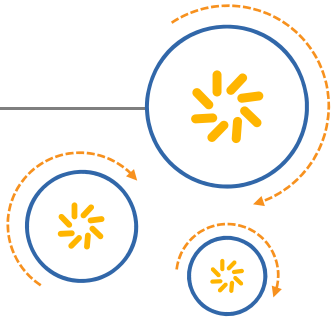
Qualcomm Technologies International, Ltd.



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.

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

Push every boundary.™

CSR uEnergy™

CSRmesh 2.0 Application Porting

User Guide

Issue 4

CS-335300-UGP4
www.csr.com

Document History

Revision	Date	History
1	30 SEP 15	Original publication of this document
2	23 OCT 15	Editorial updates
3	26 OCT 15	Editorial updates
4	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.

Life Support Policy and Use in Safety-critical Compliance

CSR's products are not authorised for use in life-support or safety-critical applications. Use in such applications is done at the sole discretion of the customer. CSR will not warrant the use of its devices in such applications.

Performance and Conformance

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

Contents

Tables

1. Introduction

This document describes the CSRmesh™ on-chip application porting from CSRmesh 1.x to CSRmesh 2.0 releases built using the CSR µEnergy® SDK.

Further the document describes the modifications in the application structure as well as changes to be done on various modules described in the forthcoming sections.

2. File Organisation Changes

This section explains the file organisational changes in the application between the 1.x applications to the 2.0 applications. Table 2.1: Light Example Application File Structure provide a file to file comparison of changes from the previous release.

CSRmesh 1.x Application Files	CSRmesh 2.x Application Files	Purpose (Changes from 1.3 to 2.0)
appearance.h	appearance.h	Contains the appearance value macro of the application.
app_data_stream.c	app_data_stream.c	This file maintains the stream state machine. The Data model on CSRmesh 2.0 does not process the STREAM_SEND messages and notifies the application as is.
app_data_stream.h	app_data_stream.h	The Function prototypes for using the data model have been changed.
app_debug.h	app_debug.h	No Change
app_gatt.h	app_gatt.h	No Change
app_gatt_db.db	app_gatt_db.db	No Change
battery_hw.c	battery_hw.c	Battery low status report added.
battery_hw.h	battery_hw.h	New function prototypes are added to support the Battery low status.
CSRMeshLight.xip	CSRmeshLight.xip	▪ New files included in project. Removed deprecated files. ▪ Updated for 2.5.0.20 SDK.
CSRMeshLight.xiw	CSRmeshLight.xiw	No change
csr_mesh_light.c	csr_mesh_light.c	This file has been reorganised into multiple files to categorise different handler functions into separate files. This file implements only the basic functions called by firmware and does application initialisation.
	app_fw_event_handler.c	Implements all the GATT related firmware event handlers.
	app_mesh_event_handler.c	Implements all the CSRmesh event (core mesh and model) handlers. See section 0 for details
	csr_mesh_light_util.c	Implements the utility functions which are generally required by different application handlers.
csr_mesh_light.h	csr_mesh_light.h	The public macro definitions are moved here

