRmesh Control Service characteristics, their descriptors and values. See Table C.4 for CSRmesh Control Service characteristics.

File Name	Description
`Gaia_db.db`	Contains information related to GAIA service characteristics, their descriptors and values. For more information about GAIA Service characteristics, see section C.5.

4 Code overview

This section describes significant functions of the applications.

NOTE: Some functions are application-specific. This is indicated in the descriptions of the functions.

4.1 Application entry points

4.1.1 Application initialization

The AppInit() function is invoked when the application is powered on or the chip resets. It performs the following initialisation functions:

- Initialisation for the Timers required for the application.
- Initialisation of debug UART if debug is enabled.
- Initialise the hardware peripherals (PIO, PWM, I2C etc.).
- Read the user keys.
- Initialise the NVM.
- Initialise the connection manager:
- Enable the raw advertisement reports to be sent from connection manager
- Initialisation the GATT server services supported in the application
- Initialise the application data structure.
- Initialise the application mesh handler.
- Configure the scheduler parameters and then initialise the mesh scheduler.
- Initialise the model data for models supported by the application. This needs to be called before calling the readPersistentStore() function.
- Read the persistent data from NVM and store onto the application and models data structure.
- Initialise the CSRmesh stack and the core mesh handler which handles the event from the core mesh stack.
- Initialise the supported model handlers and then start the CSRmesh.
- On successful start of the CSRmesh stack the application can start the association procedure or move onto a state where it's already been associated.

4.1.2 Connection handler

The CSRmesh applications have been integrated to work with the connection manager. The connection manager is a common platform for all the uEnergy application development which would ensure the use of common reusable code as well as handle the multiple connections and various roles of the Bluetooth Low Energy system in an efficient and simple way.

The below mentioned CM-specific event are received and handled by the system.

Connection Manager events

- `CM_INIT_CFM`: This event is received on the completion of the connection manager Initialisation. On receiving this event, the application initialises the required GATT services and Mesh. The application then starts advertising.
- `CM_RAW_ADV_REPORT_IND`: This event is received by the connection manager 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.
- `CM_ADV_REPORT_IND:` This event indicates there is new device found during filtered scanning.
- `CM_DISCOVERY_COMPLETE:` This event indicates that discovery procedure has completed successfully or otherwise.
- `CM_CONNECTION_NOTIFY`: This confirmation event indicates that the connection procedure is complete. The connection parameter received gives the status of the connection. The cm_conn_res_success is returned on successful connection and cm_disconn_res_success is received on successful disconnection.
- `CM_BONDING_NOTIFY:` This indication event indicates that pairing completes successfully or otherwise. Refer to *Volume 3, Part H, Section 2.4* and *Section 3.6 of Bluetooth Core Specification Version 4.1 Connection Events*.
- `CM_ENCRYPTION_NOTIFY`: This event indicates a change in the link encryption.
- `CM_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 C.2.
- `CM_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*.
- `CM_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.

4.1.3 System event handler

The below mentioned system events are handled in the application. To give an example, the device receives a low battery notification or a PIO change event. 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 Advertisement handler

The handler is responsible for sending the connectable and non-connectable user advertisements through core mesh stack. The advertisement handler by default sends the connectable advertisements for the mesh bridge functionality.

4.1.5 Scan handler

The handler is responsible for implementing the scan functionality to connect onto a peripheral device during OTA update over LOT Relay.

4.1.6 Core mesh handler

The application initialises the CSRmesh stack and a callback is provided to receive the events sent by the core stack. The below mentioned events are received from the CSRmesh network or caused by internal state change.

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.

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.

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.

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.

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.

4.1.7 OTAu Handler

The OTAu handler is responsible for handling all the GAIA OTAu related events received from GAIA OTAu server and client services.

4.1.8 Actuator Model handler

The actuator model handler handles the CSRmesh Actuator model events received from the CSRmesh network, such as setting the actuator state and values or any other responses for Actuator model commands sent over CSRmesh

- `CSRMESH_ACTUATOR_GET_TYPES:` This event is received when a CSRmesh Control application sends a command to get the supported sensor types in the actuator model. The CSRmesh Temperature Sensor application returns the updated actuator state data to the actuator model to send the response back.
- `CSRMESH_ACTUATOR_SET_VALUE:` This event is received when a CSRmesh Control application sends a command to set the desired temperature value on the temperature sensor device. The CSRmesh Temperature Sensor application returns the updated actuator state data to the actuator model to send the response back. It calls the `SensorWriteValue` function to send the `CSRMESH_SENSOR_WRITE_VALUE` message with the desired and the current air temperature.
- `CSRMESH_ACTUATOR_SET_VALUE_NO_ACK:` This event is received when the CSRmesh Control application sends a command to set the desired temperature value on the temperature sensor device. The CSRmesh Temperature Sensor application calls the `SensorWriteValue` function to send the `CSRMESH_SENSOR_WRITE_VALUE` message with the desired and the current air temperature.

4.1.9 Sensor Model handler

The sensor model handler handles the CSRmesh Sensor model events received from the CSRmesh network, such as setting the sensor state and values or any other responses for Sensor model commands sent over CSRmesh

- `CSRMESH_SENSOR_GET_TYPES:` This event is received when a device in the network sends a command to get the supported sensor types in the sensor model. The CSRmesh application returns the updated sensor state data to the sensor model to send the response back.
- `CSRMESH_SENSOR_READ_VALUE`: This event is received when a CSRmesh Heater application sends a command to read the value of the current temperature or desired

temperature. The CSRmesh application returns the updated sensor state data to the sensor model to send the response back. It calls the `SensorWriteValue` message with the current or desired temperature values.

- `CSRMESH_SENSOR_MISSING:` This event is received when a device in the group reports that it does not have the latest values for supported sensor types. On receiving this event, CSRmesh application sends the updated sensor values using the `SensorWriteValue` message for the requested sensor types.
- `CSRMESH_SENSOR_SET_STATE:` This event is received when a CSRmesh Controller application sends a command to set the repeat interval for a specific sensor type. The CSRmesh application returns the updated sensor state data to the sensor model to send the response back.
- The CSRmesh application calls the `SensorWriteValue` function to send the `CSR_MESH_SENSOR_WRITE_VALUE` message with the desired and the current air temperature at every repeat interval.
- `CSRMESH_SENSOR_GET_STATE:` This event is received when a CSRmesh Controller application to read the repeat interval for a specific sensor type. The CSRmesh application returns the updated sensor state data to the sensor model to send the response back.
- `CSRMESH_SENSOR_WRITE_VALUE:`This event is received when another CSRmesh broadcasts the current air temperature and desired temperature.
- The application updates its own `desired_temperature` value with the received value and returns the sensor state data. If the desired temperature is changed, the application calls the `SensorWriteValue` function to send the `CSR_MESH_SENSOR_WRITE_VALUE` message with the desired and the current air temperature.
- `CSRMESH_SENSOR_WRITE_VALUE_NO_ACK:` This event is received when another CSRmesh Temperature Sensor application broadcasts the current air temperature and desired temperature. The application updates its own desired air temperature value with the received value. If the desired temperature is changed, the application calls the `SensorWriteValue` function to send the `CSR_MESH_SENSOR_WRITE_VALUE` message with the desired and the current air temperature.

4.1.10 Battery Model handler

The battery model handler initializes the model onto the CSRmesh stack and 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.

4.1.11 Attention Model handler

The attention model handler initializes the model onto the CSRmesh stack and 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 is received when a control device requests the attention of a device in the network. The application blinks the LED in red when Attention state is set and resumes the last set light color when the Attention state is reset. If the duration is provided in the message, a timer is started to turn off the attention state upon timeout.

4.1.12 Data Model handler

The data model handler initializes the model onto the CSRmesh stack and handles the CSRmesh Data Stream model events received from the CSRmesh network.

- `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.

This handler also 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.

4.1.13 Firmware Model handler

- The firmware model handler initializes the model onto the CSRmesh stack and handles the CSRmesh Firmware model events received from the CSRmesh network, such as moving the device onto OTA upgrade mode in case of CSR101x platform or sending response for firmware version request received over CSRmesh:CSRMESH_FIRMWARE_GET_VERSION: This message is received to retrieve the CSRmesh version supported on the device.
- CSRMESH_FIRMWARE_UPDATE_REQUIRED: This message is received to put the device in OTA upgrade mode.

4.1.14 Light Model handler

The light model handler initializes the model onto the CSRmesh stack and 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.

4.1.15 Power Model handler

The power model handler initializes the model onto the CSRmesh stack and 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.

4.1.16 LOT Model handler

This LOT model handler initializes the model onto the CSRmesh stack and handles the following CSRmesh LOT model events received from the CSRmesh network when the device is in server role.

- `CSRMESH_LARGEOBJECTTRANSFER_ANNOUNCE:` This message is received when a LOT Client wants to transfer a large object to neighbouring mesh nodes.

This handler also handles the CSRmesh LOT model events received from the CSRmesh network when the device is in client role:

- `CSRMESH_LARGEOBJECTTRANSFER_INTEREST:` This message is sent by LOT Server when it is interested in the large object available with LOT Client.

4.1.17 Action Model handler

This Action model handler initializes the model onto the CSRmesh stack and handles the following CSRmesh Action model events received from the CSRmesh network when the device is in server role. It also implements the procedures to store and execute the actions received from other devices in the CSRmesh network.

- `CSRMESH_ACTION_SET_ACTION`: This event contains the action that needs to be executed on the node device. These actions are stored and executed at the time that have been received in the same message. These actions get invalidated once the action execution is complete.
- `CSRMESH_ACTION_GET_ACTION_STATUS`: This event is received when another device wants to receive the status of the valid existing action status on the supported action Id's.
- `CSRMESH_ACTION_DELETE`: This event is received to delete a specific valid action id on the current device.
- CSRMESH_ACTION_GET: This event is received when another device wants to query the valid actions present in the current device.

4.1.18 Time Model handler

This Time model handler initializes the model onto the CSRmesh stack and handles the CSRmesh Time model events received from the CSRmesh network when the device is in server role. The time model is also responsible for synchronizing the time across the CSRmesh network.

- `CSRMESH_TIME_BROADCAST`: The time broadcast message contains the current time in UTC format. The time model handler retransmits the received time with the offset delay as well as broadcasts the time model messages across specific broadcast intervals.
- `CSRMESH_TIME_SET_STATE`: This event is used to set the Time Broadcast interval onto the current device. The device retransmits the current UTC time during the broadcast interval.
- `CSRMESH_TIME_GET_STATE`: This event is received to retrieve the time broadcast interval present on the current device.

4.1.19 Asset Model handler

This Asset model handler initializes the model onto the CSRmesh stack and handles the following CSRmesh Asset model events received from the CSRmesh network when the device is in server role. The Asset Model is used to periodically announce the asset information onto the CSRmesh network.

- `CSRMESH_ASSET_SET_STATE`: This event contains the state information of the asset device which are stored and used by the asset device while sending the Asset Announce messages.
- `CSRMESH_ASSET_GET_STATE`: This event is received when another device wants to retrieve the state information of the asset.

4.1.20 Tracker Model handler

This Tracker model handler initializes the model onto the CSRmesh stack and handles the following CSRmesh Asset and Tracker model events received from the CSRmesh network. The tracker model handler tracks the assets present in the network and retrieves the asset information through the tracker network. The tracker handler periodically transmits the tracker report containing the cached asset information.

- `CSRMESH_ASSET_ANNOUNCE`: This event contains the asset information being broadcasted onto the n/w. The tracker on receiving the asset information stores it onto the pending cache and then moves onto the tracker cache after a delay factor timeout calculated based on the RSSI of the received asset announce.
- `CSRMESH_TRACKER_REPORT`: This event is received when a tracker report is received from another tracker containing the asset information. The device on receiving the tracker report deletes the asset information in the tracker cache only if the tracker report contains a better RSSI than received by the device.
- CSRMESH_TRACKER_CLEAR_CACHE: On receiving this event the device clears the stored asset information in the pending as well as tracker cache.
- CSRMESH_TRACKER_SET_PROXIMITY_CONFIG: This event sets the tracker state information including the RSSI thresholds and the delay offset information.
- CSRMESH_TRACKER_FIND: The device on receiving this event, checks for the asset information in the tracker cache and sends the information if present otherwise ignores the message.

4.1.21 Diagnostic Model handler

The Diagnostic model handler initializes the model with the CSRmesh stack and handles the CSRmesh Diagnostic model events received from the CSRmesh network when the device is in server role.

4.1.22 Extension Model Handler

The Extension model handler initializes the model with the CSRmesh stack and handles the CSRmesh Extension model events received from the CSRmesh network when the device is in server role. The Extension Model aims at allowing dynamic allocation of OpCode within a Mesh Network.

- CSRMESH_EXTENSION_REQUEST: On receiving the extension request message the newly requested extension opcodes are decoded and checked for conflict with the already present opcodes as well as the opcodes used by the core models of CSRmesh. The extension opcodes are accepted if there is no conflict and no response is sent for the same.
- CSRMESH_EXTENSION_CONFLICT: On receiving the extension conflict, if the conflict is received with reason as 01 then the accepted opcode is removed from the device. If the conflict is received with the reason as 0x80 then the device replaces the received opcode with the received opcode in the conflict message.

4.1.23 Watchdog Model handler

This Watchdog model handler initializes the model onto the CSRmesh stack and handles the following CSRmesh watchdog model events received from the CSRmesh network when the device is in server role.

- CSRMESH_WATCHDOG_SET_INTERVAL: This event sets the interval at which the watchdog messages need to be sent across onto the CSRmesh network.
- CSRMESH_WATCHDOG_MESSAGE: This event should be responded with a watchdog message if the response size value in the message is non zero.

4.1.24 Beacon Model handler

The Beacon model handler initializes the model with the CSRmesh stack and handles the CSRmesh Beacon model events received from the CSRmesh network. The Beacon model is mainly used to manage beacons across the CSRmesh network. The following beacon model messages are handled in the beacon model handler function.

- CSRMESH_BEACON_SET_STATUS: This event contains the status information to be set on the beacon device.
- CSRMESH_BEACON_GET_BEACON_STATUS: This event retrieves the present status information of the beacon device.
- CSRMESH_BEACON_GET_TYPES: This event is used to retrieve the beacon types supported by the current device.
- CSRMESH_BEACON_SET_PAYLOAD: This event is used to set the beacon payload information for a specific type of beacon on the current device.
- CSRMESH_BEACON_GET_PAYLOAD: This event retrieves the payload information for a specific beacon type on the current device.

- CSRMESH_BEACON_BEACON_STATUS: This event is received when another beacon present in the same CSRmesh network sends a status message containing the present status of the beacon.

4.1.25 Beacon Proxy Model handler

The Beacon proxy model handler initializes the model with the CSRmesh stack and handles the CSRmesh Beacon proxy model events received from the CSRmesh network. The Beacon proxy model is mainly used to manage the low powered beacons across the CSRmesh network. The following beacon proxy model messages are handled in the model handler function.

- CSRMESH_BEACONPROXY_ADD: This event contains the beacon or the group information for the devices or groups to be added onto the proxy.
- CSRMESH_BEACONPROXY_GET_STATUS: This event retrieves the present status information of the beacon proxy device.
- CSRMESH_BEACONPROXY_REMOVE: This event is used to remove the beacon or group information from the proxy device.

4.2 Internal state machine for GATT connection

This section describes the different state transitions in the application during connection.

