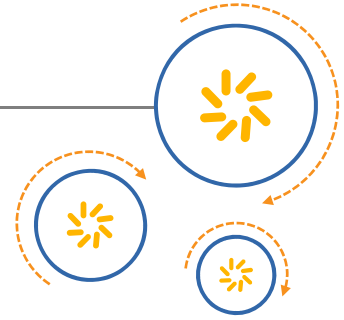




Qualcomm Technologies International, Ltd.



Confidential and Proprietary – Qualcomm Technologies International, Ltd.

(formerly known as Cambridge Silicon Radio Ltd.)

NO PUBLIC DISCLOSURE PERMITTED: Please report postings of this document on public servers or websites to:
DocCtrlAgent@qualcomm.com.

Restricted Distribution: Not to be distributed to anyone who is not an employee of either Qualcomm Technologies International, Ltd. or its affiliated companies without the express approval of Qualcomm Configuration Management.

Not to be used, copied, reproduced, or modified in whole or in part, nor its contents revealed in any manner to others without the express written permission of Qualcomm Technologies International, Ltd.

Any software provided with this notice is governed by the Qualcomm Technologies International, Ltd. Terms of Supply or the applicable license agreement at <https://www.csrsupport.com/CSRTermsandConditions>.

Qualcomm is a trademark of Qualcomm Incorporated, registered in the United States and other countries. All Qualcomm Incorporated trademarks are used with permission. Other product and brand names may be trademarks or registered trademarks of their respective owners.

This technical data may be subject to U.S. and international export, re-export, or transfer ("export") laws. Diversion contrary to U.S. and international law is strictly prohibited.

© 2015 Qualcomm Technologies International, Ltd. All rights reserved.

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



Push every boundary.™

CSR μ Energy™



CSRmesh 2.0 Heater Application

Application Note

Issue 3

Document History

Revision	Date	History
1	30 SEP 15	Original publication of this document.
2	27 OCT 15	Editorial updates.
3	30 OCT 15	Editorial updates

Contacts

General information

www.csr.com

Information on this product

sales@csr.com

Customer support for this product

www.csrsupport.com

More detail on compliance and standards

product.compliance@csr.com

Help with this document

comments@csr.com

Trademarks, Patents and Licences

Unless otherwise stated, words and logos marked with [™] or ® are trademarks registered or owned by CSR plc and/or its affiliates.

Bluetooth® and the Bluetooth logos are trademarks owned by Bluetooth SIG, Inc. and licensed to CSR.

CSRmesh is a product owned by Qualcomm Technologies International, Ltd

Other products, services and names used in this document may have been trademarked by their respective owners.

The publication of this information does not imply that any licence is granted under any patent or other rights owned by CSR plc or its affiliates.

CSR reserves the right to make technical changes to its products as part of its development programme.

While every care has been taken to ensure the accuracy of the contents of this document, CSR cannot accept responsibility for any errors.

Use of this document is permissible only in accordance with the applicable CSR licence agreement.

Safety-critical Applications

CSR's products are not designed for use in safety-critical devices or systems such as those relating to: (i) life support; (ii) nuclear power; and/or (iii) civil aviation applications, or other applications where injury or loss of life could be reasonably foreseeable as a result of the failure of a product. The customer agrees not to use CSR's products (or supply CSR's products for use) in such devices or systems.

Performance and Conformance

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

Contents

Document History	2
Contacts	2
Trademarks, Patents and Licences	2
Safety-critical Applications	2
Performance and Conformance	2
Contents	3
Tables, Figures and Equations	3
1. Introduction	5
1.1. Application Overview	6
2. Using the Application	9
2.1. Demonstration Kit	9
3. Application Structure	12
3.1. Source Files	12
3.2. Header Files	13
3.3. Database Files	14
4. Code Overview	15
4.1. Application Entry Points	15
4.2. Internal State Machine for GATT Connection	19
4.3. CSRmesh Association	21
4.4. Supported CSRmesh Models	23
4.5. Synchronising with CSRmesh Activity	23
4.6. Bearer State Management	24
4.7. Low Power Operation	24
4.8. Application Data Stream Protocol	25
5. NVM Map	26
5.1. Application NVM Version	26
6. Customising the Application	28
6.1. Advertisement Timers	28
6.2. Connection Parameters	28
6.3. Number of Supported Model Groups	28
6.4. Device Name	28
6.5. Device Address	29
6.6. Non-Volatile Memory	29
6.7. Application Features	29
6.8. Configuring the CSRmesh Parameters	31
Appendix A CSRmesh Application GATT Database	32
Document References	35
Terms and Definitions	36

Tables, Figures and Equations

Table 1.1: CSRmesh Control Profile Roles	6
Table 1.2: Application Topology	6
Table 1.3: Role and Responsibilities	7
Table 2.1: CSRmesh Components	9
Table 2.2: CSRmesh Development Board User Interface	10
Table 3.1: Source Files	12
Table 3.2: Header Files	14

Table 3.3: Database Files	14
Table 4.1: Supported CSRmesh Models	23
Table 4.2: Application Data Stream Message Format	25
Table 4.3: Application Data Stream Messages	25
Table 5.1: NVM Map for Application	26
Table 5.2: NVM Map for GAP Service	26
Table 6.1: Advertisement Timers	28
Table 6.2: Connection Parameters	28
Table 6.3: Number of Supported Model Groups	28
Table 6.4: Application Configuration	30
Table 6.5: Configuring CSRmesh Parameters	31
Table A.1: GATT Service Characteristics	32
Table A.2: GAP Service Characteristics	32
Table A.3: CSR OTA Update Application Service Characteristics	33
Table A.4: Mesh Control Service Characteristics	34
Figure 1.1: CSRmesh Temperature Sensor Use Case	5
Figure 1.2: Primary Services	7
Figure 2.1: CSRmesh Development Board	9
Figure 2.2: CSRmesh Device Tag Sticker	10
Figure 4.1: Internal GATT State Machine	19
Figure 4.2: CSRmesh Association State Machine	21
Figure 4.3: CSRmesh QR Code Scanner Application	22