CSRmesh 1.x Application Files	CSRmesh 2.x Application Files	Purpose (Changes from 1.3 to 2.0)
		from csr_mesh_light.c file.
	app_fw_event_handler.h	Contains prototypes for all the externally referred GATT related firmware event handlers.
	app_mesh_event_handler.h	Contains prototypes for all the externally referred CSRmesh event (core mesh and model) handlers.
	csr_mesh_light_util.h	Contains the prototypes of the utility functions which are called by different application handlers.
csr_mesh_light_gatt.c	csr_mesh_light_gatt.c	No major changes except that the API's for communicating with the CSRmesh stack have been changed accordingly.
csr_mesh_light_gatt.h	csr_mesh_light_gatt.h	No change
csr_mesh_light_hw.c	csr_mesh_light_hw.c	No change
csr_mesh_light_hw.h	csr_mesh_light_hw.h	No change
csr_ota_db.db	csr_ota_db.db	No change
csr_ota_service.c	csr_ota_service.c	No change
csr_ota_service.h	csr_ota_service.h	No change
csr_ota_uuids.h	csr_ota_uuids.h	No change
fast_pwm.c	fast_pwm.c	No change
fast_pwm.h	fast_pwm.h	No change
gap_conn_params.h	gap_conn_params.h	No change
gap_service.c	gap_service.c	No change
gap_service.h	gap_service.h	No change
gap_service_db.db	gap_service_db.db	No change
gap_uuids.h	gap_uuids.h	No change
gatt_service.c	gatt_service.c	No change
gatt_service.h	gatt_service.h	No change
gatt_service_db.db	gatt_service_db.db	No change
gatt_service_uuids.h	gatt_service_uuids.h	No change
gunilamp_hw.c	-	Deprecated support for Gunilamp
gunilamp_hw.h	-	Deprecated support for Gunilamp
iot_hw.c	iot_hw.c	No change
iot_hw.h	iot_hw.h	No change
light_csr101x_A05.key	CSRmesh_light_csr101x_A0	Updated for 2.5.0.20 SDK

CSRmesh 1.x Application Files	CSRmesh 2.x Application Files	Purpose (Changes from 1.3 to 2.0)
r	5.keyr	
mesh_control_service.c	mesh_control_service.c	The API's for sending the incoming mesh data onto the stack have been changed.
mesh_control_service.h	mesh_control_service.h	No change
mesh_control_service_db.db	mesh_control_service_db.db	No change
mesh_control_service_uuids.h	mesh_control_service_uuids.h	No change
nvm_access.c	nvm_access.c	Support for checking and sending low battery status during NVM access has been added.
nvm_access.h	nvm_access.h	No change
otau_bootloader.keyr	otau_bootloader.keyr	No change
ota_customisation.h	ota_customisation.h	No change
pio_ctrlr_code.asm	pio_ctrlr_code.asm	No change
user_config.h	user_config.h	No change

Table 2.1: Light Example Application File Structure Comparison

3. Application Changes

3.1. Initialisation

The application initialisation remains same except for some of the API changes required for CSRmesh Stack initialisation.

The CSRmesh stack is now more modular and modified to be platform independent. This has resulted in separation of stack and scheduler API.

The following sequence of initialisation is recommended:

1. Initialise the peripherals (PIO, PWM, I2C etc.) and application state.
2. Configure CSRmesh Scheduler `CSRSchedSetConfigParams()`
3. Start Scheduler
4. Read the NVM `ReadPersistentStore()`
5. Initialise CSRmesh and register a callback function for handling CSRmesh stack events
6. Upon successful stack initialisation, do the rest of the initialisations based on the application state

Table 3.1 and Table 3.2 show the code snippet of initialisation sequence in the CSRmesh 1.3 and 2.0 Light applications respectively

Note:

Ensure that the NVM parameters are read and scheduler is started before the `CSRmeshInit()` is called.

Note:

`AppProcessCsrMeshEvent`() application function is not called by the stack anymore, but the application needs to register a call-back function to handle the CSRmesh events.

```
/* Initialise the CSRmesh */
CsrMeshInit(&g_node_data);

/* Enable Notifications for raw messages */
CsrMeshEnableRawMsgEvent(TRUE);

/* Initialise supported models */
LightModelInit(light_model_groups, MAX_MODEL_GROUPS);
PowerModelInit(power_model_groups, MAX_MODEL_GROUPS);
BearerModelInit();
#ifdef ENABLE_FIRMWARE_MODEL
FirmwareModelInit();
g_lightapp_data.fw_version.major_version = APP_MAJOR_VERSION;
g_lightapp_data.fw_version.minor_version = APP_MINOR_VERSION;
#endif /* ENABLE_FIRMWARE_MODEL */
AttentionModelInit(attention_model_groups, MAX_MODEL_GROUPS);
#ifdef ENABLE_BATTERY_MODEL
BatteryModelInit();
#endif /* ENABLE_BATTERY_MODEL */
#ifdef ENABLE_DATA_MODEL
AppDataStreamInit(data_model_groups, MAX_MODEL_GROUPS);
#endif /* ENABLE_DATA_MODEL */

/* Start CSRmesh */
CsrMeshStart();
```