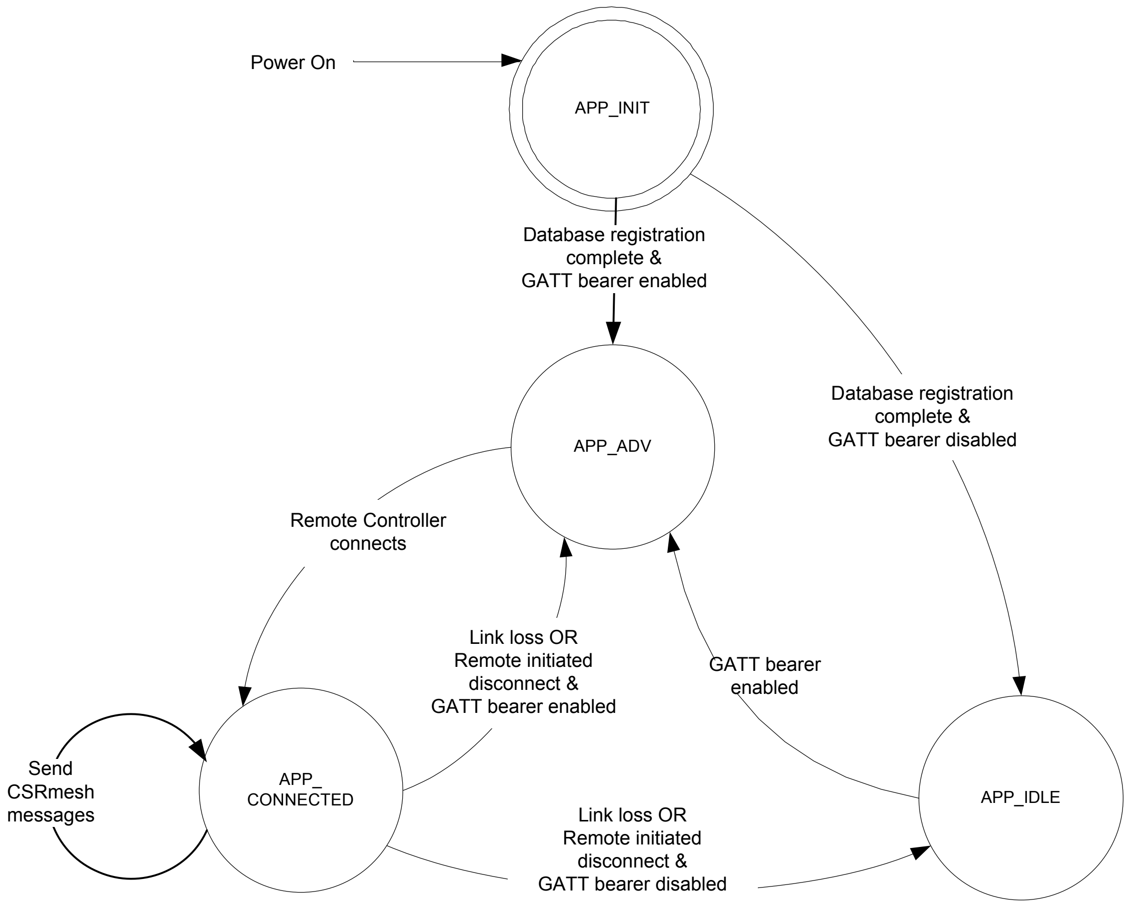

Power On
APP_INIT
Database registration complete & GATT bearer enabled
Database registration complete & GATT bearer disabled
APP_ADV
Remote Controller connects
Link loss OR Remote initiated disconnect & GATT bearer enabled
GATT bearer enabled
Send CSRmesh messages
APP_ CONNECTED
APP_IDLE
Link loss OR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

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 it starts advertising.

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. See Section 6.1 for more information about advertisement timers.

4.2.3 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 device.

- If link loss occurs, the application switches to the `app_state_advertising` state.
- In the case of a remote triggered disconnection, it again starts advertising and enters the `app_state_advertising` state.

4.2.4 app_state_disconnecting

The CSRmesh application never triggers a disconnection on its own. The application no longer stops the CSRmesh activity on disconnection.

NOTE: If the disconnection is triggered, the application enters the `app_state_advertising` state.

4.3 Synchronizing with CSRmesh activity

The CSRmesh application connects to the CSRmesh control application in a bridge device role. The application must synchronize with the CSRmesh library to avoid collision of the advertisements and connection events with the CSRmesh activity. The application calls the CSRmesh APIs to synchronize the connection radio events and the connectable advertisements with the CSRmesh library.

4.4 CSRmesh association

The device needs to be associated with a CSRmesh network to communicate with other devices in the network. In CSRmesh 2.1, the application does not send device identification messages periodically by default. The UUID and AC of the device can be programmed into the device by the µEnergy Production Test Tool. **CSRmeshQRCodeScanner** generates the QR code corresponding to the known UUID and Authorization Code, see section 5. For details about device association, refer to the *CSRmesh 2.0 Android Control Application Note* or *CSRmesh 2.0 iOS Control Application Note*. Figure 4-2 shows the application association state machine:

- `app_state_not_associated`: When the application is first flashed on the device it is in this state. In this state the application is ready to associate with a CSRmesh network and sends the CSRmesh device ID advertisements every five seconds.
- `app_state_association_started`: The application enters this state when it receives an association request from the control device.
- `app_state_associated`: The application enters this state when association completes. The application saves the association state on the NVM and it continues to be associated after power cycle. The application moves to the `app_state_not_associated` state when it receives `CSR_MESH_CONFIG_RESET_DEVICE` or on a two-second long press of the SW2 button.

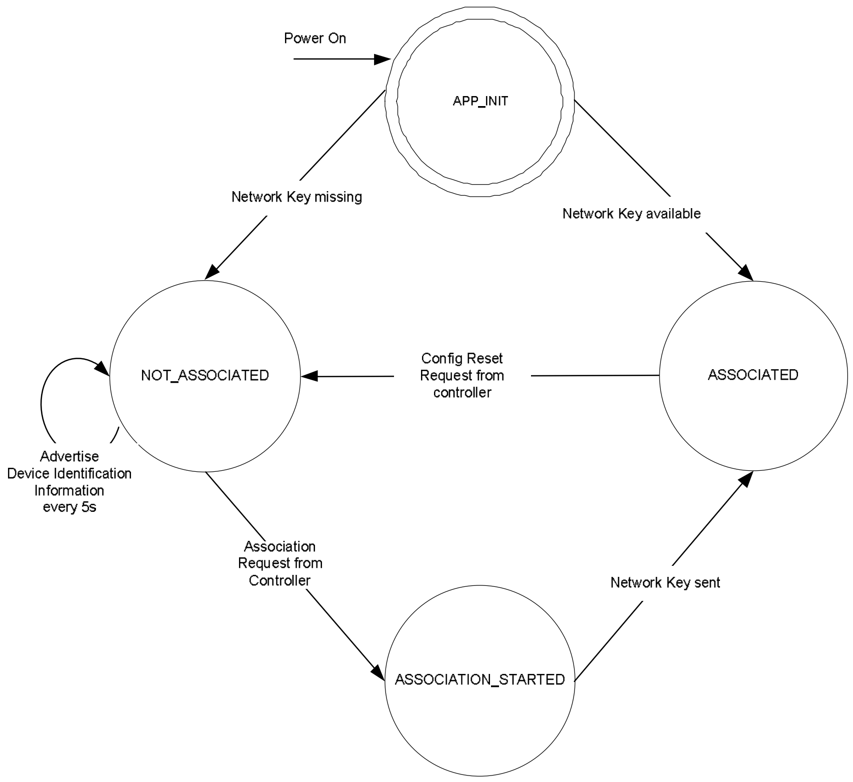

Power On
APP_INIT
Network Key missing
Network Key available
NOT_ASSOCIATED
Config Reset Request from controller
ASSOCIATED
Advertise Device Identification Information every 5s
Association Request from Controller
Network Key sent
ASSOCIATION_STARTED

Figure 4-2 CSRmesh Association State Machine

4.5 Synchronizing with CSRmesh activity

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

4.5.1 Application connectable advertising

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

The application calls `CSRSchedSendUserAdv()` for sending connectable adverts instead of the firmware API. This function schedules one application advertising event as soon as it is called.

4.5.2 Connection events

The application notifies the GATT connection events to the CSRmesh stack for it to schedule the CSRmesh activity. The application calls `CSRSchedNotifyGattEvent(ucid, conn_interval)` for the CSRmesh library to synchronize with connection events. This 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 applications support two bearers:

- GATT Bearer
- BLE Advertising Bearer

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

- Relay is always enabled on both the bearers unless it is disabled by the `CSR_MESH_BEARER_SET_STATE` message or the CS User Key configuration. See Section 4.6.
- When the device is not associated the promiscuous mode is enabled on both the bearers. This helps relay 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 CSRmesh network 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. The application uses a message format listed in Table 4-2 to exchange messages.

Table 4-1 Application data stream message format

Message Code (1 octet)	Length (1 or 2 octets)	Data (0 – 32767 octets)
Identifies type of message. See Table 4-2 for supported messages.	Length of the data. 1. If the MSB of the first octet is 0 then length will be only 7 bits. 2. If the MSB of the first octet is 1 then the length will be a 15 bits value	Data associated with the message

Table 4-2 Application data stream messages

Message	Code	Length	Data	Description
`CSR_DEVICE_INFO_REQ`	`0x01`	0	None	Used to request device information. It can be sent as stream or a datagram message.
`CSR_DEVICE_INFO_RSP`	`0x02`	Variable (1-32767)	Device information	A text string containing information about device. Will be sent over a stream. The application sends a text string containing information about supported CSRmesh features.

Message	Code	Length	Data	Description
`CSR_DEVICE_INFO_SET`	`0x03`	Variable (1-32767)	Device information	A text/binary data that is stored on the device as device information. The application stores the data associated with this message as the device information and responds with this data for any further device information requests.
`CSR_DEVICE_INFO_RESET`	`0x04`	0	None	Resets device information. The application returns default string to any further device information requests.
-	`0x05 - 0xFF`	-	-	Not defined.

4.7.1 Application connectable advertising

The application sends connectable advertisements at regular intervals when the device is powered and 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 one application advertising event when called.

4.7.2 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)` for the CSRmesh library to synchronize with connection events. It is called in the following event handlers:
  - `LM_EV_CONNECTION_COMPLETE`
  - `LS_CONNECTION_PARAM_UPDATE_IND`
  - `LM_EV_DISCONNECT_COMPLETE`

4.8 GAIA Over-the-Air upgrade

The VM Upgrade Protocol over GAIA service enables wireless upgrade of the application software of CSR102x devices. 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 GAIA OTA upgrade, the application needs to:

- Add support for the GAIA server service to an application.
- Add GAIA OTA Upgrade server functionality to the on-chip application, see Section 6.8.

The user can further specify the version of the new image to be downloaded using `.upd` files provided in the project workspace, see section 6.10 for more information. When the device is enabled for GAIA OTAu, the CSR µEnergy Over-the-Air Upgrader Host application included in the SDK can be used to update the device. When the device gets updated, it can further relay the new image using Large Object Transfer (LOT) Model. Currently LOT Relay functionality is only supported in Light application, see Section 4.8.

For information on GAIA OTAu host applications for iOS and Android, see www.csrsupport.com.

Detailed information on the GAIA service is available in the service specification *CS-346805-SP GAIA GATT Service Specification*. Technical details of the GAIA upgrade protocol are available in *CS-347923-SP Upgrade Protocol Specification.* For more information on CSR102x OTAu system, see *CS-348308-AN CSR102x Over-the-air Upgrade (OTAU) System Application Note.*

NOTE:

- Only one GATT connection is allowed at a time. So at a time node application will be either connected as bridge or it will be connected for GAIA OTAu.
- Mesh is temporarily stopped when GAIA OTA upgrade is taking place. Mesh is restarted once the update gets over.
- All node applications support LOT Server and GAIA OTAu server functionality.
- Only Light application supports LOT Client and GAIA OTAu client functionality. (LOT Relay)

4.8.1 Gaia OTAu (App Store)

All node applications support GAIA OTAu in app store. This functionality allows node devices to update their own image if any new version is available. Node devices will only accept the same image types in app store. For e.g. a beacon device will not accept a switch image.

Following procedure needs to be followed for performing GAIA OTAu on CSR102x devices:-

1. Open the node application workspace in xIDE.
2. Flash CSR102x CSRmesh development board with the application.
3. Create a new .bin file with a new version, see section 6.10.
4. Run the µEnergy Over-the-Air Upgrader PC application included with the µEnergy SDK.
5. Choose **Select Image File**… and navigate to the output folder of the node application directory (usually named depend_Release_CSR102x_A05 or depend_Debug_CSR102x_A05 or, depending on the build option selected) and select the .bin file created in Step 3.
6. Select **Scan for Device** on the PC application.
7. Select the device flashed in Step 4 in the list and press the **Upgrade** button on the PC application.

When the download is complete, the device will disconnect from the host, reboot in to the new application, and then wait for the host to automatically reconnect to complete the upgrade.

The Light application also supports the LOT Relay functionality. Once the update is done, light application will start sending LOT Announce messages to neighboring nodes, see Section 4.9.

4.8.2 Gaia OTAu (User Store)

Light application also supports GAIA OTAu in user store. This functionality allows light device to store other types of images (switch, beacon, heater, etc.) in user store and relay the image to the appropriate device.

User Store should only be enabled for 16Mbit or more flash devices .For enabling user store, following line need to be uncommented from `app_store_config_smem_a.stores` file in Light Application project.

```
# Enable User Store for OTAu update. This should be only enabled for
testing with 16Mbit devices
#User Store 1, FLASH, USER, 3, 106496 , No, <Null>
```

Following procedure needs to be followed for performing GAIA OTAu:-

1. Open the Light application workspace in xIDE.
2. Flash CSR102x CSRmesh development board with the Light application.
3. Open a different application workspace (other than light e.g. beacon, switch, etc.) in xIDE.
4. Create a new .bin file with a new version of that application see section 6.10.
5. Run the µEnergy Over-the-Air Upgrader PC application included with the µEnergy SDK.
6. Choose **Select Image File**… and navigate to the output folder of the node application directory (usually named depend_Release_CSR102x_A05 or depend_Debug_CSR102x_A05 or, depending on the build option selected) and select the .bin file created in Step 4.
7. Select **Scan for Device** on the PC application.
8. Select the Light device flashed in Step 2 in the list and press the **Upgrade** button on the PC application.

When the download is complete, the original light application will boot up as the app store will still contain the light application. The new image will be present in the user store. Once the update is done, Light application will start sending LOT Announce messages to neighboring nodes. See Section 4.9 for more information.

4.9 LOT Relay

The Light application also supports the LOT Relay functionality. After receiving a new image, the node will issue a LOT_Announce message with associated content type and TTL=0.The immediate neighboring nodes will respond with LOT_Interest message if they are interested in that content type and will start advertising with a connectable 128bits service id composed through the concatenation of agreed 48 bit value, the LOT Consumer device ID, and bits 64..127 of the SHA-256 of the payload of the LOT_ANNOUNCE message. When LOT publisher receives the LOT_Interest message, it will start scanning for the same service id composed in a similar way. Once the connection is done, LOT publisher node can upgrade the LOT consumer node using VM Upgrade Protocol over GAIA service. The progress of update can be seen on UART. Once the update has been done, the neighboring node will boot into the new application if update has been done on app store. If neighboring node is a relay device, it will further relay this new image.

To enable a device for LOT Relay, the application needs to add support of GAIA client service and GAIA OTA Upgrade client functionality to the on-chip application, see Section 6.9.See Section 6.9 for information on customising LOT Relay parameters.

5 NVM map

Table 5-1 lists the application stores the parameters in the NVM to prevent loss in the event of a power off or a chip panic.

Table 5-1 NVM map for applications

Entity Name	Type	Size of Entity (Words)	NVM Offset (Words)
CSRmesh Stack NVM	uint16 array	32	0
CSRmesh Device UUID (This is part of 31 words of Stack NVM)	uint16 array	8	1
CSRmesh Device Authorization Code (This is part of 31 words of Stack NVM)	uint16 array	4	9
Sanity Word	uint16	1	32

Table 5-2 NVM map for GAP service

Entity Name	Type	Size of Entity (Words)	NVM Offset (Words)
GAP Device Name Length	uint16	1	33
GAP Device Name	uint8 array	20	34

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

Table 5-3 NVM Map for GAIA server service

Entity Name	Type	Size of Entity (Words)
Gaia In Progress Identifier	uint32	4
Gaia Client Configuration Descriptor	uint16	1

Table 5-4 NVM Map for GAIA Client Service

Entity Name	Type	Size of Entity (Words)
Previous Partition Header	uint8 array	14
Previous Partition Information	uint8 array	8
Previous Partition Footer Signature	uint8 array	32
Current Partition Header	uint8 array	14
Current Partition Information	uint8 array	8

Entity Name	Type	Size of Entity (Words)
Current Partition Footer Signature	uint8 array	32
Previous Store Id	uint16	1
Previous Store Type	uint16	1
Previous Partition Size	uint32	2
Current Store Id	uint16	1
Current Store Type	uint16	1
Current Partition Size	uint32	2
Commit Done	bool	1
Calculate Hash	bool	1

6 Customizing the application

This section describes how to customize some parameters of the CSRmesh application.