1. Introduction

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

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

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

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

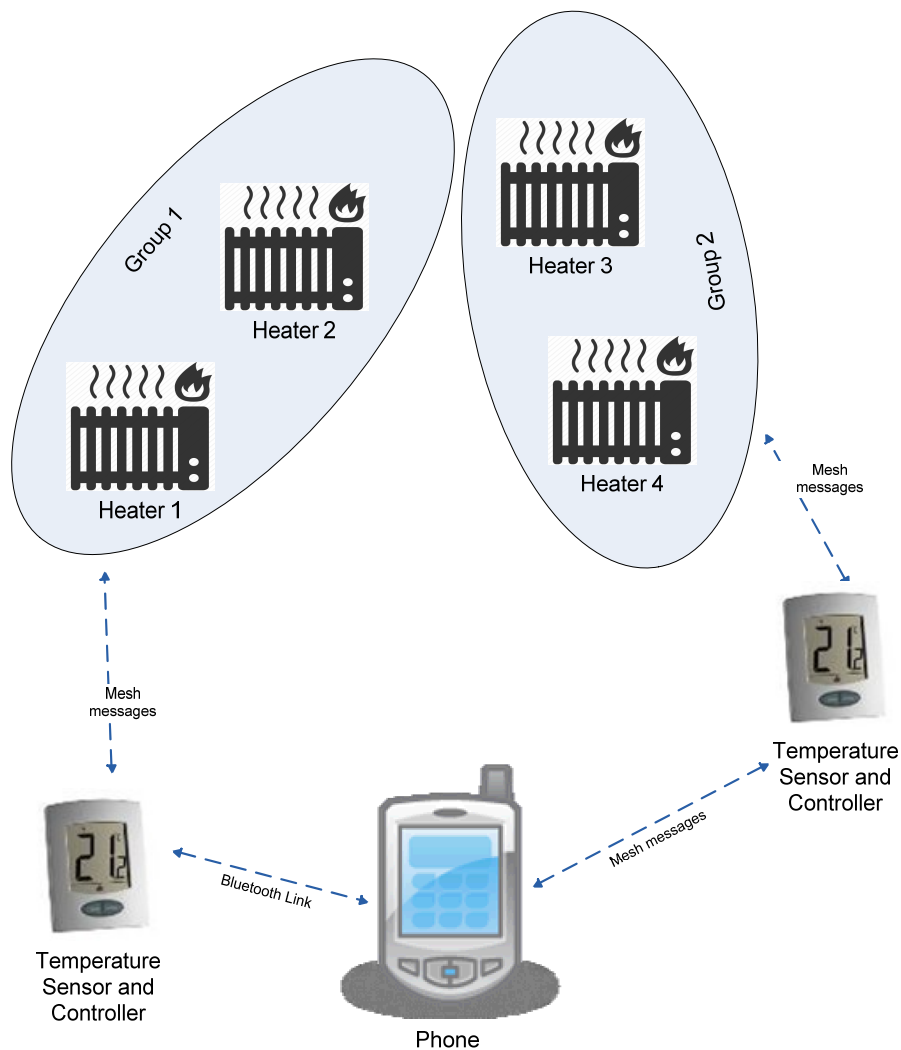


Figure 1.1: CSRmesh Temperature Sensor Use Case

1.1. Application Overview

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

1.1.1. Profiles Supported

The CSRmesh Heater Application implements the following CSR custom profiles to support the use cases described.

1.1.1.1. CSRmesh Control Profile

The CSRmesh Control Profile defines the behaviour when:

- A network of devices such as temperature sensors and heaters need to be created.
- Switching the heater on/off based on the desired and current air temperature values received from the group.
- Reading the status of the devices in the network, for example desired and current air temperatures currently set on the network.

Table 1.1 lists the roles defined by CSRmesh Control Profile.

Role	Description	Implementation
CSRmesh Bridge Device	Receives commands from the 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 via a Bluetooth Smart connection.	On the CSRmesh Heater application
CSRmesh Control Device	The CSRmesh Control Device provides the interface to create a network of devices and control the associated devices. The control commands are sent via a Bluetooth Smart connection to the CSRmesh devices.	On a Bluetooth Smart enabled phone or tablet

Table 1.1: CSRmesh Control Profile Roles

1.1.2. Application Topology

Table 1.2 and Table 1.3 list the topology used by the CSRmesh Bridge.

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

Table 1.2: Application Topology

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

Table 1.3: Role and Responsibilities

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

1.1.3. Services Supported in GATT Server Role

The application exposes the following services:

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

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

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

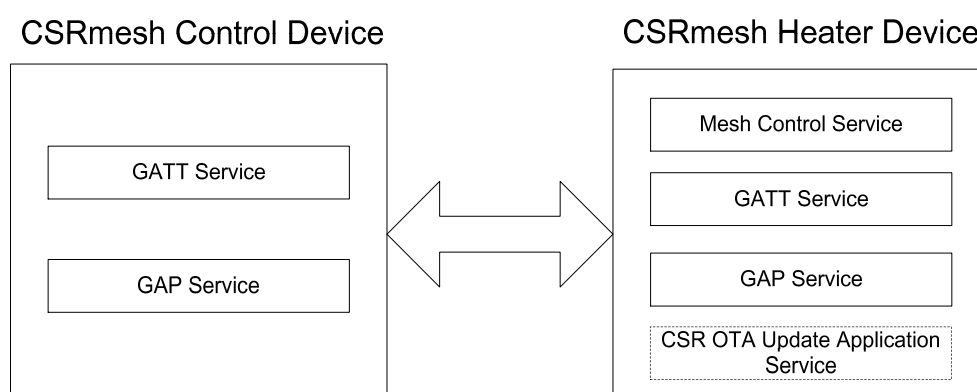


Figure 1.2: Primary Services

1.1.3.1. CSR OTA Update Application Service

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

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

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

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

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

For more information on CSR OTA Update, see

- CSR μ Energy Over-the-Air (OTA) Update System Application Note
- CSR μ Energy Modifying an Application to Support OTA Update Application Note



- CSR μ Energy Over-the-Air (OTA) Update Application and Bootloader Services Specification

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

2. Using the Application

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

2.1. Demonstration Kit

Table 2.1 lists the components that demonstrate the application.