Table 3.1: Initialisation Sequence of CSRmesh 1.3 Light Application

```
…
    /* Initialise CSRmesh light application State */
    AppSetState(app_state_init);

    /* Set LE Config Params */
    SetLeConfigParams(&g_lightapp_data.le_params);
    CSRSchedSetConfigParams(&g_lightapp_data.le_params);

    /* Initialize Light Hardware */
    LightHardwareInit();
    /* Start ADV GATT Scheduler */
    CSRSchedStart();

#ifdef NVM_TYPE_EEPROM
    /* Configure the NVM manager to use I2C EEPROM for NVM store */
    NvmConfigureI2cEeprom();
#elif NVM_TYPE_FLASH
    /* Configure the NVM Manager to use SPI flash for NVM store. */
    NvmConfigureSpiFlash();
#endif /* NVM_TYPE_EEPROM */
    NvmDisable();

    /* Read persistent storage */
    ReadPersistentStore();

    /* Register with CSR Mesh */
    result = CSRmeshInit(CSR_MESH_NON_CONFIG_DEVICE);

    CSRmeshRegisterAppCallback(CSRmeshAppProcessMeshEvent);

    if(result == CSR_MESH_RESULT_SUCCESS)
    {
        /* Initialise supported models */
        LightModelInit(0, light_model_groups, MAX_MODEL_GROUPS,
                                                    AppLightEventHandler);
        PowerModelInit(0, power_model_groups, MAX_MODEL_GROUPS,
                                                    AppPowerEventHandler);
        AttentionModelInit(0, attention_model_groups, MAX_MODEL_GROUPS,
                                                AppAttentionEventHandler);
        BatteryModelInit(0, NULL, MAX_MODEL_GROUPS, AppBatteryEventHandler);
#ifdef ENABLE_DATA_MODEL
        AppDataStreamInit(data_model_groups, MAX_MODEL_GROUPS);
#endif /* ENABLE_DATA_MODEL */

        /* Start CSRmesh */
        result = CSRmeshStart();

        if(result == CSR_MESH_RESULT_SUCCESS)
        {
            /* Further initialisations */
        }
```

Table 3.2: Initialisation sequence of CSRmesh 2.0 Light application

3.2. Connectable Advertisements

In the CSRmesh 2.0 applications have to schedule the advertisements at the required intervals and the CSRmesh stack would send one advertising event at the earliest possible time every time it is called by the application. If the user is interested then a call-back can be registered which is called when the advertisement has been sent.

Table 3.3 shows the code snippet for to send a connectable advertisement in the CSRmesh 2.0 Application.

```
/* API call to send the connectable advertisements */
CSRSchedSendUserAdv(le_adv_data, callBack);
```

Table 3.3: Connectable Advertisements

3.3. Notify Scheduler on GATT Connection

The Application should inform the CSRmesh stack on GATT connection completion. This is done by calling the below function in 1.3 and 2.0 releases.

Table 3.4 and Table 3.5 show the code snippets to notify GATT connection completion even to the CSRmesh stack CSRmesh 1.3 and CSRmesh 2.0.

```
/* Inform CSRmesh that we are connected now */
CsrMeshHandleDataInConnection(g_lightapp_data.gatt_data.st_ucid,
                              g_lightapp_data.gatt_data.conn_interval);
```

Table 3.4: Notify GATT Connection in CSRmesh 1.3 Application

```
/* Inform CSRmesh that we are connected now */
CSRSchedNotifyGattEvent(CSR_SCHED_GATT_STATE_CHANGE_EVENT,
                        &g_lightapp_data.gatt_event_data, NULL);
```

Table 3.5: Notify GATT Connection in CSRmesh 2.0 Application

3.4. Processing Received CSRmesh Packets

The Application receives the CSRmesh packets on non-connectable advertisements and GATT. These packets has to be passed to the CSRmesh stack for processing. The below function needs to be called to send the mesh data onto CSRmesh stack.

```
CSRSchedHandleIncomingData(CSR_SCHED_INCOMING_DATA_EVENT_T data_event,
CSR_SCHED_DATA_T *sched_data);
```

The `CSR_SCHED_INCOMING_LE_MESH_DATA_EVENT` should be called for sending the incoming mesh data through the advertisements and `CSR_SCHED_INCOMING_GATT_MESH_DATA_EVENT` should be called for sending the incoming mesh data through the GATT connection.