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

6.1 Advertisement timers

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

Table 6-1 Advertisement timers

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

6.2 Connection parameters

The CSRmesh application uses the connection parameters listed in Table 6-2 by default. The macros for these values are defined in the file `gap_conn_params.h`. These values are chosen by considering the overall current consumption of the device and optimum performance of the device in the CSRmesh network. QTIL recommends not modifying this parameter. If the connection interval is set to less than 13 ms, the device cannot scan for CSRmesh messages from the associated network but only messages received from the GATT connection. Refer to the *Bluetooth Core Specification Version 4.1* for the connection parameter range.

Table 6-2 Connection parameters

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

6.3 Device name

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

6.4 Device address

The application uses a public address by default. The `USE_STATIC_RANDOM_ADDRESS` macro in the `app_gatt.h` file enables the support for static random addresses. If enabled, the application sets a new random address during the application initialization upon a power on reset.

6.5 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. This macro is defined during compilation to let the application know for which NVM type it is built. If EEPROM is selected, `NVM_TYPE_EEPROM` is defined; if SPI Flash is selected, the macro `NVM_TYPE_FLASH` is defined. Follow the comments in the `.keyr` file.

6.6 Application features

Some parameters of the CSRmesh applications can be customized by uncommenting the required definition in the file `user_config.h`. These parameters vary between the different applications, and are described in the relevant application sections.

6.7 CSRmesh parameters

You can change default the CSRmesh parameters (in the `.keyr` file) suit your particular application requirements. The CSRmesh library sets the parameters based on the CS user key values. Table 6-3 lists recommended values for these parameters for optimal performance of devices over the CSRmesh network.

Table 6-3 Configuring CSRmesh parameters

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 unconnectable 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 it runs for the first time. This can avoid having same UUID on multiple devices without explicitly programming a UUID on each device. Bits 3 to 15 are ignored.

6.8 Configuring GAIA OTA upgrade support

GAIA OTA Upgrade support is enabled in the application by default. The user can disable GAIA OTAu server support by setting the macro **Create OTA Update Files** to **No** in **Project Properties**. If LOT Relay is supported, the user can disable GAIA OTAu client support by removing the macro `GAIA_OTAU_RELAY_SUPPORT` from **Define Macros** in **Project Properties.**

6.9 Configuring LOT Relay Support

LOT Relay support is enabled in the application by default. The user can disable LOT Relay support by removing the macro `GAIA_OTAU_RELAY_SUPPORT` from **Define Macros** in **Project Properties**.

Below table lists the LOT Relay customizable parameters given in `user_config.h`.

Table 6-4 LOT Relay Customizable Parameters

Parameter	Value
`SCAN_WINDOW`	Scan window to be used while scanning Value : 10 ms
`SCAN_INTERVAL`	Scan interval to be used while scanning Value : 10 ms

Parameter	Value
SCAN_TIMEOUT	Time after which scanning is stopped if no device is found Value: 3 minutes
DISCOVERY_TIMER	Time after which discovery is started after connection Value: 150 ms
ANNOUNCE_COUNT	Number of times Announce message is sent with ANNOUNCE_COUNT_INTERVAL interval Value: 8
ANNOUNCE_COUNT_INTERVAL	ANNOUNCE_COUNT messages will be sent with this interval Value: 5 seconds
ANNOUNCE_INTERVAL	Gap between two sets of Announce messages Value: 2 minutes
CONN_TIMEOUT	Time after which connection request is sent to verify new image reboot Value: 15 seconds
CONN_RETRY_COUNT	Number of connection retries Value: 13

6.10 Configuring upgrade header file

The user can change the version of the new image to be downloaded using `.upd` files provided in the project workspace. For `Release` build, `mesh_release.upd` file is used and for `Debug` build, `mesh_debug.upd` is used. Figure 6-1 shows the default values of application version set in .upd files. If a new image is created, revision version should be increased in .upd file. `APP_NEW_VERSION` should also be updated with the same value in `user_config.h`, see Figure 6-2.

```
# App Version
# Major Version is 6 bits (MSB).Value should be in range of (0 - 63)
# Minor Version is 4 bits. Value should be in range of (0 - 15)
# Revision Version is 6 bits (LSB).Value should be in range of (0 -
63)
user_header_value_8bit 2
user_header_value_8bit 1
user_header_value_8bit 0
```

Figure 6-1 Application version in mesh_release.upd

```
/* Application version */
#define APP_MAJOR_VERSION (2)
#define APP_MINOR_VERSION (1)
#define APP_NEW_VERSION   (0)
```

Figure 6-2 Application version in user_config.h

User can also update the name of .bin file to be created in .upd file, see Figure 6-3.

```
# Get .bin file
# Release mode
partition -1 -1 depend_Release_CSR102x_A05/light_update.bin
```

Figure 6-3 Modifying .bin file

7 CSRmesh Light application

The main example application supplied with CSRmesh is the Light application. This section covers the unique features of the CSRmesh Light application. The functions and features described in the previous sections of this document also apply.

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, refer to the *CSRmesh API Guide*.

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

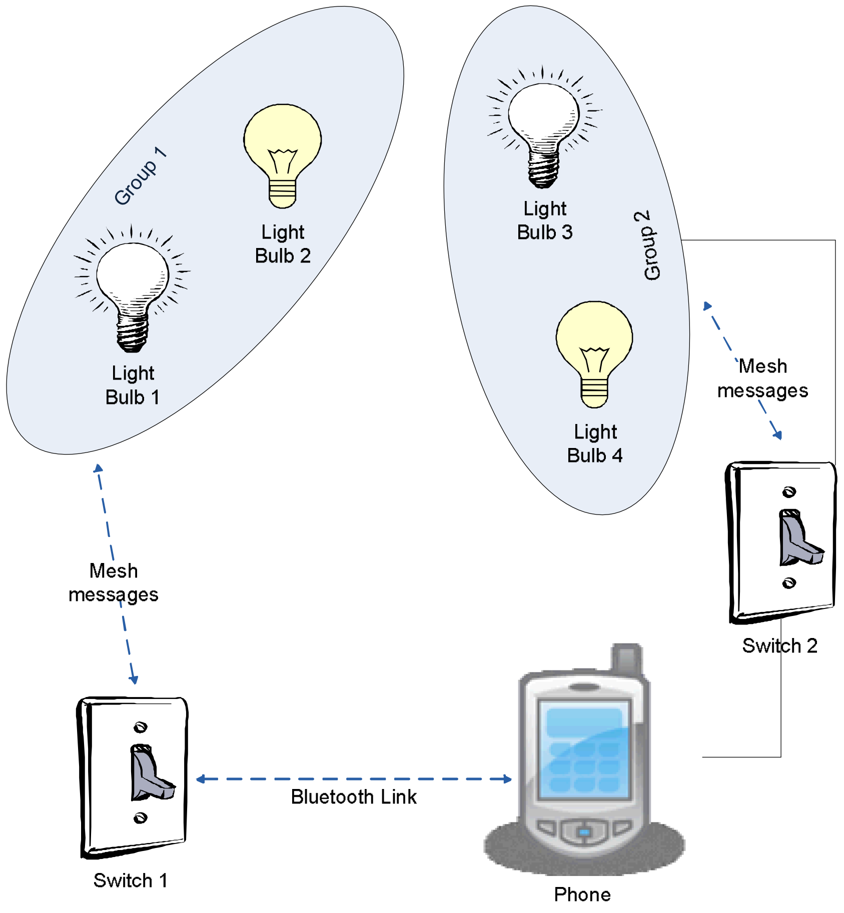


Figure 7-1 CSRmesh Light application use case

7.1.1 CSRmesh models supported by Light application

Table 7-1 lists the CSRmesh models that the Light application supports.

Table 7-1 CSRmesh models supported by Light application

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.

CSRmesh Model	Application Action
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.
Time Model	Enables the application to store and synchronize the UTC time across CSRmesh network.
Action Model	Enables the node application to perform few scheduled actions on devices present in the same CSRmesh network.
Asset Model	Enables the node application to announce the presence of an asset onto CSRmesh network.
Tracker Model	Enabling the tracker model helps to track the assets present in the CSRmesh network.
Tuning Model	Tuning model is designed to allow for the local management of duty cycles through the determination of the local network density.
Ping Model	The Ping model is used to send a reliable message delivery across the CSRmesh network.
LOT Model	Gives the Node application the ability to transfer large objects across nodes in the CSRmesh network.

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

7.1.2 Light application source files

Table 7-2 Light application source files

File Name	Description
`main_app.c`	Implements all the entry functions such as `AppInit()`, `AppProcessSystemEvent()` and `AppProcessLmEvent()`. Handles events received from the hardware and firmware first. Contains handling functions for the system events.
`app_hw.c`	Implements the abstract hardware interface to configure and control the peripherals on specific hardware.
`app_mesh_handler.c`	Implements the application functions to communicate with the model and the core mesh handlers.
`pio_ctrlr_code.asm`	Implements the 8051 assembly code for fast PWM.

7.1.3 Light application header files

Table 7-3 Light application header files

File Name	Description
`main_app.h`	Contains data structures and prototypes of externally referenced functions defined in the `main_app.c` file. It also contains the NVM structure defined for the application.
`app_conn_param.h`	Contains the connection parameters for various configurations.
`app_hw.h`	Contains the definitions for the hardware interface files for the application.
`app_mesh_handler.h`	Contains the definitions for the application interface functions implemented to communicate with the common and the mesh model handlers.
`user_config.h`	Contains the models and the application configurations supported.

7.1.4 Light application configuration files

Table 7-4 Light application configuration files

File Name	Description
`app_gatt_db.db`	The database file containing the service database supported by the application.
`app_store_config_smem_a.stores`	The application smem configuration containing the size of the supported stores on the CSR102x platform.
`bootloader.keyr`	The bootloader keyr file containing the keys to be used by the bootloader for OTA update on the CSR101x platform
`csr_mesh_light_csr101x.xiw` & `csr_mesh_light_csr101x.xip`	The xide project files used by the SDK for building and flashing the application onto CSR101x hardware.
`csr_mesh_light_csr101x_A05.keyr`	The keyr file containing the user keys supported by the application to be used on the CSR101x platform.
`csr_mesh_light_csr101x_uenergyprops.xml`	The xml file containing the connection manager features for the supported application on CSR101x platform.
`csr_mesh_light_csr102x.xiw` & `csr_mesh_light_csr102x.xip`	The xide project files used by the SDK for building and flashing the application onto CSR102x hardware.
`csr_mesh_light_csr102x.htf`	The htf file containing the user keys supported by the application to be used on the CSR102x platform.
`csr_mesh_light_csr102x_uenergyprops.xml`	The xml file containing the connection manager features for the supported application on CSR102x platform.

File Name	Description
`mesh_release.upd`	The file containing the partition information of the application on CSR102x platform for release build configuration
`Mesh_debug.upd`	The file containing the partition information of the application on CSR102x platform for debug build configuration

7.1.5 Light application features

This section describes how to customize some parameters of the CSRmesh Light application. The application can be configured by uncommenting the required definition in `user_config.h`.

Table 7-5 Light application configurations

Configuration	Description
`ENABLE_DEVICE_UUID_ADVERTS`	If defined 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_BATTERY_MODEL`	Enables support for CSRmesh battery model.
`USE_AUTHORISATION_CODE`	Enforces Authorisation Code check on device during association. If this is enabled, the associating control device must have the same authorisation code to associate device to network. Enabled by default.
`COLOUR_TEMP_ENABLED`	Enable support for setting the color temperature.
`ENABLE_TUNING_MODEL`	Enable support for tuning model in the application
`DEFAULT_TUNING_PROBE_PERIOD`	Set the default tuning probe period for analysing the density of the network
`DEFAULT_TUNING_REPORT_PERIOD`	Set the tuning report period for sending periodic tuning reports onto the network.
`ENABLE_ASSET_MODEL`	Support asset model in the application
`ASSET_SIDE_EFFECT_VALUE`	The asset side effect value supported by the device
`ENABLE_TRACKER_MODEL`	Enable the tracker model support in the application
`TRACKER_MAX_CACHED_ASSETS`	Tracker cache size for storing the assets
`TRACKER_MAX_PENDING_ASSETS`	Tracker temporary pending cache for storing assets
`TRACKER_CACHE_RSSI_ROLLING_AVG`	Tracker to store asset rssi based on the rolling average.
`ENABLE_TIME_MODEL`	Enable time model support in the application
`ENABLE_ACTION_MODEL`	Enable action model support in the application
`MAX_ACTIONS_SUPPORTED`	Maximum actions stored in the application NVM.

A CSRmesh directory structure

Below is the directory structure for your CSRmesh application. It lists the files that you may need to amend when building new applications, in the directories where they are stored.

Directories are indicated by being in **bold**.

Csr_mesh_*application-name*_app

```
app_conn_params.h
app_gatt_db.db
app_hw.c
app_hw.h
app_mesh_handler.c
app_mesh_handler.h
app_store_config_smem_a.stores
csr_mesh_application-name_csr101x.xip
csr_mesh_application-name_csr101x.xiw
csr_mesh_application-name_csr101x_A05.keyr
csr_mesh_application-name_csr101x_uenergyprops.xml
csr_mesh_application-name_csr102x.htf
csr_mesh_application-name_csr102x.xip
csr_mesh_application-name_csr102x.xiw
csr_mesh_application-name_csr102x_uenergyprops.xml
application-name_hw.c
application-name_hw.h
main_app.c
main_app.h
pio_ctrlr_code.asm
user_config.h
```

Csr_mesh_*application-name*_app_common/

- **components**
- **interfaces**
- **mesh**
- **peripheral**
- **server**

Csr_mesh_*application-name*_app_common/components

```
connection_manager
nvm_manager
```

Csr_mesh_*application-name*_app_common/components/connection_manager:

```
cm_api.h
cm_appearance.h
cm_central.c
cm_central.h
cm_client.c
cm_client.h
cm_common.c
cm_hal.c
cm_hal.h
cm_observer.c
cm_observer.h
cm_peripheral.c
cm_peripheral.h
cm_private.c
cm_private.h
cm_security.c
cm_security.h
cm_server.c
cm_server.h
cm_types.h
docs
```

Csr_mesh_*application-name*_app_common/components/connection_manager/docs

```
Doxyfile
api_description.h
```

Csr_mesh_*application-name*_app_common/components/nvm_manager

```
nvm_access.c
nvm_access.h
```

Csr_mesh_*application-name*_app_common/interfaces

i2c

pio

pwm

spi

Csr_mesh_*application-name*_app_common/interfaces/i2c

```
i2c_comms.c
i2c_comms.h
i2c_comms_csr101x.c
i2c_comms_csr102x.c
```

Csr_mesh_*application-name*_app_common/interfaces/pio

```
pio_csr101x.c
pio_csr102x.c
pio_interface.h
```

Csr_mesh_*application-name*_app_common/interfaces/pwm

```
pwm.c
pwm.h
```

Csr_mesh_*application-name*_app_common/interfaces/spi

spi_interface.c
spi_interface.h

Csr_mesh_*application-name*_app_common/mesh

drivers
handlers

Csr_mesh_*application-name*_app_common/mesh/drivers

csr_mesh_nvm.h
csr_mesh_ps_ifce.c
fast_pwm.c
fast_pwm.h
iot_hw.c
iot_hw.h

Csr_mesh_*application-name*_app_common/mesh/handlers

advertisement
common
connection
core_mesh
data_model
***application-name*_model**
power_model

Csr_mesh_*application-name*_app_common/mesh/handlers/advertisement

advertisement_handler.c
advertisement_handler.h

Csr_mesh_*application-name*_app_common/mesh/handlers/common

app_debug.c
app_debug.h
app_gatt.h
app_util.c
app_util.h
appearance.h

Csr_mesh_*application-name*_app_common/mesh/handlers/connection

connection_handler.c
connection_handler.h

Csr_mesh_*application-name*_app_common/mesh/handlers/core_mesh

core_mesh_handler.c
core_mesh_handler.h

Csr_mesh_*application-name*_app_common/mesh/handlers/data_model

data_model_handler.c
data_model_handler.h

Csr_mesh_*application-name*_app_common/mesh/handlers/*application-name*_model

*application-name*_model_handler.c
*application-name*_model_handler.h

Csr_mesh_*application-name*_app_common/mesh/handlers/power_model
power_model_handler.c
power_model_handler.h

Csr_mesh_*application-name*_app_common/peripheral
conn_param_update.c
conn_param_update.h

Csr_mesh_*application-name*_app_common/server
csr_ota
dev_info
gap
gatt
link_loss
mesh_control

Csr_mesh_*application-name*_app_common/server/csr_ota
csr_ota_db.db
csr_ota_service.c
csr_ota_service.h
csr_ota_uuids.h

Csr_mesh_*application-name*_app_common/server/dev_info
dev_info_service.c
dev_info_service.h
dev_info_service_db.db
dev_info_uuids.h

Csr_mesh_*application-name*_app_common/server/gap
gap_service.c
gap_service.h
gap_service_db.db
gap_uuids.h

Csr_mesh_*application-name*_app_common/server/gatt
gatt_service.c
gatt_service.h
gatt_service_db.db
gatt_service_uuids.h

Csr_mesh_*application-name*_app_common/server/link_loss
link_loss_service.c
link_loss_service.h
link_loss_service_db.db
link_loss_uuids.h

Csr_mesh_*application-name*_app_common/server/mesh_control

```
      mesh_control_service.c
      mesh_control_service.h
      mesh_control_service_db.db
      mesh_control_service_uuids.h
```

B Other CSRmesh applications

The following sections describe some of the other example applications that can be built using the µEnergy SDK. They also describe the features unique to the particular applications.