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

Table 2.1: CSRmesh Components

2.1.1. CSRmesh Development Board

The µEnergy SDK downloads the CSRmesh Heater application on the development boards. See *CSR µEnergy xIDE User Guide* for more information.

Ensure the development board is switched on using the Power Slider switch. Figure 2.1 shows the switch in the **Off** position.

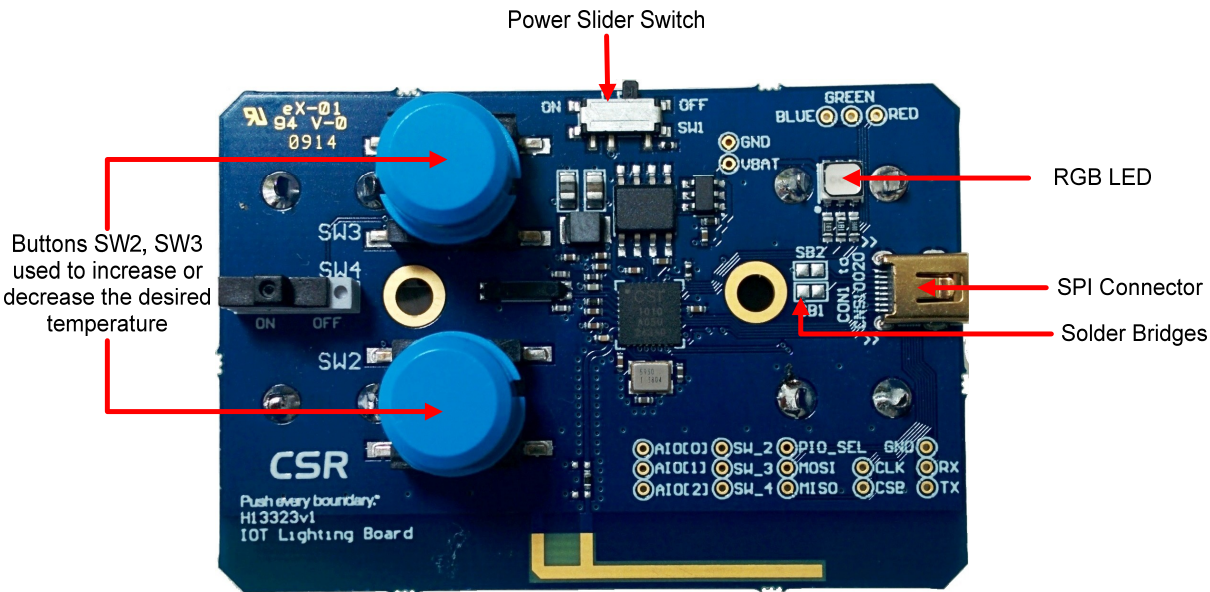


Figure 2.1: CSRmesh Development Board

Note:

When disconnected from the USB to SPI Adapter, wait at least 1 minute before switching the board on. This allows any residual charge received from the SPI connector to be dissipated.

Shorting the solder bridges allows the UART to be brought out via the CSR SPI connector. The PIOs connected to SW2 and SW3 are also mapped to UART Rx and Tx lines. Pressing any of these buttons shorts the UART lines to ground and corrupts the data on the UART. It is recommended not to press these buttons during UART communication.

2.1.1.1. User Interface

This application uses the components available on the CSRmesh development board for the CSRmesh Heater applications. Table 2.2 lists the components of the board user interface shown in Figure 2.1.

User Interface Component	Function
Switch SW1	Power slider switch allows the user to power on/off the board.
Button SW2	Each button press increases the current set desired temperature by 1 degree. If the button is pressed and held for more than 1 second the temperature is changed by 1 degree every 1 second as long as the button is pressed. See Section 6.7 for details.
Button SW3	Each button press decreases the current set desired temperature by 1 degree. If the button is pressed and held for more than 1 second the temperature is changed by 1 degree every 1 second as long as the button is pressed. See Section 6.7 for details.
Switch SW4	Unused.
RGB LED	Blinks blue until it is not associated with any CSRmesh network. Blinks yellow when the device association is in progress. Blinks red when device attention is requested. The application turns the light off after association. Turns red to indicate Heater status changes to be on. Turns off if the heater status changes to be off.

Table 2.2: CSRmesh Development Board User Interface

2.1.1.2. CSRmesh Device Tag

The CSRmesh development board is supplied with a CSRmesh device tag sticker as shown in Figure 2.1. The sticker contains:

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



Figure 2.2: CSRmesh Device Tag Sticker

The user can program the device with the Bluetooth Address and the XTAL Trim printed on the sticker by setting these values in the `CSRmesh_heater_csr101x_A05.keyr` file.