Table 3.6 and Table 3.7 show the code snippets for sending the received mesh packets to the mesh stack for processing in CSRmesh 1.3 and 2.0 applications respectively.

```
/* Send the raw mesh data received through a GATT connection onto CSRmesh
 * stack for processing.
 */
CsrMeshProcessRawMessage(g_mesh_svc_data.mesh_data.mesh_data,
g_mesh_svc_data.mesh_data.length);

/* Send the advertising reports to the CSRmesh to check and process if it has
 * a CSRmesh payload.
 */
CsrMeshProcessMessage((LM_EV_ADVERTISING_REPORT_T *)p_event_data);
```

Table 3.6: Passing Received Mesh Packets to Stack in CSRmesh 1.3

```
/* Send the raw mesh data received through a GATT connection onto CSRmesh
 * stack for processing
 */
    CSRSchedHandleIncomingData(CSR_SCHED_INCOMING_GATT_MESH_DATA_EVENT,
                               g_mesh_svc_data.mesh_data.mesh_data,
                               g_mesh_svc_data.mesh_data.length,
                               0x00);
/* Send the raw mesh data received through the advertisement onto CSRmesh
 * stack for processing.
 */
    CSRSchedHandleIncomingData(CSR_SCHED_INCOMING_LE_MESH_DATA_EVENT,
                               &unpackedData[index+4],
                               (length-3),
                               report->rssi);
```

Table 3.7: Passing Received Mesh Packets to the Stack in CSRmesh 2.0

Note:

Since the CSRmesh 2.0 is expected to run on multiple platforms the `CSRSchedHandleIncomingData` API expects the application to identify and extract the CSRmesh payload from the advertising report before it is passed to the stack

Extracting the CSRmesh payload from the advertising report is implemented in the `HandleLEAdvMessage()` in the `app_fw_event_handler.c` file.

3.5. Configure Scheduler to Notify Mesh Packets

In the CSRmesh 2.0 applications have to configure the scheduler to notify the mesh packets over the GATT connection onto the bridge. The scheduler should be configured to enable and disable sending of the mesh packets over the GATT connection whenever the client character configuration on either the MTL_CP or MTL_CP_2 character is written.

Table 3.8 shows the code snippet for to configure scheduler to notify mesh packets over the GATT connection in mesh_control_service.c file in CSRmesh 2.0 Application.

```
case HANDLE_MTL_CP_CLIENT_CONFIG:
case HANDLE_MTL_CP2_CLIENT_CONFIG:
{
    CSR_SCHED_GATT_EVENT_DATA_T gatt_event_data;
    gatt_event_data.cid = p_ind->cid;
    pValue = p_ind->value;
    g_mesh_svc_data.mtl_cp_ccd = BufReadUint16(&pValue);
    /* Reset the reserved bits in any case */
    g_mesh_svc_data.mtl_cp_ccd &= ~gatt_client_config_reserved;

    /* Configure the scheduler with the ccd value written */
    if((g_mesh_svc_data.mtl_cp_ccd & gatt_client_config_notification)
            == gatt_client_config_notification)
    {
        gatt_event_data.is_notification_enabled = TRUE;
        CSRSchedNotifyGattEvent(CSR_SCHED_GATT_CCCD_STATE_CHANGE_EVENT,
                                &gatt_event_data, MeshControlNotifyResponse);
    }
    else
    {
        gatt_event_data.is_notification_enabled = FALSE;
        CSRSchedNotifyGattEvent(CSR_SCHED_GATT_CCCD_STATE_CHANGE_EVENT,
                                &gatt_event_data, NULL);
    }
}
break;
```

Table 3.8: Configure Scheduler to Notify Mesh Packets

3.6. Updating Bearer States

In the CSRmesh 1.3 Applications, the relay and the promiscuous states are configured using the `CsrMeshRelayEnable`() and `CsrMeshEnablePromiscuousMode`() API's.