B.1 CSRmesh Bridge

This section cover the unique features of the CSRmesh bridge application. The functions and features described in the previous sections of this document also apply.

The Bridge application implements the custom Mesh Control Service. This service allows a Bluetooth Smart enabled phone to connect using the Mesh Control GATT Service to send and receive CSRmesh messages to other devices. The bridge application does not support CSRmesh association.

The Bridge Use Case is illustrated in Figure B-1Figure B-.

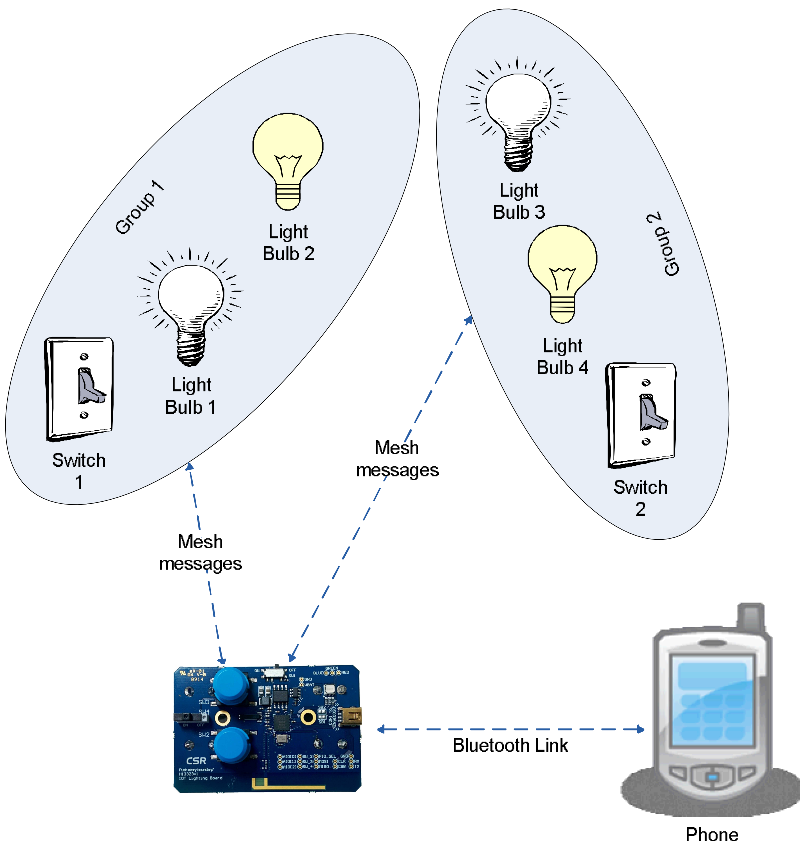


Figure B-1 CSRmesh Bridge application use case

B.1.1 Bridge application source files

Table B-3 Bridge application source files

File Name	Description
`main_app.c`	Implements all the entry functions such as `AppInit()`, `AppProcessSystemEvent()` and `AppProcessLmEvent()`. Handles events received from the hardware and firmware first. Contains handling functions for the system events.
`app_hw.c`	Implements the abstract hardware interface to configure and control the peripherals on specific hardware.
`app_mesh_handler.c`	Implements the application functions to communicate with the model and the core mesh handlers.

B.1.2 Bridge application header files

Table 7-3 Bridge application header files

File Name	Description
`main_app.h`	Contains data structures and prototypes of externally referenced functions defined in the `main_app.c` file. It also contains the NVM structure defined for the application.
`app_conn_param.h`	Contains the connection parameters for various configurations.
`app_hw.h`	Contains the definitions for the hardware interface files for the application.
`app_mesh_handler.h`	Contains the definitions for the application interface functions implemented to communicate with the common and the mesh model handlers.
`user_config.h`	Contains the models and the application configurations supported.

B.1.3 Bridge application configuration files

Table 7-4 Bridge application configuration files

File Name	Description
`app_gatt_db.db`	The database file containing the service database supported by the application.
`app_store_config_smem_a.stores`	The application smem configuration containing the size of the supported stores on the CSR102x platform.
`bootloader.keyr`	The bootloader keyr file containing the keys to be used by the bootloader for OTA update on the CSR101x platform
`csr_mesh_bridge_csr101x.xiw` & `csr_mesh_bridge_csr101x.xip`	The xide project files used by the SDK for building and flashing the application onto CSR101x hardware.
`csr_mesh_bridge_csr101x_A05.keyr`	The keyr file containing the user keys supported by the application to be used on the CSR101x platform.
`csr_mesh_bridge_csr101x_uenergyprops.xml`	The xml file containing the connection manager features for the supported application on CSR101x platform.
`csr_mesh_bridge_csr102x.xiw` & `csr_mesh_bridge_csr102x.xip`	The xide project files used by the SDK for building and flashing the application onto CSR102x hardware.
`csr_mesh_bridge_csr102x.htf`	The htf file containing the user keys supported by the application to be used on the CSR102x platform.
`csr_mesh_bridge_csr102x_uenergyprops.xml`	The xml file containing the connection manager features for the supported application on CSR102x platform.

File Name	Description
`mesh_release.upd`	The file containing the partition information of the application on CSR102x platform for release configuration
`mesh_debug.upd`	The file containing the partition information of the application on CSR102x platform for debug configuration

B.1.4 Bridge application features

This section describes how to customize some parameters of the CSRmesh Bridge application. The application can be configured by uncommenting the required definition in `user_config.h`.

Table B-2 Bridge application configurations

Configuration	Description
`USE_STATIC_RANDOM_ADDRESS`	If this feature is enabled, the application uses static random address for sending.

B.2 CSRmesh Heater application

This section cover the unique features of the CSRmesh Heater application. The functions and features described in the previous sections of this document also apply.

The CSRmesh Heater application demonstrates the following use cases:

- CSRmesh Heater: The Heater application implements the heater functionality by switching the heater on/off based on the desired and the current air temperature received from the group. The heater status is indicated by the red LED. The application implements the handlers for the CSRmesh messages related to the Sensor Model, Stream Model, Firmware Model and Attention Model.
- CSRmesh GATT Bridge: The application also implements the CSR custom Mesh Control Service (see B.1 CSRmesh Bridge). This service allows a Bluetooth Smart enabled phone to send and receive CSRmesh messages to many devices with the Heater application acting as a bridge.

The application uses the CSRmesh library provided as part of the CSRmesh release. Refer to the *CSRmesh API Guide* documentation for details.

NOTE: The CSRmesh Heater application does not support bonding and is not compatible with the previous releases of CSRmesh.

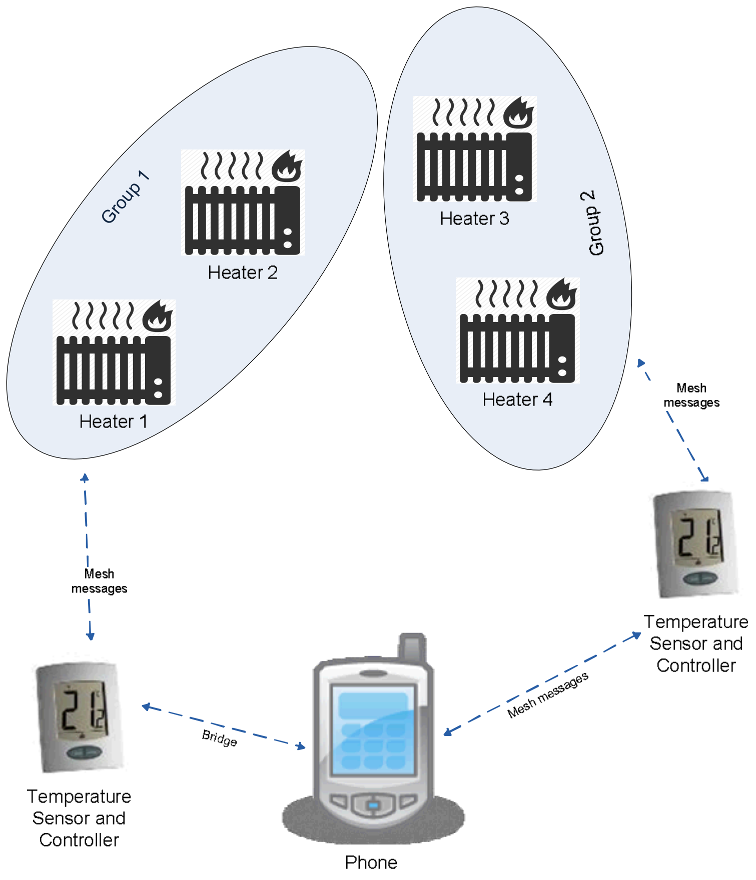


Figure B-2 CSRmesh Heater application use case

B.2.1 CSRmesh models supported by Heater application

Table B-3 CSRmesh models supported by Heater application

CSRmesh Model	Application Action
Sensor Model	Handles the sensor model messages. The application provides the provision of writing and reading the sensor type values from the supported devices.
Attention Model	Handles the `CSR_MESH_ATTENTION_SET_STATE` command. When the attract attention is set the application blinks red.
Battery Model	Upon receiving the `CSR_MESH_GET_BATTERY_STATE` message, it reads the battery level and responds with the current battery level and the state. The application is implemented with reference to the CSRmesh development board, which runs on AA Batteries. So the application sets `BATTERY_MODEL_STATE_POWERING_DEVICE` indicating that the device is battery powered. If the battery level is below threshold it sets the `BATTERY_MODEL_STATE_NEEDS_REPLACEMENT` bit.
Data Stream Model	Enables the application to send and receive stream of bytes from another CSRmesh device.
Time Model	Enables the application to store and synchronize the UTC time across CSRmesh network.
Action Model	Enables the node application to perform few scheduled actions on devices present in the same CSRmesh network.
Ping Model	The Ping model is used to send a reliable message delivery across the CSRmesh network.
LOT Model	Gives the Node application the ability to transfer large objects across nodes in the CSRmesh network.

NOTE: The CSRmesh Heater application supports groups for sensor, Data stream, LOT and attention models. It statically allocates memory to store the assigned group IDs and save them on the NVM.

B.2.2 Low power operation

The sensors are expected to consume very low power as they are generally battery powered and have to give a good battery life. Scanning for CSRmesh messages consumes the maximum power compared to any other application activity. Reducing the scan duty cycle to a very low value reduces the power consumption almost proportionally.

Dynamic Scan Duty Cycle

When scanning in very low duty cycles, the probability of the messages received is greatly reduced, causing failures in association and configuration. So the application dynamically switches between `DEFAULT_RX_DUTY_CYCLE` (a very low percentage value) and `HIGH_RX_DUTY_CYCLE` in different application states as listed below:

- When the application is not associated and the sensor model is not grouped, the application runs with `HIGH_RX_DUTY_CYCLE`.
- Once associated and grouped to a group it switches to `DEFAULT_RX_DUTY_CYCLE`.
- Upon receiving a `CSR_MESH_ATTENTION_SET_STATE` message with attention set to attract attention, the application moves to `HIGH_RX_DUTY_CYCLE` till the attention times out or it is explicitly reset.

- When a control device requests for device data, the application sets `HIGH_RX_DUTY_CYCLE` and starts the data stream. It switches back to the `DEFAULT_RX_DUTY_CYCLE` once the stream transfer completes or the stream times out.

Periodic behavior and multiple retransmissions

The temperature sensor sends the sensor value messages to the group of which the device is a member. The sensor messages are broadcast periodically as configured by the user. Since all the recipients of the sensor value messages are expected to be running on low scan duty cycle, the probability of the message received by the devices is low. To increase the probability of the messages received by the destination devices, the application retransmits each message multiple times. `NUM_OF_RETRANSMISSIONS` defines the number of times the same sensor message needs to be re-transmitted.

Transmit Message Density

Transmit message density is defined as the number of retransmissions of a message within the CSRmesh advertising period. The application uses the message interleaved advertising feature of the CSRmesh stack to set the message density. CSRmesh stack holds the messages in a queue and advertises each message multiple times, interleaving them in the same order as they are inserted in the queue. The interval between adjacent messages are dynamically adjusted with number of messages in the queue such that the time interval between successive transmissions of a message in the queue is the same.

Table B-4 lists the different transmission density settings.

Table B-4 Transmit message density

Message Transmit Density	Advertising Period (approximate)	Number of Advertising Events
1	500 ms	6
2	700 ms	12
3	800 ms	18
4	900 ms	24
5	1100 ms	36

Acknowledged mode message broadcast

The application implements an acknowledged mode of message retransmission where it waits for an acknowledgement for the sensor value messages from the destination devices. Once a response is received from another device in the group, the device stops further retransmissions of the messages until the next period.

B.2.3 Heater application source files

Table B-7 Heater application source files

File Name	Description
`main_app.c`	Implements all the entry functions such as `AppInit()`, `AppProcessSystemEvent()` and `AppProcessLmEvent()`. Handles events received from the hardware and firmware first. Contains handling functions for the system events.

File Name	Description
`app_hw.c`	Implements the abstract hardware interface to configure and control the peripherals on specific hardware.
`app_mesh_handler.c`	Implements the application functions to communicate with the core mesh handlers.
`app_mesh_model_handler.c`	Implements the application functions to communicate with the model handlers.

B.2.4 Heater application header files

Table 7-3 Heaters application header files

File Name	Description
`main_app.h`	Contains data structures and prototypes of externally referenced functions defined in the `main_app.c` file. It also contains the NVM structure defined for the application.
`app_conn_param.h`	Contains the connection parameters for various configurations.
`app_hw.h`	Contains the definitions for the hardware interface files for the application.
`app_mesh_handler.h`	Contains the definitions for the application interface functions implemented to communicate with the core mesh handlers.
`app_mesh_model_handler.h`	Contains the definitions for the application interface functions implemented to communicate with the mesh model handlers.
`user_config.h`	Contains the models and the application configurations supported.