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

To program the example UUID `0x0123456789ABCDEF` and Authorisation Code `0x0123456789ABCDEF` can be programmed on the NVM:

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


```
<CSR_uEnergy_Tools path>\uEnergyProdTest.exe -k
CSRmesh_heater_csr101x_A05.keyr -m1 0x02 0x3210 0x7654 0xBA98 0xFEDC
0xCDEF 0x89AB 0x4567 0x0123 0xCDEF 0x89AB 0x4567 0x0123
```

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


```
<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 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 Authorisation Code and the UUID from the QR-Code printed on the sticker during association. See *CSRmesh 2.0 Android Control Application Note* or *CSRmesh 2.0 iOS Control Application Note* for details. For further information about programming the NVM, see *CSRmesh 2.0 Production Test Tool User Guide*.

2.1.2. CSRmesh Control Application

The CSRmesh Control application runs on an Android or iOS device that supports Bluetooth Low Energy. It communicates with the CSRmesh devices by connecting to one of the devices that support the CSR custom CSRmesh Control Profile. This application is required for:

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

Note:

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

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

For details on the supported iOS and Android version, see *CSRmesh 2.0 Mobile Application Release Note*.

3. Application Structure

3.1. Source Files

Table 3.1 lists the source files and their purposes.

File Name	Purpose
app_data_stream.c	Handles data stream model events. Implements a protocol to exchange large blocks of data with other CSRmesh devices. See Section Error! Reference source not found. for more details.
app_fw_event_handler.c	Defines the handler functions for firmware events.
app_mesh_event_handler.c	Defines the handler functions for CSRmesh events.
battery_hw.c	Implements the routines to read the battery level.
csr_mesh_heater.c	Implements all the entry functions such as AppInit(), AppProcessSystemEvent(), AppProcessCsrMeshEvent() and AppProcessLmEvent(). Events received from the hardware, CSRmesh network and firmware are first handled here. This file contains handling functions for all the LM, CSRmesh and system events.
csr_mesh_heater_gatt.c	Implements routines for triggering advertisement procedures.
csr_mesh_heater_util.c	Implements some utilities functions called by other handlers.
csr_mesh_tempsensor_util.c	Implements some utilities functions called by other handlers.
csr_ota_service.c	Implements the routines for handling the read/write access on the CSR custom OTA Update Service.
gap_service.c	Implements routines for GAP service, for example, handling read/write access indications on the GAP service characteristics, reading/writing device name on NVM.
gatt_service.c	Defines routines for using the GATT service.
iot_hw.c	Implements the hardware interface to configure and control the peripherals on the CSRmesh development board.
mesh_control_service.c	Implements routines for handling read/write access on Mesh Control characteristics and for sending notifications of CSRmesh device responses.
nvm_access.c	Implements the NVM read/write routines.

Table 3.1: Source Files

3.2. Header Files

Table 3.2 lists the header files and their purposes.

File Name	Purpose
app_data_stream.h	Contains enumerations and function prototypes for externally referred functions defined in app_data_stream.c.
app_debug.h	Contains macro definitions for enabling debug prints.
app_fw_event_handler.h	Contains prototypes of the externally referred functions defined in app_fw_event_handler.c.
app_gatt.h	Contains macro definitions, user defined data type definitions and function prototypes used across the application.
app_mesh_event_handler.h	Contains prototypes of the externally referred functions defined in app_mesh_event_handler.c.
appearance.h	Contains the appearance value macro of the application.
battery_hw.h	Contains prototypes of the externally referred functions defined in battery_hw.c.
csr_mesh_heater.h	Contains the macros and definitions used in the csr_mesh_heater.c application file.
csr_mesh_heater_gatt.h	Contains macro for setting advertisement parameters.
csr_mesh_heater_util.h	Contains prototypes of the externally referred functions defined in the csr_mesh_heater_util.c file.
csr_ota_service.h	Contains prototypes of the externally referred functions defined in the csr_ota_service.c file.
csr_ota_uuids.h	Contains macros for UUID values for CSR OTA Update service.
gap_conn_params.h	Contains macro definitions for fast/slow advertising, preferred connection parameters, idle connection timeout values etc.
gap_service.h	Contains prototypes of the externally referred functions defined in the gap_service.c file.
gap_uuids.h	Contains macros for UUID values for GAP service.
gatt_service.h	Contains prototypes of the externally referred functions defined in the gatt_service.c file.
gatt_service_uuids.h	Contains macros for UUID values for GATT service.
iot_hw.h	Contains the function declarations to control the IoT hardware interfaces.
mesh_control_service.h	Contains prototypes of the externally referred functions defined in the mesh_control_service.c file.
mesh_control_service_uuids.h	Contains macros for UUID values for Mesh Control service.
ota_customisation.h	Contains defines for the application data variables used by csr_ota_service.c.
user_config.h	Contains macros for customising the application.

Table 3.2: Header Files

3.3. Database Files

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

Table 3.3 lists the database files and their purposes.