In CSRmesh 2.0 applications the enabling and disabling of the relay and the promiscuous `CSRmeshSetTransmitState`() function.

Table 3.9 and Table 3.10 show the code snippets of changing the promiscuous setting on association completion in CSRmesh 1.3 and 2.0 applications respectively.

```
/* The association is complete set LE bearer to non-promiscuous.*/
g_lightapp_data.bearer_data.bearerPromiscuous &= ~BLE_BEARER_MASK;
CsrMeshEnablePromiscuousMode(g_lightapp_data.bearer_data.bearerPromiscuous);
```

Table 3.9: Promiscuous Setting in CSRmesh 1.3 Application

Table 3.10 shows the code snippet of changing the promiscuous setting on association completion in CSRmesh 2.0 Application.

```
extern void AppUpdateBearerState(CSR_MESH_TRANSMIT_STATE_T *p_bearer_state)
{
    CSR_MESH_APP_EVENT_DATA_T ret_evt_data;
    CSR_MESH_TRANSMIT_STATE_T ret_bearer_state;

    ret_evt_data.appCallbackDataPtr = &ret_bearer_state;
    CSRmeshSetTransmitState(p_bearer_state, &ret_evt_data);
}

/* Disable promiscuous mode */
g_tsapp_data.bearer_tx_state.promiscuousState = FALSE;
g_tsapp_data.bearer_tx_state.bearerPromiscuous = 0;
AppUpdateBearerState(&g_tsapp_data.bearer_tx_state);
```

Table 3.10: Promiscuous Setting on Association Complete in CSRmesh 2.0 Application

3.7. NVM Changes

The CSRmesh 2.0 stack reserves the first few words of the NVM. The number of words used by the CSRmesh stack is given by `CSR_MESH_NVM_SIZE` defined in `csr_mesh.h`

The application must start to store its own NVM parameters after this offset. In CSRmesh 2.0 release, the applications need not handle the NVM used in the library.

3.8. Model Handlers

The model API is made uniform across the models. Each model defines a structure type for every supported message. There is an option of registering a different call-back function for each model during initialisation. Each model is released as a Client and a Server module. The application needs to include the corresponding model (**client/server**) library which needs to be supported.

Table 3.11 shows the code example for handling the battery model messages present in the 2.0 CSRmeshLight application.

```
CSRmeshResult AppBatteryEventHandler(CSRMESH_MODEL_EVENT_T event_code,
                                     CSRMESH_EVENT_DATA_T* data,
                                     CsrUint16 length,
                                     void **state_data)
{
    switch(event_code)
    {
        case CSRMESH_BATTERY_GET_STATE:
        {
            CSRMESH_BATTERY_GET_STATE_T *p_get_state =
                                      (CSRMESH_BATTERY_GET_STATE_T *)data->data;

            /* Read Battery Level */
            g_lightapp_data.battery_model.batterylevel = ReadBatteryLevel();
            g_lightapp_data.battery_model.batterystate = GetBatteryState();
            g_lightapp_data.battery_model.tid = p_get_state->tid;

            /* Pass Battery state data to model */
            if (state_data != NULL)
            {
                *state_data = (void *)&g_lightapp_data.battery_model;
            }
        }
        break;

        default:
        break;
    }

    return CSR_MESH_RESULT_SUCCESS;
}
```

Table 3.11: Battery Model Message Handling

Document References

Document	Reference
CSRmesh API Definition	https://www.csrsupport.com/CSRmesh

Terms and Definitions

API	Application Programming Interface
Bluetooth®	Set of technologies providing audio and data transfer over short-range radio connections
CSR	Cambridge Silicon Radio
I^2C	Inter-Integrated Circuit
NVM	Non Volatile Memory
PIO	Programmable Input Output
PWM	Pulse Width Modulation
SDK	Software Development Kit