B.2.5 Heater application configuration files

Table 7-4 Heater application configuration files

File Name	Description
`app_gatt_db.db`	The database file containing the service database supported by the application.
`app_store_config_smem_a.stores`	The application smem configuration containing the size of the supported stores on the CSR102x platform.
`bootloader.keyr`	The bootloader keyr file containing the keys to be used by the bootloader for OTA update on the CSR101x platform
`csr_mesh_heater_csr101x.xiw &` `csr_mesh_ heater_csr101x.xip`	The xide project files used by the SDK for building and flashing the application onto CSR101x hardware.
`csr_mesh_heater_csr101x_A05.keyr`	The keyr file containing the user keys supported by the application to be used on the CSR101x platform.

File Name	Description
`csr_mesh_heater_csr101x_uenergyprops.xml`	The xml file containing the connection manager features for the supported application on CSR101x platform.
`csr_mesh_heater_csr102x.xiw` & `csr_mesh_heater_csr102x.xip`	The xide project files used by the SDK for building and flashing the application onto CSR102x hardware.
`csr_mesh_heater_csr102x.htf`	The htf file containing the user keys supported by the application to be used on the CSR102x platform.
`csr_mesh_heater_csr102x_uenergyprops.xml`	The xml file containing the connection manager features for the supported application on CSR102x platform.
`mesh_release.upd`	The file containing the partition information of the application on CSR102x platform for release build configuration
`mesh_debug.upd`	The file containing the partition information of the application on CSR102x platform for debug build configuration

B.2.6 Heater application features

This section describes how to customize some parameters on the CSRmesh Heater application. The application can be configured by uncommenting the required define in `user_config.h`.

Table B-6 Application configuration

Configuration	Description
`ENABLE_DEVICE_UUID_ADVERTS`	If defined 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_BATTERY_MODEL`	Enables support for CSRmesh battery model.
`USE_AUTHORISATION_CODE`	Enforces Authorisation Code check on device during association. If this is enabled, the associating control device must have the same authorisation code to associate device to network. Enabled by default.
`ENABLE_DATA_MODEL`	Enables data stream model to send/receive stream of octets to/from another CSRmesh device. If this is enabled, the application supports streaming of device information string over the data model. The application uses a simple protocol to send and receive messages. See Section 4.6 for details.
`DEBUG_ENABLE`	Enables application debug logging on UART. If enabled, the debug messages can be viewed on a terminal application by connecting the device to the PC and opening the corresponding COM Port with 2400-8-N-1 configuration. Enabled by default. See Section 2.1.1 for details.
`DEFAULT_RX_DUTY_CYCLE`	The percentage Rx duty cycle set by the application once the device is associated and grouped. Default: 2%.

Configuration	Description
`HIGH_RX_DUTY_CYCLE`	Defines the percentage Rx duty cycle value when the device is either in attention mode or in data transfer mode. A higher value here makes the device more responsive for the period of time defined during attention or data transfer. Default: 100%.
`MAX_RETRANSMISSION_TIME`	Defines the total amount of time an acknowledgement message can be retransmitted.
`RETRANSMIT_INTERVAL`	Defines the amount of time after which the acknowledged message to be retransmitted after. The number of times a single message to be retransmitted is calculated by `MAX_RETRANSMISSION_TIME/ RETRANSMIT_INTERVAL`.
`ENABLE_ACK_MODE`	This enables the acknowledged mode communication between the Temperature Sensor and Heater applications. If `ENABLE_ACK_MODE` is defined, the application stores and manages the group of heaters it is communicating with and replies the acknowledgement for the `SensorWriteValue` messages received from the Temperature Sensor applications. Disabled by default.

B.3 CSRmesh Switch application

This section cover the unique features of the CSRmesh Switch application. The functions and features described in the previous sections of this document also apply.

The Switch application has the following use cases:

- Light Control: The Switch application implements the CSRmesh messages related to the Light Model, Stream Model and the Power 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 commands to many devices with the Switch application acting as a bridge.

The CSRmesh Switch application is part of the CSRmesh release to show CSRmesh light control using an associated switch. It uses the CSRmesh library provided as part of the CSRmesh release. For more information, refer to the *CSRmesh Library API* documentation.

NOTE: The CSRmesh Switch application does not support bonding and is not compatible with previous CSRmesh releases.

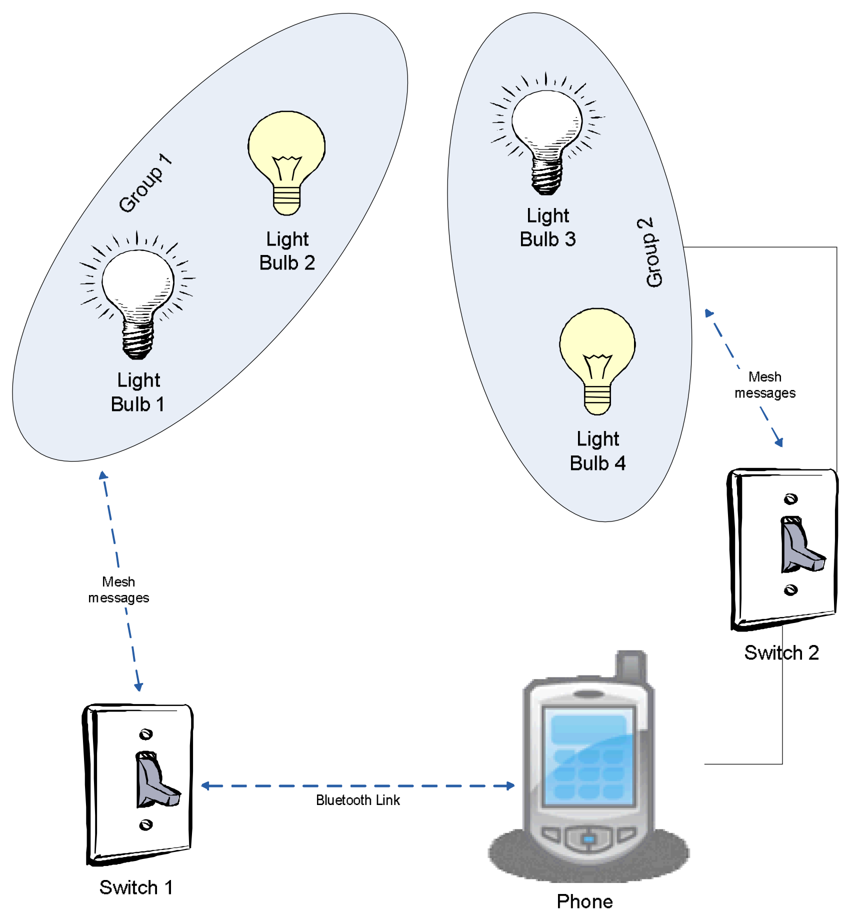


Figure B-3 CSRmesh Switch use case

B.3.1 CSRmesh models supported by Switch application

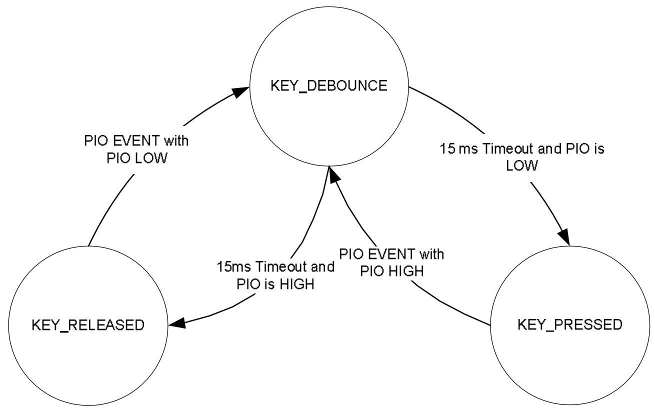
Table B-7 CSRmesh models supported by Switch application

CSRmesh Model	Application Action
Light Model	Sets the light level when the button is pressed.
Power Model	Sends power set commands to the assigned target device when the switch SW4 is toggled. Handles the `POWER_STATE` message received from the target device.
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 the `BATTERY_MODEL_STATE_POWERING_DEVICE` indicating that the device is battery powered. If the battery level is below the threshold, it sets the `BATTERY_MODEL_STATE_NEEDS_REPLACEMENT` bit.
Switch Model	Enables the Switch model. This allows the Control application to identify the device as a switch and to assign group IDs.
Watchdog Model	Enables a device to save power by periodically listening to CSRmesh network for a specified period of time. It also sends a message at the start of listening to notify other CSRmesh devices.
Data Stream Model	Enables the application to send and receive stream of bytes from another CSRmesh device.
Time Model	Enables the application to store and synchronize the UTC time across CSRmesh network.
Action Model	Enables the node application to perform few scheduled actions on devices present in the same CSRmesh network.
Ping Model	The Ping model is used to send a reliable message delivery across the CSRmesh network.
LOT Model	Gives the Node application the ability to transfer large objects across nodes in the CSRmesh network.

NOTE: The CSRmesh Switch application supports groups for Switch, Data Stream, LOT, attention and Watchdog models. It statically allocates memory to store the assigned group IDs and saves them on the NVM.

B.3.2 Button de-bouncing

Figure B-4 shows how the application handles button de-bouncing.



■

Figure B-4 Handling button de-bounce

NOTE: The CSRmesh Switch application blinks blue until associated to the CSRmesh network. When getting an association request, the application blinks yellow until the association completes.

B.3.3 Switch application source files

Table B-10 Switch application source files

File Name	Description
`main_app.c`	Implements all the entry functions such as `AppInit()`, `AppProcessSystemEvent()` and `AppProcessLmEvent()`. Handles events received from the hardware and firmware first. Contains handling functions for the system events.
`app_hw.c`	Implements the abstract hardware interface to configure and control the peripherals on specific hardware.
`app_mesh_handler.c`	Implements the application functions to communicate with the model and the core mesh handlers.

B.3.4 Switch application header files

Table B-9 Switch application header files

File Name	Description
`main_app.h`	Contains data structures and prototypes of externally referenced functions defined in the `main_app.c` file. It also contains the NVM structure defined for the application.
`app_conn_param.h`	Contains the connection parameters for various configurations.
`app_hw.h`	Contains the definitions for the hardware interface files for the application.
`app_mesh_handler.h`	Contains the definitions for the application interface functions implemented to communicate with the common and the mesh model handlers.
`user_config.h`	Contains the models and the application configurations supported.

B.3.5 Switch application configuration files

Table 7-4 Switch application configuration files

File Name	Description
`app_gatt_db.db`	The database file containing the service database supported by the application.
`app_store_config_smem_a.stores`	The application smem configuration containing the size of the supported stores on the CSR102x platform.

File Name	Description
`bootloader.keyr`	The bootloader keyr file containing the keys to be used by the bootloader for OTA update on the CSR101x platform
`csr_mesh_switch_csr101x.xiw` & `csr_mesh_switch_csr101x.xip`	The xide project files used by the SDK for building and flashing the application onto CSR101x hardware.
`csr_mesh_switch_csr101x_A05.keyr`	The keyr file containing the user keys supported by the application to be used on the CSR101x platform.
`csr_mesh_switch_csr101x_uenergyprops.xml`	The xml file containing the connection manager features for the supported application on CSR101x platform.
`csr_mesh_switch_csr102x.xiw` & `csr_mesh_switch_csr102x.xip`	The xide project files used by the SDK for building and flashing the application onto CSR102x hardware.
`csr_mesh_switch_csr102x.htf`	The htf file containing the user keys supported by the application to be used on the CSR102x platform.
`csr_mesh_switch_csr102x_uenergyprops.xml`	The xml file containing the connection manager features for the supported application on CSR102x platform.
`mesh_release.upd`	The file containing the partition information of the application on CSR102x platform for release build configuration
`mesh_debug.upd`	The file containing the partition information of the application on CSR102x platform for debug build configuration

B.3.6 Switch application features

Table B-10 lists how to customize some parameters on the CSRmesh Switch application. The application can be configured by uncommenting the required definition in the `user_config.h` header file.

Table B-10 Switch application configuration

Configuration	Description
`ENABLE_GATT_OTA_SERVICE`	Enables the CSR OTA Update Service. If this feature is enabled, you can request for firmware update over the air using the CSR OTA Update tool.
`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 2400-8-N-1 configuration. **NOTE**: If this feature is enabled, the brightness control function of the buttons SW2 and SW3 is disabled.

Configuration	Description
`USE_AUTHORISATION_CODE`	Enforces Authorisation code check on device during association. If this feature is enabled, the associating control device must have the same authorization code to associate the device to network. Enabled by default.
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 data stream model to send/receive stream of octets to/from another CSRmesh device. If this features is enabled, the application supports streaming of device information string over the data model. The application uses a simple protocol to send and receive messages. See section 4.5.1 for more information.
ENABLE_WATCHDOG_MODEL	Enables watchdog model in the application. If this feature is enabled, the device listens only for a certain interval periodically to save power.
USE_STATIC_RANDOM_ADDRESS	If this features is enabled, the application uses static random address for sending connectable adverts.

B.4 CSRmesh Temperature Sensor application

This section cover the unique features of the CSRmesh Temperature Sensor application. The functions and features described in the previous sections of this document also apply.

It implements the temperature sensor driver to periodically sample the current temperature and broadcasts it to the group.

The application demonstrates the following use cases:

- CSRmesh Temperature Sensor: The Temperature Sensor application implements a temperature sensor which periodically reads the air temperature and broadcasts it to the group of devices. It implements the handlers for the CSRmesh messages related to the Sensor Model, Actuator Model, Stream Model, Firmware Model and Attention Model.
- Desired temperature setting: Optionally this application implements a desired temperature setting control using the on-board buttons. It broadcasts the desired temperature whenever the user changes the setting.
- 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 temperature sensor application acting as a bridge.

The CSRmesh Temperature Sensor Application is part of the CSRmesh release to demonstrate CSRmesh Temperature Sensor use case.

The application uses the CSRmesh library provided as part of the CSRmesh release. Refer to the *CSRmesh API Guide* documentation for details.

NOTE: The CSRmesh Temperature Sensor application does not support bonding and is not compatible with previous releases of CSRmesh.

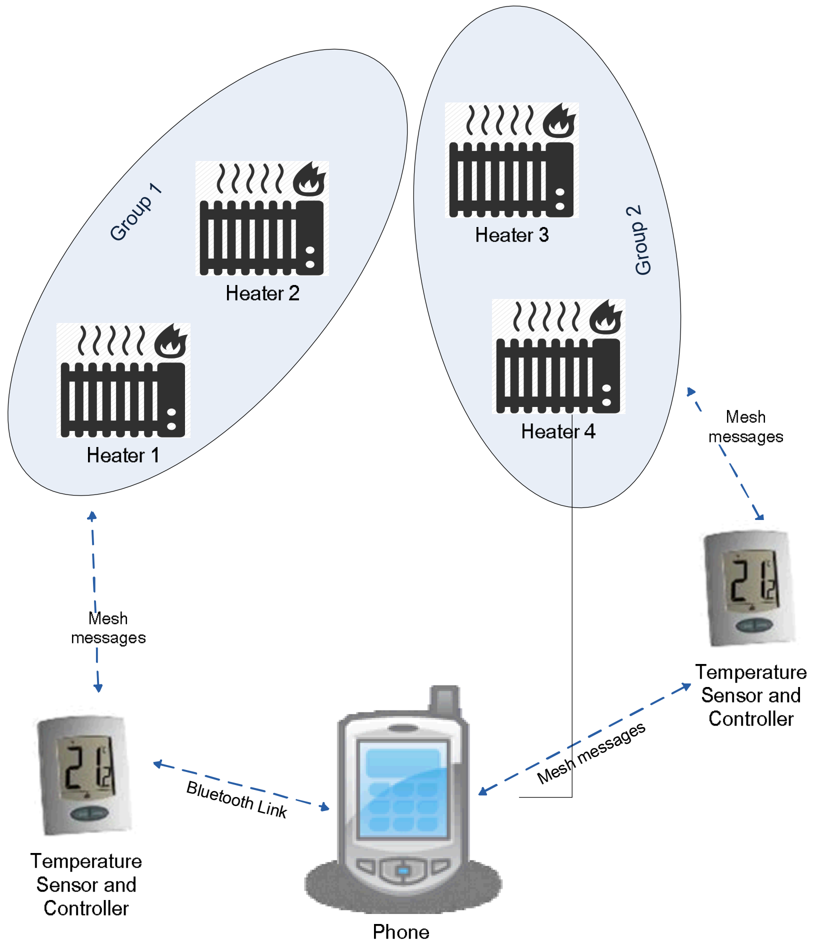


Figure B-5 CSRmesh Temperature Sensor use case

B.4.1 CSRmesh models supported by Temperature Sensor application

Table B-11 lists the CSRmesh models that he Temperature Sensor application supports.