File Name	Purpose
app_gatt_db.db	Masters database file which includes all service specific database files. This file is imported by the GATT database generator.
csr_ota_db.db	Contains information related to CSR OTA Update service characteristics, their descriptors and values. See Table A.3 for CSR OTA service characteristics.
gatt_service_db.db	Contains information related to GATT service characteristics, their descriptors and values. See Table A.1 for GATT service characteristics.
gap_service_db.db	Contains information related to GAP service characteristics, their descriptors and values. See Table A.2 for GAP service characteristics.
mesh_control_service_db.db	Contains information related to CSRmesh Control service characteristics, their descriptors and values. See Table A.4 for CSRmesh Control service characteristics.

Table 3.3: Database Files

4. Code Overview

This section describes significant functions of the application.

4.1. Application Entry Points

4.1.1. ApplInit()

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

- Initialises the application timers, hardware and application data structures
- Configures GATT entity for server role
- Configures the NVM manager to use I²C EEPROM
- Initialises all the services
- Reads the persistent store. Sets a random device UUID based on the Bit 3 of `CS_USER_KEY` 1 setting when the application runs for the first time
- Initialises the CSRMesh stack:
 - Configures CSRMesh bearer parameters and initialises the CSRMesh scheduler:
`CSRSchedSetConfigParams (params)`
`CSRSchedStart ()`
 - Registers an application callback function for handling core mesh events:
`CSRMeshRegisterAppCallback (CSRMeshAppProcessMeshEvent)`
 - Initialises the core stack: `CSRMeshInit ()`
 - Initialises the supported model server and client modules
- Registers the attribute database with the firmware

4.1.2. AppProcessLmEvent()

This function is invoked whenever the system receives a LM-specific event. It handles the following events:

4.1.2.1. Database Access

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

4.1.2.2. LS Events

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

4.1.2.3. LM Events

- `LM_EV_ADVERTISING_REPORT`: This event is received when an advertisement packet is received. This can be a CSRMesh advertisement. The application passes the event data to the CSRMesh library to process the packet by calling the `CSRSchedHandleIncomingData ()` API.

- **LM_EV_CONNECTION_COMPLETE:** This event is received when the connection with the master is considered to be complete and includes the new connection parameters.
- **LM_EV_DISCONNECT_COMPLETE:** This event is received on link disconnection. Disconnection can be due to link loss, locally triggered or triggered by the remote connected device.
- **LM_EV_ENCRYPTION_CHANGE:** This event indicates a change in the link encryption.
- **LM_EV_CONNECTION_UPDATE:** This event indicates that the connection parameters are updated to a new set of values and is generated when the connection parameter update procedure is initiated by either the master or the slave. These new values are stored by the application for comparison against the preferred connection parameter, see Section 6.2.

4.1.2.4. SMP Events

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

4.1.2.5. GATT Events

- **GATT_CONNECT_CFM:** This confirmation event indicates that the connection procedure completes. If the connection does not complete successfully, the application goes to idle state and waits for a user action. See Section **Error! Reference source not found.** for more information on application states.
- **GATT_CANCEL_CONNECT_CFM:** This confirmation event confirms the cancellation of connection procedure. When the application stops advertisements to changes advertising parameters or to save power, this signal confirms the successful stopping of advertisements by the CSRmesh application.

4.1.3. AppProcessSystemEvent()

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

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

4.1.4. CSRmeshAppProcessMeshEvent ()

This function handles the CSRmesh core stack events received from the CSRmesh network or internal state changes.

4.1.4.1. Network Association Messages

- **CSR_MESH_ASSOC_STARTED_EVENT:** This event is received when a CSRmesh Control application sends an association request to a light that is ready for association. The application starts blinking yellow to indicate association in progress.
- **CSR_MESH_KEY_DISTRIBUTION:** This event is received when the CSRmesh Control device provides the network key for all future messages to communicate on the network. The application switches the state to associate and switches off 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 only relays 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 continues till timeout or till 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. The application uses this event to clean up the model data and then sets the device to be ready for association state.

- `CSR_MESH_BEARER_STATE_EVENT`: This message is received when a configuring device sends a bearer state change message.

4.1.4.2. Device configuration Messages

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

4.1.4.3. Device Information Messages

The application receives `CSR_MESH_OPERATION_REQUEST_FOR_INFO` for the following events, which have sub-events related to device information.

- `CSR_MESH_GET_VID_PID_VERSION_EVENT`: This message is received when configuring device requests for Vendor Identifier, Product Identifier and Version number information from the device. The application sends this information to the library for transmission.
- `CSR_MESH_GET_DEVICE_APPEARANCE_EVENT`: This message is received when configuring device requests for device appearance information from the device. The application sends this information to the library for transmission.

4.1.4.4. Group Model Messages

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

4.1.4.5. Bearer Model Messages

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

4.1.5. AppSensorEventHandler

This function handles the CSRmesh Sensor model events received from the CSRmesh network, such as reading of the sensor state, value or any other responses for Sensor model commands sent over CSRmesh.

- `CSRMESH_SENSOR_GET_TYPES`: This event is received when another device in the network sends a command to get the supported sensor types in the sensor model.
- `CSRMESH_SENSOR_VALUE/ CSRMESH_SENSOR_WRITE_VALUE`: This event is received when a CSRmesh Temperature Sensor application broadcasts the current air temperature and the desired temperature. The application updates the current and the desired temperature value with the received value and returns the sensor state data.
- `CSRMESH_SENSOR_WRITE_VALUE_NO_ACK`: This event is received when a CSRmesh Temperature Sensor application broadcasts the current air temperature and the desired temperature. The application updates the current and the desired temperature value.

4.1.6. AppBatteryEventHandler

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

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

4.1.7. AppAttentionEventHandler

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

- `CSRMESH_ATTENTION_SET_STATE`: This 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 colour when the Attention state is reset.

4.1.8. AppDataServerHandler

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

- `CSRMESH_DATA_STREAM_FLUSH`: This message is received when a data stream client wants to start sending a data stream or to indicate completion of data transfer 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 streamed message and stores it in the device information string.
- `CSRMESH_DATA_BLOCK_SEND`: This message is received when a data stream client sends a single block of data.

4.1.9. AppDataClientHandler

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

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

4.2. Internal State Machine for GATT Connection

This section describes the different state transition in the application during connection. Figure 4.1 shows the state transition process.

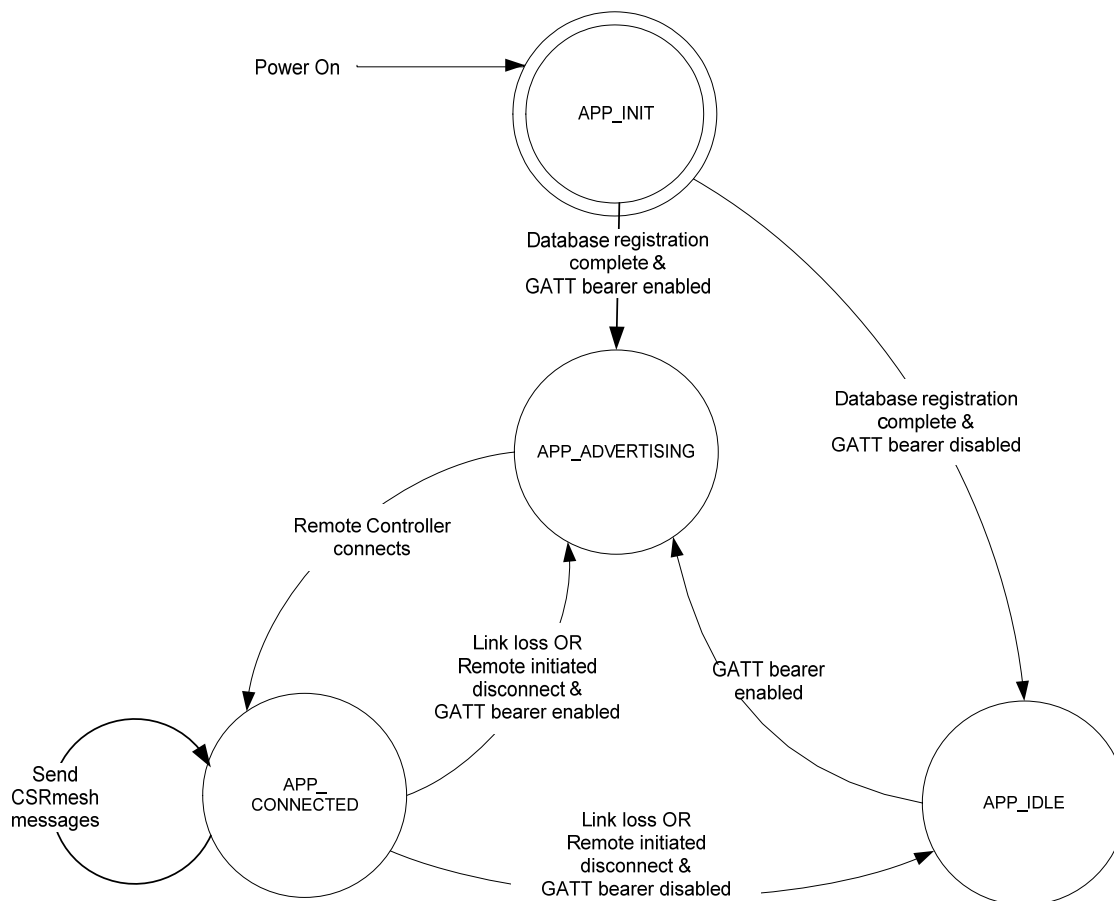


Figure 4.1: Internal GATT State Machine

4.2.1. app_state_init

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

4.2.2. app_state_advertising

The application starts in this state and transmits connectable advertising events at an interval defined by `ADVERT_INTERVAL`. When a central device connects to the application, the application stops advertising and enters the `app_state_connected` state. See Section 6.1 for more information on advertisement timers. If the GATT bearer is disabled in this state then the application moves to the `app_state_idle` state.

4.2.3. app_state_idle

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

4.2.4. app_state_connected

In this 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 and the GATT bearer is enabled then the application switches to the `app_state_advertising` state; otherwise the application moves to the `app_state_idle` state.
- In case of a remote triggered disconnection, the application switches to the `app_state_advertising` state if the GATT bearer is enabled; otherwise the application moves to the `app_state_idle` state.

4.2.5. app_state_disconnecting

The CSRmesh application never triggers a disconnection on its own. If the disconnection is triggered, the application enters the `app_state_advertising` state if the GATT bearer is enabled; otherwise it moves to the `app_state_idle` state.

4.3. CSRmesh Association

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

Figure 4.2 shows the application association state machine.

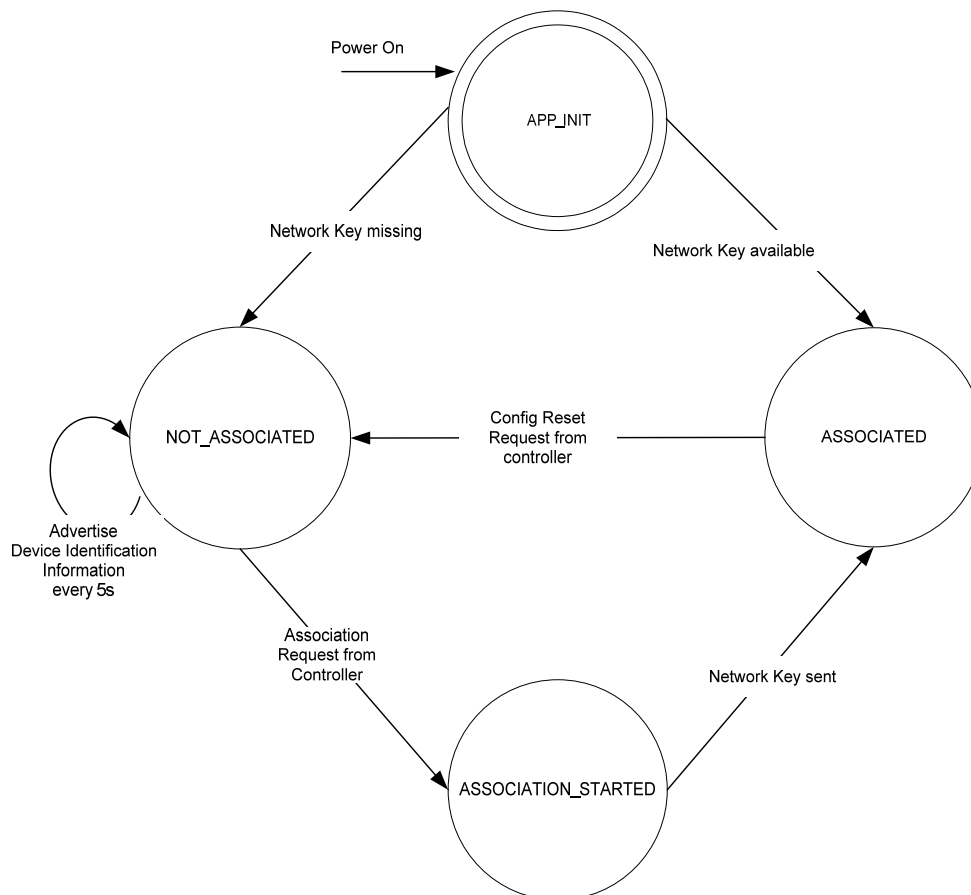


Figure 4.2: CSRmesh Association State Machine



Figure 4.3: CSRmesh QR Code Scanner Application

- `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 5 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 2 second long press of the SW2 button.

4.4. Supported CSRMesh Models

Table 4.1 lists the CSRMesh models that the Temperature Sensor application supports.

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 <code>CSR_MESH_ATTENTION_SET_STATE</code> command. When the attract attention is set the application blinks red.
Battery Model	Upon receiving the <code>CSR_MESH_GET_BATTERY_STATE</code> 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 <code>BATTERY_MODEL_STATE_POWERING_DEVICE</code> indicating that the device is battery powered. If the battery level is below threshold it sets the <code>BATTERY_MODEL_STATE_NEEDS_REPLACEMENT</code> bit.
Data Stream Model	Enables the application to send and receive stream of bytes from another CSRMesh device.

Table 4.1: Supported CSRMesh Models

Note:

The CSRMesh Heater application supports groups for sensor, actuator, data stream and attention models. It statically allocates memory to store the assigned group IDs and save them on the NVM.

4.5. Synchronising with CSRMesh Activity