Table B-11 CSRmesh models supported by Temperature Sensor application

CSRmesh Model	Application Action
Sensor Model	Handles the sensor model messages. The application provides the provision of writing and reading the sensor type values from the supported devices.
Actuator Model	Handles the actuator model messages from the control devices. The application provides the provision of writing the desired temperature value from the control devices.
Attention Model	Handles the `CSR_MESH_ATTENTION_SET_STATE` command. When the attract attention is set the application blinks in red colour.
Battery Model	Upon receiving the `CSR_MESH_GET_BATTERY_STATE` message, it reads the battery level and responds with the current battery level and the state. The application is implemented with reference to the CSRmesh development board which runs on AA Batteries. So the application sets `BATTERY_MODEL_STATE_POWERING_DEVICE` indicating that the device is battery powered. If the battery level is below threshold it sets the `BATTERY_MODEL_STATE_NEEDS_REPLACEMENT` bit.

CSRmesh Model	Application Action
Data Model	This model enables the application to send and receive stream of bytes from another CSRmesh device.
Time Model	Enables the application to store and synchronize the UTC time across CSRmesh network.
Action Model	Enables the node application to perform few scheduled actions on devices present in the same CSRmesh network.
Ping Model	The Ping model is used to send a reliable message delivery across the CSRmesh network.
LOT Model	Gives the Node application the ability to transfer large objects across nodes in the CSRmesh network.

NOTE: The CSRmesh Temperature Sensor application supports groups for sensor, data stream, LOT and attention models. It statically allocates memory to store the assigned group IDs and save them on the NVM.

B.4.2 Temperature measurement

Sensor

The application uses the STTS751 I^{2}C temperature sensor on the CSRmesh development board to read the temperature periodically. The interval at which the temperature is read can be configured by `TEMPERATURE_SAMPLING_INTERVAL`. The drivers for configuring the sensor and the interfacing peripherals are implemented in `stts751_temperature_sensor.c` and `stts751_temperature_sensor.h` files.

Broadcasting temperature change

The application starts broadcasting the current temperature at the defined sampling interval when the device is associated and grouped with other sensors and actuators. The current temperature value is broadcasted only if the change in temperature is more than the tolerance value from the last broadcast value. The tolerance value can be configured by `TEMPERATURE_CHANGE_TOLERANCE`. See Section 6.6 for configuration

B.4.3 Low power operation

The sensors are expected to consume very low power as they are generally battery powered and have to give a good battery life. Scanning for CSRmesh messages consumes the maximum power compared to any other application activity. Reducing the scan duty cycle to a very low value reduces the power consumption almost proportionally.

Dynamic scan duty cycle

When scanning in very low duty cycles, the probability of the messages received is greatly reduced, causing failures in association and configuration. So the application dynamically switches between `DEFAULT_RX_DUTY_CYCLE` (a very low percentage value) and `HIGH_RX_DUTY_CYCLE` in different application states as listed below:

- When the application is not associated and the sensor model is not grouped, the application runs with `HIGH_RX_DUTY_CYCLE`.
- Once associated and grouped to a group it switches to `DEFAULT_RX_DUTY_CYCLE`.

- Upon receiving a `CSR_MESH_ATTENTION_SET_STATE` message with attention set to attract attention, the application moves to `HIGH_RX_DUTY_CYCLE` till the attention times out or it is explicitly reset.
- When a control device requests for device data, the application sets `HIGH_RX_DUTY_CYCLE` and starts the data stream. It switches back to the `DEFAULT_RX_DUTY_CYCLE` once the stream transfer completes or the stream times out.

Periodic behavior and multiple retransmissions

The temperature sensor sends the sensor value messages to the group of which the device is a member. The sensor messages are broadcast periodically as configured by the user. Since all the recipients of the sensor value messages are expected to be running on low scan duty cycle, the probability of the message received by the devices is low. To increase the probability of the messages received by the destination devices, the application retransmits each message multiple times. `NUM_OF_RETRANSMISSIONS` defines the number of times the same sensor message needs to be re-transmitted.

Transmit message density

Transmit message density is defined as the number of retransmissions of a message within the CSRmesh advertising period. The application uses the message interleaved advertising feature of the CSRmesh stack to set the message density. CSRmesh stack holds the messages in a queue and advertises each message multiple times, interleaving them in the same order as they are inserted in the queue. The interval between adjacent messages are dynamically adjusted with number of messages in the queue such that the time interval between successive transmissions of a message in the queue is the same.

Table B-12 lists the different transmission density settings.

Table B-12 Transmit message density

Message Transmit Density	Advertising Period (approximate)	Number of Advertising Events
1	500 ms	6
2	700 ms	12
3	800 ms	18
4	900 ms	24
5	1100 ms	36

Acknowledged mode message broadcast

The application implements an acknowledged mode of message retransmission where it waits for an acknowledgement for the sensor value messages from the destination devices. Once a response is received from another device in the group, the device stops further retransmissions of the messages until the next period.

B.4.4 Temperature Sensor application source files

Table B-15 Temperature Sensor application source files

File Name	Purpose
`main_app.c`	Implements all the entry functions such as `AppInit()`, `AppProcessSystemEvent()` and `AppProcessLmEvent()`.

File Name	Purpose
	Handles events received from the hardware and firmware first. Contains handling functions for the system events.
`app_hw.c`	Implements the abstract hardware interface to configure and control the peripherals on specific hardware.
`app_mesh_handler.c`	Implements the application functions to communicate with the core mesh handlers.
`app_mesh_model_handler.c`	Implements the application functions to communicate with the model handlers.
`i2c_comms.c`	Implements the I^{2}C interface drivers for communicating with the peripheral device.

B.4.5 Temperature Sensor application header files

Table B-14 Temperature Sensor application header files

File Name	Purpose
`main_app.h`	Contains data structures and prototypes of externally referenced functions defined in the `main_app.c` file. It also contains the NVM structure defined for the application.
`app_conn_param.h`	Contains the connection parameters for various configurations.
`app_hw.h`	Contains the definitions for the hardware interface files for the application.
`app_mesh_handler.h`	Contains the definitions for the application interface functions implemented to communicate with the core mesh handlers.
`app_mesh_model_handler.h`	Contains the definitions for the application interface functions implemented to communicate with the mesh model handlers.
`user_config.h`	Contains the models and the application configurations supported.
`i2c_comms.h`	Contains macro definitions and the prototypes for externally defined functions in the `i2c_comms.c`.

B.4.6 Temperature Sensor application configuration files

Table 7-4 Temperature sensor application configuration files

File Name	Description
`app_gatt_db.db`	The database file containing the service database supported by the application.
`app_store_config_smem_a.stores`	The application smem configuration containing the size of the supported stores on the CSR102x platform.
`bootloader.keyr`	The bootloader keyr file containing the keys to be used by the bootloader for OTA update on the CSR101x platform
`csr_mesh_temp_sensor_csr101x.xiw` & `csr_mesh_temp_sensor_csr101x.xip`	The xide project files used by the SDK for building and flashing the application onto CSR101x hardware.
`csr_mesh_temp_sensor_csr101x_A05.keyr`	The keyr file containing the user keys supported by the application to be used on the CSR101x platform.

File Name	Description
`csr_mesh_temp_sensor_csr101x_uenergyprops.xml`	The xml file containing the connection manager features for the supported application on CSR101x platform.

`csr_mesh_temp_sensor_csr102x.xiw &`
`csr_mesh_temp_sensor_csr102x.xip`
	The xide project files used by the SDK for building and flashing the application onto CSR102x hardware.

`csr_mesh_temp_sensor_csr102x.htf`	The htf file containing the user keys supported by the application to be used on the CSR102x platform.
`csr_mesh_temp_sensor_csr102x_uenergyprops.xml`	The xml file containing the connection manager features for the supported application on CSR102x platform.
`mesh_release.upd`	The file containing the partition information of the application on CSR102x platform for release build configuration
`mesh_debug.upd`	The file containing the partition information of the application on CSR102x platform for debug build configuration

B.4.7 Temperature Sensor application features

This section describes how to customize some parameters on the CSRmesh Temperature Sensor application. The application can be configured by uncommenting the required define in `user_config.h`.

Table B-15 Temperature Sensor application configuration

Configuration	Description
`ENABLE_DEVICE_UUID_ADVERTS`	If defined 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_BATTERY_MODEL`	Enables support for CSRmesh battery model.
`USE_AUTHORISATION_CODE`	Enforces Authorisation Code check on device during association. If this is enabled, the associating control device must have the same authorisation code to associate device to network.
`ENABLE_DATA_MODEL`	Enables data stream model to send/receive stream of octets to/from another CSRmesh device. If this is enabled, the application supports streaming of device information string over the data model. The application uses a simple protocol to send and receive messages. See Section 4.6 for details.

Configuration	Description
`ENABLE_TEMPERATURE_CONTROLLER`	Enables the Temperature Sensor application to operate as a temperature controller which can set the desired temperature value on the application. If `DEBUG_ENALBE` is defined, the desired temperature setting can be controlled via UART. The desired temperature can be changed by typing **+** or **-** on the UART terminal. If `DEBUG_ENABLE` is not defined, the desired temperature value can be changed by pressing the SW2 or SW3 button on the CSRmesh development board. See Section 2.1.1 for the button functions.
`DEBUG_ENABLE`	Enables application debug logging on UART. If enabled, the debug messages can be viewed on a terminal application by connecting the device to the PC and opening the corresponding COM Port with 2400-8-N-1 configuration. Enabled by default. See Section 2.1.1 for details.
`TEMPERATURE_SENSOR_STTS751`	Enables the initialisation and working with the STTS751 temperature sensor on the CSRmesh development board.
`TEMPERATURE_SAMPLING_INTERVAL`	Defines the interval in seconds to query the current air temperature from the temperature sensor. The application queries the temperature every sampling interval indefinitely. Default: 15 seconds.
`TEMPERATURE_CHANGE_TOLERANCE`	Defines the temperature change tolerance in 1/32 kelvin units. Default: 32 (=1 degree kelvin) See Section B.4.2 for details.
`DEFAULT_RX_DUTY_CYCLE`	The percentage Rx duty cycle set by the application once the device is associated and grouped. Default: 2%.
`HIGH_RX_DUTY_CYCLE`	Defines the percentage Rx duty cycle value when the device is either in attention mode or in data transfer mode. A higher value here makes the device more responsive for the period of time defined during attention or data transfer. Default: 100%.
`TRANSMIT_MSG_DENSITY`	Defines the number of messages inserted into the mesh queue in one single instance. The value that can be defined is 1 to 5. The default value is set to 2. See Section B.4.3 for details.
`NUM_OF_RETRANSMISSIONS`	Defines the number of times the message needs to be retransmitted. This is by default set to 60.
`ENABLE_ACK_MODE`	This enables the acknowledged mode communication between the Temperature Sensor and Heater applications. If `ENABLE_ACK_MODE` is defined, the application stores and manages the group of heaters it is communicating with and waits for the acknowledgement from the heaters for all the `SensorWriteValue` messages.

B.5 CSRmesh Beacon application

This section cover the unique features of the CSRmesh Beacon application. The functions and features described in the previous sections of this document also apply.

The CSRmesh Beacon application demonstrates the following use cases:

- CSRmesh Beacon: The Beacon application implements the Beacon functionality by periodically sending the supported beacons over LE. The beacon application by default supports the below mentioned beacon types.
    - CSR Retail Beacon
    - Eddystone URL Beacon
    - Eddystone UID Beacon
    - Apple Ibeacon

  The beacon payload for all the supported beacon types are updated through the CSRmesh network over the beacon model messages. The beacon application runs in low power mode during the beaconing and then moves into active mesh mode at periodic intervals where the beacon payload and the beacon status information gets updated.

  The application implements the handlers for the CSRmesh messages related to the Beacon Model, Stream Model, Time Model, Action Model and Attention Model.
- CSRmesh GATT Bridge: The application also implements the CSR custom Mesh Control Service (see B.1 CSRmesh Bridge). This service allows a Bluetooth Smart enabled phone to send and receive CSRmesh messages to many devices with the Heater application acting as a bridge.

The application uses the CSRmesh library provided as part of the CSRmesh release. Refer to the *CSRmesh API Guide* documentation for details.

NOTE: The CSRmesh Beacon application does not support bonding and is not compatible with the previous releases of CSRmesh.

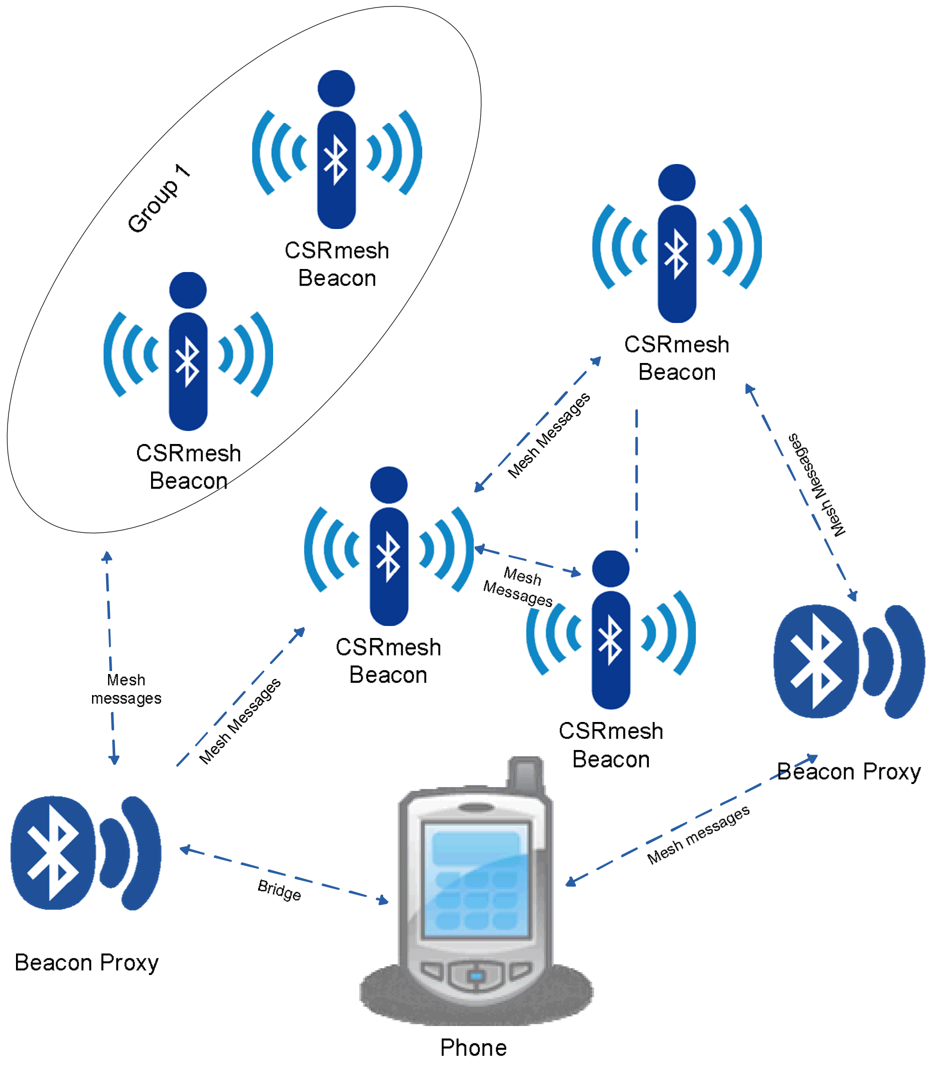


Figure B-6 CSRmesh Beacon application use case

B.5.1 CSRmesh models supported by Beacon application

Table B-16 CSRmesh models supported by Beacon application

CSRmesh Model	Application Action
Beacon Model	Handles the beacon model messages. The application implements the provision of sending multiple beacon types over the LE channel at a time. The Beacon model also provides a way to update the beacon intervals as well as update the beacon payload over the CSRmesh network.
Attention Model	Handles the `CSR_MESH_ATTENTION_SET_STATE` command. When the attract attention is set the application blinks red.
Battery Model	Upon receiving the `CSR_MESH_GET_BATTERY_STATE` message, it reads the battery level and responds with the current battery level and the state. The application is implemented with reference to the CSRmesh development board, which runs on AA Batteries. So the application sets `BATTERY_MODEL_STATE_POWERING_DEVICE` indicating that the device is battery powered. If the battery level is below threshold it sets the `BATTERY_MODEL_STATE_NEEDS_REPLACEMENT` bit.
Data Stream Model	Enables the application to send and receive stream of bytes from another CSRmesh device.
Time Model	Enables the application to store and synchronize the UTC time across CSRmesh network.
Action Model	Enables the node application to perform few scheduled actions on devices present in the same CSRmesh network.
Ping Model	The Ping model is used to send a reliable message delivery across the CSRmesh network.
LOT Model	Gives the Node application the ability to transfer large objects across nodes in the CSRmesh network.