The CSRMesh Heater application can connect to the CSRMesh Control application in a bridge device role. The Heater application has to synchronise with the CSRMesh library to avoid collision of the advertisements and connection events with the CSRMesh activity. The application calls the CSRMesh APIs to synchronise 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 CSRMesh stack of the GATT connection events to schedule the CSRMesh activity. The application calls `CSRSchedNotifyGattEvent(ucid, conn_interval)` for the CSRMesh library to synchronise 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 Heater application supports 2 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_STATE_EVENT` message or the CS User Key configuration. See Section **Error! Reference source not found.**
- 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. Low Power Operation

The heaters are expected to consume very low power because they are generally battery powered and have to give a good battery life. Scanning for CSRMesh messages by far 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.

4.7.1. 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.

4.7.2. Acknowledged Mode Message Broadcast

The application implements an acknowledged mode where it acknowledges the sensor write value messages received from the sensors in the same group. The application retransmits the acknowledgement multiple times because the sensor devices are listening in low duty cycle scan.

4.7.2.1. Multiple retransmissions

The sensor value messages are broadcast periodically as configured by the user. Because all the recipients of the sensor value messages are expected to run 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. The `RETRANSMIT_INTERVAL` defines the time at which the same sensor message needs to be re-transmitted. `MAX_RETRANSMISSION_TIME` defines the total time for each message retransmission. The number of retransmissions of each of the sensor value message can be calculated by $\text{MAX_RETRANSMISSION_TIME} / \text{RETRANSMIT_INTERVAL}$.

4.8. 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 4.3 to exchange messages.

Message Code (1 octet)	Length (1 or 2 octets)	Data (0 – 32767 octets)
Identifies type of message. See Table 4.3 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 Message Format

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.
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.

Table 4.3: Application Data Stream Messages

5. NVM Map

The application stores the parameters listed in Table 5.1 in the NVM to prevent loss in case of a power off or a chip panic.

Entity Name	Type	Size of Entity (Words)	NVM Offset (Words)
CSRMesh Stack NVM	uint16 array	32	0
CSRMesh Device UUID (This is part of 31 words of Stack NVM)	uint16 array	8	1
CSRMesh Device Authorisation Code (This is part of 31 words of Stack NVM)	uint16 array	4	9
Sanity Word	uint16	1	32
Application NVM Version	uint16	1	33
CSRMesh Association State	boolean	1	34
Sensor State Data	uint16	2	35
Sensor Actuator Model Group IDs	uint16 array	4	37
Attention Model Group IDs	uint16 array	4	41
Data Model Group IDs	uint16 array	4	45

Table 5.1: NVM Map for Application

Entity Name	Type	Size of Entity (Words)	NVM Offset (Words)
GAP Device Name Length	uint16	1	49
GAP Device Name	uint8 array	20	50

Table 5.2: NVM Map for GAP Service

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.

5.1. Application NVM Version

Before reading the parameters from the NVM in Table 5.1 and Table 5.2, the application reads NVM version at word offset 1. If the application updates and the version number stored on the NVM does not match with the defined `APP_NVM_VERSION`, the application re-initialises the NVM starting from Sensor and Actuator Model Group Id's and stores the defined `APP_NVM_VERSION` at word offset 1.

Note:

Device UUID, Authorisation Code, CSRMesh library data and association information are not reset. This ensures that the contents of the NVM are valid for the NVM offsets defined in the application. If the NVM offsets



of the parameters change upon an application update, this value can be incremented to invalidate the contents of the NVM.

6. Customising the Application

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

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

6.1. Advertisement Timers

The CSRMesh Heater application sends connectable advertisements as long as it is powered on and not connected. The application uses a timer to send a connectable advertising event at regular intervals. The advertising interval is defined in `csr_mesh_heater_gatt.h`.

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

Table 6.1: Advertisement Timers

6.2. Connection Parameters

The CSRMesh Heater application uses the connection parameters listed in Table 6.2 by default. The macros for these values are defined in `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. It is strongly recommended not to modify these parameters. If the connection interval is set to fewer than 13 ms, the device is not able to scan for CSRMesh messages from the associated network, and only messages received from the GATT connection are processed. See *Bluetooth Core Specification Version 4.1* for the connection parameter range.

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

Table 6.2: Connection Parameters

6.3. Number of Supported Model Groups

The CSRMesh application supports Group ID assignment for sensor, stream and attention models of CSRMesh. The maximum number of model groups supported by default is set to 4 in `user_config.h`.

Parameter Name	Parameter value
MAX_MODEL_GROUPS	4

Table 6.3: Number of Supported Model Groups

Note:

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

6.4. Device Name

The user can change the device name for the application. By default, it is set to CSRMesh in `gap_service.c`. The maximum length of the device name is 20 octets.