NOTE: The CSRmesh Beacon application supports groups for beacon, Data stream, LOT and attention models. It statically allocates memory to store the assigned group IDs and save them on the NVM.

B.5.2 Beacon and Mesh Mode

The application would be either in Beacon or Mesh mode at any particular instinct. During the Beacon mode the application would be beaconing the beacons set and would not be acting as mesh node. Hence the CSR mesh stack would not be processing the mesh messages and relaying them over the network in this mode. The Beacon application would periodically enter the mesh mode to update the beacon payload and the status information. This is done by periodically sending a Beacon Status message onto the CSRmesh network announcing the presence of the beacons in the mesh mode.

B.5.3 Beacon application source files

Table 7-17 Beacon application source files

File Name	Description
`main_app.c`	Implements all the entry functions such as `AppInit()`, `AppProcessSystemEvent()` and `AppProcessLmEvent()`. Handles events received from the hardware and firmware first. Contains handling functions for the system events.
`app_hw.c`	Implements the abstract hardware interface to configure and control the peripherals on specific hardware.
`app_mesh_handler.c`	Implements the application functions to communicate with the core mesh handlers.
`app_mesh_beacon_handler.c`	Implements the application functions to communicate with the beacon model handlers.

B.5.4 Beacon application header files

Table 7-3 Beacon application header files

File Name	Description
`main_app.h`	Contains data structures and prototypes of externally referenced functions defined in the `main_app.c` file. It also contains the NVM structure defined for the application.
`app_conn_param.h`	Contains the connection parameters for various configurations.
`app_hw.h`	Contains the definitions for the hardware interface files for the application.
`app_mesh_handler.h`	Contains the definitions for the application interface functions implemented to communicate with the core mesh handlers.
`app_mesh_beacon_handler.h`	Contains the definitions for the application interface functions implemented to communicate with the beacon model handlers.
`user_config.h`	Contains the models and the application configurations supported.

B.5.5 Beacon application configuration files

Table 7-4 Beacon application configuration files

File Name	Description
`app_gatt_db.db`	The database file containing the service database supported by the application.
`app_store_config_smem_a.stores`	The application smem configuration containing the size of the supported stores on the CSR102x platform.
`bootloader.keyr`	The bootloader keyr file containing the keys to be used by the bootloader for OTA update on the CSR101x platform
`csr_mesh_beacon_csr101x.xiw` & `csr_mesh_beacon_csr101x.xip`	The xide project files used by the SDK for building and flashing the application onto CSR101x hardware.

File Name	Description
`csr_mesh_beacon_csr101x_A05.keyr`	The keyr file containing the user keys supported by the application to be used on the CSR101x platform.
`csr_mesh_beacon_csr101x_uenergyprops.xml`	The xml file containing the connection manager features for the supported application on CSR101x platform.
`csr_mesh_beacon_csr102x.xiw &` `csr_mesh_beacon_csr102x.xip`	The xide project files used by the SDK for building and flashing the application onto CSR102x hardware.
`csr_mesh_beacon_csr102x.htf`	The htf file containing the user keys supported by the application to be used on the CSR102x platform.
`csr_mesh_beacon_csr102x_uenergyprops.xml`	The xml file containing the connection manager features for the supported application on CSR102x platform.
`mesh_release.upd`	The file containing the partition information of the application on CSR102x platform for the release build configuration
`mesh_debug.upd`	The file containing the partition information of the application on CSR102x platform for the debug build configuration

B.5.6 Beacon application features

This section describes how to customize some parameters on the CSRmesh Beacon application. The application can be configured by uncommenting the required define in `user_config.h`.

Table B-18 Application configuration

Configuration	Description
`ENABLE_DEVICE_UUID_ADVERTS`	If defined 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_BATTERY_MODEL`	Enables support for CSRmesh battery model.
`USE_AUTHORISATION_CODE`	Enforces Authorisation Code check on device during association. If this is enabled, the associating control device must have the same authorisation code to associate device to network. Enabled by default.
`ENABLE_DATA_MODEL`	Enables data stream model to send/receive stream of octets to/from another CSRmesh device. If this is enabled, the application supports streaming of device information string over the data model. The application uses a simple protocol to send and receive messages. See Section 4.6 for details.

Configuration	Description
`BEACON_TYPE_SUPPORTED ((1 << csr_mesh_beacon_type_csr)| (1 << csr_mesh_beacon_type_ibeacon) | (1 << csr_mesh_beacon_type_eddystone_url) | (1 << csr_mesh_beacon_type_eddystone_uid))`	The supported beacon types in the application. The application developer can define any subset of the defined beacon types to be supported.
`MAX_BEACONS_SUPPORTED`	The Max beacons supported by the application
`MAX_BEACON_PAYLOAD_SIZE`	The maximum size of the payload for the beacons supported by the application
`APP_BEACON_MODE`	This definition is used to say the mode of the application for the beacon model handler to implement.

B.6 CSRmesh Beacon Proxy application

This section cover the unique features of the CSRmesh Beacon Proxy application. The functions and features described in the previous sections of this document also apply.

The CSRmesh Beacon Proxy application demonstrates the following use cases:

- CSRmesh Beacon Proxy: The Beacon Proxy application implements the Beacon Model and the Beacon Proxy Model functionality. The Beacon proxy application acts as a proxy to the beacon devices and the Beacon groups that are configured through the Beacon Proxy Model. The Beacon Proxy application is always present in the mesh mode and is always awake to get the beacon payload and the status information update. Hence the controller would be able to update the beacon proxy at any instant which would then update the individual or group of beacons whenever they come back to Mesh Mode in the CSRmesh network. The beacon application by default supports the below mentioned beacon types.
  - CSR Retail Beacon
  - Eddystone URL Beacon
  - Eddystone UID Beacon
  - Apple Ibeacon

  The beacon payload for all the supported beacon types are updated through the CSRmesh network over the beacon model messages.

  The application implements the handlers for the CSRmesh messages related to the Beacon Model, Beacon Proxy Model, Data Stream Model, Time Model, Action Model and Attention Model.
- CSRmesh GATT Bridge: The application also implements the CSR custom Mesh Control Service (see B.1 CSRmesh Bridge). This service allows a Bluetooth Smart enabled phone to send and receive CSRmesh messages to many devices with the Heater application acting as a bridge.

The application uses the CSRmesh library provided as part of the CSRmesh release. Refer to the *CSRmesh API Guide* documentation for details.

NOTE: The CSRmesh Beacon application does not support bonding and is not compatible with the previous releases of CSRmesh.

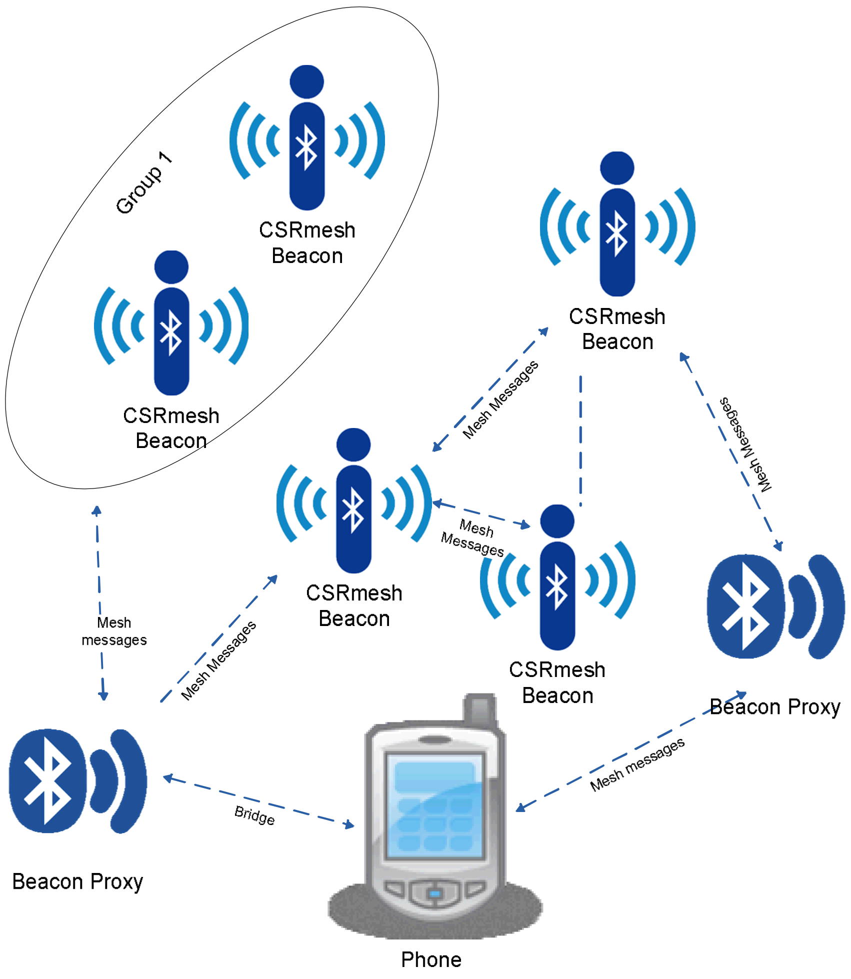


Figure B-7 CSRmesh Beacon Proxy application use case

B.6.1 CSRmesh models supported by Beacon application

Table B-19 CSRmesh models supported by Beacon application

CSRmesh Model	Application Action
Beacon Model	Handles the beacon model messages. The application implements the provision of sending multiple beacon types over the LE channel at a time. The Beacon model also provides a way to update the beacon intervals as well as update the beacon payload over the CSRmesh network.
Beacon Proxy Model	Handles the addition and deletion of the Beacons or groups of beacons that need to be supported by the beacon proxy.
Attention Model	Handles the `CSR_MESH_ATTENTION_SET_STATE` command. When the attract attention is set the application blinks red.
Battery Model	Upon receiving the `CSR_MESH_GET_BATTERY_STATE` message, it reads the battery level and responds with the current battery level and the state. The application is implemented with reference to the CSRmesh development board, which runs on AA Batteries. So the application sets `BATTERY_MODEL_STATE_POWERING_DEVICE` indicating that the device is battery powered. If the battery level is below threshold it sets the `BATTERY_MODEL_STATE_NEEDS_REPLACEMENT` bit.
Data Stream Model	Enables the application to send and receive stream of bytes from another CSRmesh device.
Time Model	Enables the application to store and synchronize the UTC time across CSRmesh network.
Action Model	Enables the node application to perform few scheduled actions on devices present in the same CSRmesh network.
Ping Model	The Ping model is used to send a reliable message delivery across the CSRmesh network.
LOT Model	Gives the Node application the ability to transfer large objects across nodes in the CSRmesh network.

NOTE: The CSRmesh Beacon application supports groups for beacon, Data stream, LOT and attention models. It statically allocates memory to store the assigned group IDs and save them on the NVM.

B.6.2 Beacon Proxy Update

The Proxy application would always be in Mesh mode responding to the requests by the beacons as well as other controllers present in the CSRmesh network. The Controller can update the new beacon payload or the status information onto the beacon proxy application at any instant of time as the proxy application is always in mesh mode. The Beacon application would periodically enter the mesh mode to update the beacon payload and the status information from the proxy. This is done by periodically sending a Beacon Status message onto the CSRmesh network announcing the presence of the beacons in the mesh mode. When the proxy applications listens to the status of beacons which are part of its network, it would check whether there are any pending payloads or newer beacon status information which would be updated to these individual beacons. The proxy application would delete these beacon information after the update for individual beacons.

B.6.3 Beacon Proxy application source files

Table 7-20 Beacon Proxy application source files

File Name	Description
`main_app.c`	Implements all the entry functions such as `AppInit()`, `AppProcessSystemEvent()` and `AppProcessLmEvent()`. Handles events received from the hardware and firmware first. Contains handling functions for the system events.
`app_hw.c`	Implements the abstract hardware interface to configure and control the peripherals on specific hardware.
`app_mesh_handler.c`	Implements the application functions to communicate with the core mesh and the model handlers.

B.6.4 Beacon Proxy application header files

Table 7-3 Beacon Proxy application header files

File Name	Description
`main_app.h`	Contains data structures and prototypes of externally referenced functions defined in the `main_app.c` file. It also contains the NVM structure defined for the application.
`app_conn_param.h`	Contains the connection parameters for various configurations.
`app_hw.h`	Contains the definitions for the hardware interface files for the application.
`app_mesh_handler.h`	Contains the definitions for the application interface functions implemented to communicate with the core mesh and the model handlers.
`user_config.h`	Contains the models and the application configurations supported.

B.6.5 Beacon Proxy application configuration files

Table 7-4 Beacon Proxy application configuration files

File Name	Description
`app_gatt_db.db`	The database file containing the service database supported by the application.
`app_store_config_smem_a.stores`	The application smem configuration containing the size of the supported stores on the CSR102x platform.
`bootloader.keyr`	The bootloader keyr file containing the keys to be used by the bootloader for OTA update on the CSR101x platform
`csr_mesh_beacon_proxy_csr101x.xiw &` `csr_mesh_beacon_proxy_csr101x.xip`	The xide project files used by the SDK for building and flashing the application onto CSR101x hardware.
`csr_mesh_beacon_proxy_csr101x_A05.keyr`	The keyr file containing the user keys supported by the application to be used on the CSR101x platform.

File Name	Description
`csr_mesh_beacon_proxy_csr101x_uenergyprops.xml`	The xml file containing the connection manager features for the supported application on CSR101x platform.
`csr_mesh_beacon_proxy_csr102x.xiw` & `csr_mesh_beacon_proxy_csr102x.xip`	The xide project files used by the SDK for building and flashing the application onto CSR102x hardware.
`csr_mesh_beacon_proxy_csr102x.htf`	The htf file containing the user keys supported by the application to be used on the CSR102x platform.
`csr_mesh_beacon_proxy_csr102x_uenergyprops.xml`	The xml file containing the connection manager features for the supported application on CSR102x platform.
`mesh_release.upd`	The file containing the partition information of the application on CSR102x platform for release configuration
`mesh_debug.upd`	The file containing the partition information of the application on CSR102x platform for debug configuration

B.6.6 Beacon Proxy application features

This section describes how to customize some parameters on the CSRmesh Beacon Proxy application. The application can be configured by uncommenting the required define in `user_config.h`.