6.5. Device Address

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

6.6. Non-Volatile Memory

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

- `NVM_TYPE_EEPROM` for I²C 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 which NVM type it is built for. If EEPROM is selected, `NVM_TYPE_EEPROM` is defined and for SPI Flash the macro `NVM_TYPE_FLASH` is defined. Follow the comments in the `.keyr` file as per the selection above.

6.7. Application Features

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

Configuration	Description
<code>ENABLE_DEVICE_UUID_ADVERTS</code>	If defined the application sends UUID adverts periodically with an interval defined in <code>DEVICE_ID_ADVERT_TIME</code> . Otherwise the device UUID adverts are not sent. Disabled by default.
<code>ENABLE_BATTERY_MODEL</code>	Enables support for CSRmesh battery model.
<code>USE_AUTHORISATION_CODE</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.
<code>ENABLE_DATA_MODEL</code>	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 Error! Reference source not found. for details.
<code>DEBUG_ENABLE</code>	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.
<code>DEFAULT_RX_DUTY_CYCLE</code>	The percentage Rx duty cycle set by the application once the device is associated and grouped. Default: 2%.
<code>HIGH_RX_DUTY_CYCLE</code>	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%.

Configuration	Description
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 $\text{MAX_RETRANSMISSION_TIME} / \text{RETRANSMIT_INTERVAL}$.
ENABLE_ACK_MODE	This enables the acknowledged mode communication between the Temperature Sensor and Heater applications. If <code>ENABLE_ACK_MODE</code> is defined, the application stores and manages the group of heaters it is communicating with and replies the acknowledgement for the <code>SensorWriteValue</code> messages received from the Temperature Sensor applications. Disabled by default.

Table 6.4: Application Configuration

6.8. Configuring the CSRmesh Parameters

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

CS User Key Index	Parameter	Recommended Value	Description
0	CSRmesh Configuration Bitmask	0000 to 0007	<ul style="list-style-type: none"> Bit-0 : CSRmesh Relay Enable <ul style="list-style-type: none"> 1: Enables relay of CSRmesh messages 0: Disables relay of CSRmesh messages Bit-1 : CSRmesh Bridge Enable <ul style="list-style-type: none"> 0: Disables connectable advertisements 1: Enables connectable advertisements <p>Disabling bridge leaves the device unconnectable on any service. The connectable adverts can be enabled by sending a <code>CSR_MESH_BEARER_SET_STATE</code> message with the LE GATT server bearer bit set.</p> Bit-2 : CSRmesh Random Device UUID Enable <ul style="list-style-type: none"> 1: Enables generation of random device UUID 0: Reads the UUID from NVM <p>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.</p> Bits 3 to 15 are ignored.

Table 6.5: Configuring CSRmesh Parameters

Appendix A CSRmesh Application GATT Database

A.1 GATT Service Characteristics

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

Table A.1: GATT Service Characteristics

A.2 GAP Service Characteristics

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

Table A.2: GAP Service Characteristics

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

A.3 CSR OTA Update Application Service Characteristics

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

Table A.3: CSR OTA Update Application Service Characteristics

A.4 Mesh Control Service Characteristics

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

Table A.4: Mesh Control Service Characteristics

Document References

Document	Reference
<i>Bluetooth Core Specification Version 4.1</i>	www.bluetooth.org
<i>CSR μEnergy Modifying an Application to Support OTA Update Application Note</i>	CS-304564-AN
<i>CSR μEnergy Over-the-Air (OTA) Update System Application Note</i>	CS-316019-AN
<i>CSR μEnergy xIDE User Guide</i>	CS-212742-UG
<i>CSRmesh 2.0 Android Controller Application Note</i>	CS-337680-AN
<i>CSRmesh 2.0 Gateway SB User Guide</i>	CS-332701-UG
<i>CSRmesh 2.0 iOS Controller Application Note</i>	CS-337682-AN
<i>CSRmesh 2.0 Mobile Application User Guide</i>	CS-337051-UG
<i>CSRmesh 2.0 Production Test Tool User Guide</i>	CS-335123-UG
<i>CSRmesh Application 1.x to 2.0 Porting Guide</i>	CS-335300-DC
<i>CSRmesh 2.0 Release Note</i>	CS-334311-RN
<i>CSRmesh 2.0 Node API Guide</i>	www.csrsupport.com
<i>CSRmesh 2.0 Node Release Note</i>	CS-339050-RN
<i>GATT Database Generator</i>	CS-219225-UG
<i>Installing the CSR Driver for the Profile Demonstrator Application</i>	CS-235358-UG
<i>Over-the-Air Update Application and Bootloader Services Specification</i>	CS-316220-SP
<i>Service Characteristics And Descriptions</i>	developer.bluetooth.org

Terms and Definitions

AC	Authorisation Code
API	Application Programmer's Interface
BLE	Bluetooth Low Energy (now known as Bluetooth Smart)
BlueCore®	Group term for CSR's range of Bluetooth wireless technology chips
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.e.	<i>Id est</i> , that is
I²C	Inter-Integrated Circuit
IoT	Internet of Things
IRK	Identity Resolving Key
LED	Light Emitting Diode
LM	Link Manager
LS	Least Significant
MTL	Message Transport Layer
NVM	Non Volatile Memory
OTA	Over The Air
PCB	Printed Circuit Board
PIO	Programmable Input Output
PWM	Pulse Width Modulation
QR-Code	Quick Response Code
Rx	Receiver
SDK	Software Development Kit
SMP	Security Manager Protocol
SPI	Serial Peripheral Interface
Tx	Transmit
UART	Universal Asynchronous Receiver Transmitter



USB	Universal Serial Bus
UUID	Universally Unique Identifier