Table B-21 Application configuration

Configuration	Description
`ENABLE_DEVICE_UUID_ADVERTS`	If defined 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_BATTERY_MODEL`	Enables support for CSRmesh battery model.
`USE_AUTHORISATION_CODE`	Enforces Authorisation Code check on device during association. If this is enabled, the associating control device must have the same authorisation code to associate device to network. Enabled by default.
`ENABLE_DATA_MODEL`	Enables data stream model to send/receive stream of octets to/from another CSRmesh device. If this is enabled, the application supports streaming of device information string over the data model. The application uses a simple protocol to send and receive messages. See Section 4.6 for details.
`BEACON_TYPE_SUPPORTED ((1 << csr_mesh_beacon_type_csr)| (1 << csr_mesh_beacon_type_ibeacon) | (1 << csr_mesh_beacon_type_eddystone_url) | (1 << csr_mesh_beacon_type_eddystone_uid))`	The supported beacon types in the application. The application developer can define any subset of the defined beacon types to be supported.

Configuration	Description
`MAX_BEACONS_SUPPORTED`	The Max buffers allocated for the beacon information supported by the application
`MAX_BEACON_PAYLOAD_SIZE`	The maximum size of the payload for the beacons supported by the application
`APP_PROXY_MODE`	This definition is used to say the mode of the application for the beacon model handler to implement.
`MAX_MANAGED_BEACON_GRPS`	The max number of beacon groups supported by the proxy application.
`MAX_MANAGED_BEACON_DEVS`	The max number of individual beacons supported by the proxy application.

B.7 CSRmesh Lumicast application

This section cover the unique features of the CSRmesh Lumicast application. The functions and features described in the previous sections of this document also apply.

The CSRmesh Lumicast application demonstrates the following use cases:

- CSRmesh Lumicast Beacon: The Lumicast application implements the Beacon Model functionality to transmit the Lumicast messages over the CSRmesh Beacon network. The lumicast application by default supports the below mentioned beacon types.
  - Lumicast Beacon Type

  The Lumicast commands are transmitted over the CSRmesh beacon network through as beacon payload information until they reach the destined device in the mesh network.

  The application implements the handlers for the CSRmesh messages related to the Beacon Model, Stream Model, LOT, Attention and Battery Models.
- CSRmesh GATT Bridge: The application also implements the CSR custom Mesh Control Service (see B.1 CSRmesh Bridge). This service allows a Bluetooth Smart enabled phone to send and receive CSRmesh messages to many devices with the Heater application acting as a bridge.

The application uses the CSRmesh library provided as part of the CSRmesh release. Refer to the *CSRmesh API Guide* documentation for details.

NOTE: The CSRmesh Lumicast application does not support bonding and is not compatible with the previous releases of CSRmesh.

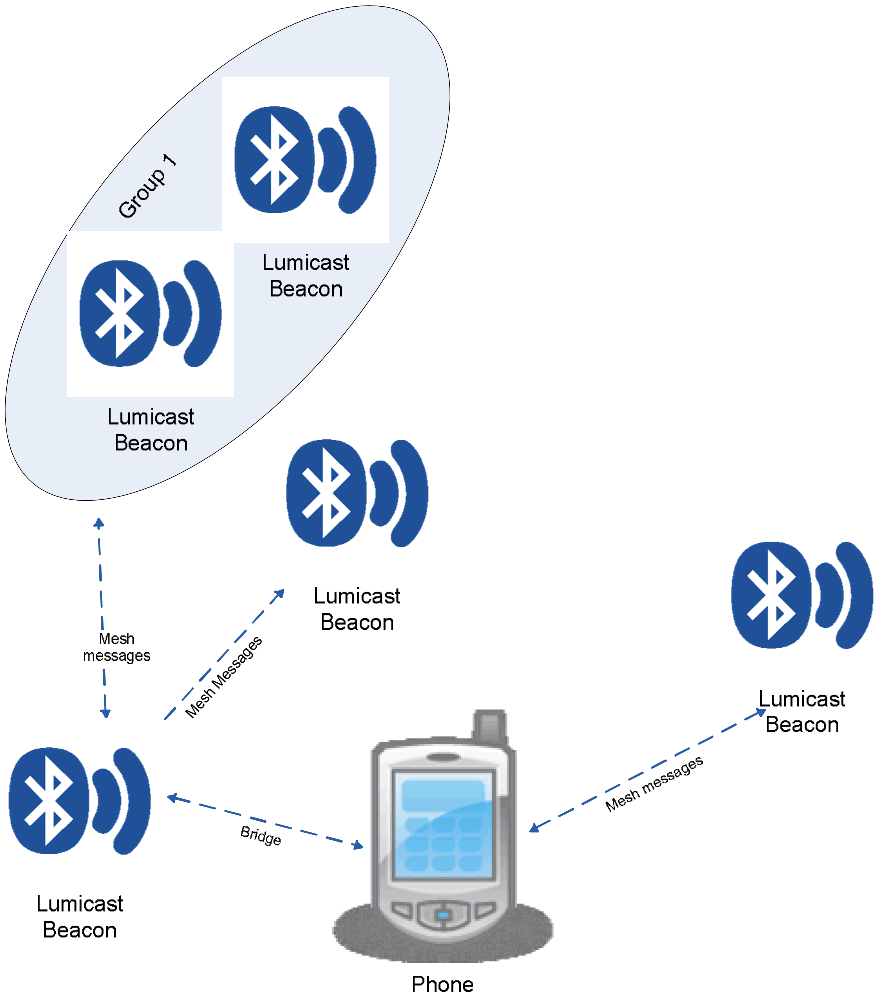


Figure B-8 CSRmesh Lumicast beacon use case

B.7.1 CSRmesh models supported by Beacon application

Table B-22 CSRmesh models supported by Beacon application

CSRmesh Model	Application Action
Beacon Model	Handles the beacon model messages. The application implements the provision of sending multiple beacon types over the LE channel at a time. The Beacon model also provides a way to update the beacon intervals as well as update the beacon payload over the CSRmesh network.
Attention Model	Handles the `CSR_MESH_ATTENTION_SET_STATE` command. When the attract attention is set the application blinks red.
Battery Model	Upon receiving the `CSR_MESH_GET_BATTERY_STATE` message, it reads the battery level and responds with the current battery level and the state. The application is implemented with reference to the CSRmesh development board, which runs on AA Batteries. So the application sets `BATTERY_MODEL_STATE_POWERING_DEVICE` indicating that the device is battery powered. If the battery level is below threshold it sets the `BATTERY_MODEL_STATE_NEEDS_REPLACEMENT` bit.
Data Stream Model	Enables the application to send and receive stream of bytes from another CSRmesh device.
Ping Model	The Ping model is used to send a reliable message delivery across the CSRmesh network.
LOT Model	Gives the Node application the ability to transfer large objects across nodes in the CSRmesh network.

NOTE: The CSRmesh Lumicast application supports groups for beacon, Data stream, LOT and attention models. It statically allocates memory to store the assigned group IDs and save them on the NVM.

B.7.2 Lumicast command

The Lumicast commands are transmitted over the CSRmesh network through the Lumicast beacon type payload. The lumicast commands are transmitted over the mesh network until the destination of the lumicast command is reached and the command is sent to the lumicast library for processing.

B.7.3 Lumicast application source files

Table 7-23 Lumicast application source files

File Name	Description
`main_app.c`	Implements all the entry functions such as `AppInit()`, `AppProcessSystemEvent()` and `AppProcessLmEvent()`. Handles events received from the hardware and firmware first. Contains handling functions for the system events.
`app_hw.c`	Implements the abstract hardware interface to configure and control the peripherals on specific hardware.
`app_mesh_handler.c`	Implements the application functions to communicate with the core mesh handlers.
`app_mesh_lumicast_handler.c`	Implements the application functions to communicate with the lumicast library. These functions are called from the beacon model when the new payload for the lumicast is received over the CSRmesh network.

B.7.4 Lumicast application header files

Table 7-3 Lumicast application header files

File Name	Description
`main_app.h`	Contains data structures and prototypes of externally referenced functions defined in the `main_app.c` file. It also contains the NVM structure defined for the application.
`app_conn_param.h`	Contains the connection parameters for various configurations.
`app_hw.h`	Contains the definitions for the hardware interface files for the application.
`app_mesh_handler.h`	Contains the definitions for the application interface functions implemented to communicate with the core mesh handlers.
`app_mesh_lumicast_handler.h`	Contains the definitions for the lumicast library interface functions implemented to communicate with the beacon model handlers.
`user_config.h`	Contains the models and the application configurations supported.

B.7.5 Lumicast application configuration files

Table 7-4 Lumicast application configuration files

File Name	Description
`app_gatt_db.db`	The database file containing the service database supported by the application.
`app_store_config_smem_a.stores`	The application smem configuration containing the size of the supported stores on the CSR102x platform.
`bootloader.keyr`	The bootloader keyr file containing the keys to be used by the bootloader for OTA update on the CSR101x platform
`csr_mesh_lumicast_csr101x.xiw &` `csr_mesh_lumicast_csr101x.xip`	The xide project files used by the SDK for building and flashing the application onto CSR101x hardware.
`csr_mesh_lumicast_csr101x_A05.keyr`	The keyr file containing the user keys supported by the application to be used on the CSR101x platform.
`csr_mesh_lumicast_csr101x_uenergyprops.xml`	The xml file containing the connection manager features for the supported application on CSR101x platform.
`csr_mesh_lumicast_csr102x.xiw &` `csr_mesh_lumicast_csr102x.xip`	The xide project files used by the SDK for building and flashing the application onto CSR102x hardware.
`csr_mesh_lumicast_csr102x.htf`	The htf file containing the user keys supported by the application to be used on the CSR102x platform.
`csr_mesh_lumicast_csr102x_uenergyprops.xml`	The xml file containing the connection manager features for the supported application on CSR102x platform.
`mesh_release.upd`	The file containing the partition information of the application on CSR102x platform

B.7.6 Lumicast application features

This section describes how to customize some parameters on the CSRmesh Lumicast application. The application can be configured by uncommenting the required define in `user_config.h`.

Table B-24 Application configuration

Configuration	Description
`ENABLE_DEVICE_UUID_ADVERTS`	If defined 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_BATTERY_MODEL`	Enables support for CSRmesh battery model.
`USE_AUTHORISATION_CODE`	Enforces Authorisation Code check on device during association. If this is enabled, the associating control device must have the same authorisation code to associate device to network. Enabled by default.
`ENABLE_DATA_MODEL`	Enables data stream model to send/receive stream of octets to/from another CSRmesh device. If this is enabled, the application supports streaming of device information string over the data model. The application uses a simple protocol to send and receive messages. See Section 4.6 for details.
`BEACON_TYPE_SUPPORTED ((1 << csr_mesh_beacon_type_lumicast))`	The supported beacon types in the application. The application developer can define any subset of the defined beacon types to be supported.
`MAX_BEACONS_SUPPORTED`	The Max beacons supported by the application
`MAX_BEACON_PAYLOAD_SIZE`	The maximum size of the payload for the beacons supported by the application
`APP_BEACON_MODE`	This definition is used to say the mode of the application for the beacon model handler to implement.

C CSRmesh application GATT database

C.1 GATT service characteristics

Table C-1 GATT service characteristics

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

C.2 GAP service characteristics

Table C-2 GAP service characteristics

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

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

C.3 CSR OTA update application service characteristics

Table C-3 CSR OTA update application service characteristics

Characteristic Name	Access Permissions	Managed By	Security Permissions	Value
Current Application	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	Write	Application	Security Mode 1 and Security Level 2	Format: uint16[2] Index 0: An offset in 16-bit words into the CS defined in the SDK documentation. Index 1: The size of the CS block expected in octets.
Data Transfer	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	Read Write	Application	Security Mode 1 and Security Level 2	Current client configuration for Data Transfer characteristic
Version	Read	Firmware	Security Mode 1 and Security Level 2	Service version Format: uint8

C.4 Mesh control service characteristics

Table C-4 Mesh control service characteristics

Characteristic Name	Access Permissions	Managed By	Security Permissions	Value
Network Key	Write	Application	Security Mode 1 and Security Level 1	0
Device UUID	Read	Application	Security Mode 1 and Security Level 1	`22e4-b12c-5042-11e3-9618-ce3f-5508-acd9`
Device ID	Read Write	Application	Security Mode 1 and Security Level 1	`0x8001`
MTL Continuation Control Point	Write	Application	Security Mode 1 and Security Level 1	Dynamic
MTL Continuation Control Point Client Characteristic Configuration	Read Write	Application	Security Mode 1 and Security Level 1	Current client configuration for MTL Continuation Control Point characteristic
MTL Complete Control Point	Write Notify	Application	Security Mode 1 and Security Level 1	Dynamic

Characteristic Name	Access Permissions	Managed By	Security Permissions	Value
MTL Complete Control Point Client Characteristic Configuration	Read Write	Application	Security Mode 1 and Security Level 1	Current client configuration for MTL Complete Control Point characteristic
MTL TTL	Read Write	Application	Security Mode 1 and Security Level 1	50
MESH Appearance	Read Write	Application	Security Mode 1 and Security Level 1	0

C.5 GAIA service characteristics

Table C-5 GAIA service characteristics

Characteristic Name	Access Permissions	Managed By	Security Permissions	Value
GAIA Command Endpoint	Write	Application	Security Mode 1 and Security Level 2	This characteristic is used by the Host to send GAIA packets to the Device.
GAIA Response Endpoint	Read Notify	Application	Security Mode 1 and Security Level 2	This characteristic is used by the Device to send GAIA packets to the Host.
GAIA Data Endpoint	Write Read	Application	Security Mode 1 and Security Level 2	This characteristic is used by Android applications to determinate if pairing is required.

D Document references

Document	Reference
Bluetooth Core Specification Version 4.1	www.bluetooth.org/
µEnergy Modifying an Application to Support OTA Update Application Note	CS-304564-AN
µEnergy Over-the-Air (OTA) Update System Application Note	CS-316019-AN
µEnergy xIDE User Guide	CS-212742-UG
CSRmesh 2.0 Node API Guide	www.csrsupport.com
Service Characteristics And Descriptions	developer.bluetooth.org
CSRmesh 2.1 Admin Console User Guide	CS-346130-DC
CSRmesh 2.1 Beacon and Tracker Gateway Application Note	CS-348610-UG
CSRmesh 2.1 Cloud Deployment Guide	CS-341394-DC
CSRmesh 2.1 Console Apps Release Note	CS-348395-RN
CSRmesh 2.1 Gateway NB Build and Installation User Guide	CS-343479-DC
CSRmesh 2.1 Gateway North Bound Release Note	CS-348394-RN
CSRmesh 2.1 Gateway SB Security Feature Integration Guide	CS-347536-UG
CSRmesh 2.1 Gateway Southbound User Guide	CS-347501-UG
CSRmesh 2.1 User Guide	CS-335262-UG
CSRmesh Android Controller Application Note	CS-347470-AN
CSRmesh Application Porting Guide	CS-348721-DC
CSRmesh Cloud and Gateway Platforms Architecture	CS-346475-TC
CSRmesh Cloud Release Note	CS-348393-RN
CSRmesh Console User Guide	CS-346131-DC
CSRmesh Gateway Release Note	CS-347500-RN
CSRmesh Gateway Southbound API Specification	www.csrsupport.com
CSRmesh Home Application Note	CS-348212-AN
CSRmesh iOS Controller Application Note	CS-347471-AN
CSRmesh Mobile Application User Guide	CS-347469-UG
CSRmesh Mobile Applications Release Note	CS-347473-RN
CSRmesh Node API Specification	www.csrsupport.com
CSRmesh Node Release Note	CS-348818-RN
CSRmesh Production Tool User Guide	CS-335123-UG
CSRmesh REST API Online Help	www.csrsupport.com
CSRMesh Sniffer Application Note	CS-348720-AN
CSRmesh™ LOT Android application Release Notes	CS-348755-RN
CSRmesh™ LOT Android application User Guide	CS-348753-UG
CSRmesh™ LOT OTAU iOS Application Release Notes	CS-348756-RN

Document	Reference
CSRmesh™ LOT OTAU iOS Application User Guide	CS-348754-UG
GAIA OTAU Android application 1.1.12 release note	CS-348777-RN
GAIA OTAU Android application 1.1.12 user guide	CS-348775-UG
QBeacon Application Release Notes	CS-347474-RN
QBeacon Mobile Application User Guide	CS-347464-UG

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