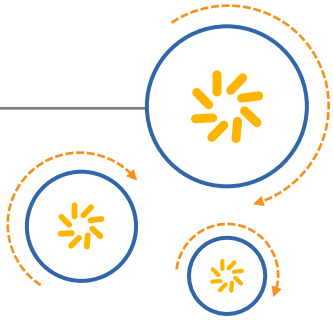
Qualcomm Technologies International, Ltd.



# CSRmesh v2.1 Application

## Porting Guide

80-CG012-1 Rev. AB

April 10, 2018

# Revision history

| Revision | Date | Description |
|---|---|---|
| AA | February 2018 | Initial release, alternate document number CS-00348721-DC |
| AB | April 2018 | Minor corrections. |

# Contents



## Tables

# 1 Introduction

This document describes the CSRmesh v2.1 on-chip application's new design with the integration of the Connection Manager built using the µEnergy SDKs. It mainly captures the process for porting the CSRmesh v2.0 on-chip applications to the new design.

The CSRmesh applications prior to the CSRmesh v2.1 release did not contain the common code for handling connection, core mesh stack, and model handlers resulting in duplication of a lot of code between applications nor had a structure which can have the capability to handle multiple connections to devices. The current changes move the CSRmesh connection, core mesh stack and model handling code to be reused across various applications and would make the integration much easier than the earlier versions. The integration of the Connection Manager paves the way for connection with multiple devices in future implementations.

The document also describes the modifications in the application structure as well as required changes on various modules.

# 2 New application design

## 2.1 Initialization

The application initialization has been split into the hardware initialization, mesh model initialization, services initialization and the mesh core stack initialization.

The application starts the initialization sequence from the `AppInit()` function as mentioned below.

1. Initialization for the Timers required for the application.
2. Initialization of debug UART if debug is enabled.
3. Initialize the hardware peripherals (PIO, PWM, I2C etc.)
4. Read the user keys.
5. Initialize the NVM.
6. Initialize the Connection Manager.

The code snippet in Table 2-1shows `Appinit()` initialization.

**Table 2-1 AppInit() initialization sequence of CSRmesh v2.1 Light application**

```
void AppInit(sleep_state last_sleep_state)
{
    /* Initialise the application timers */
    TimerInit(MAX_APP_TIMERS, (void*)app_timers);

#ifdef DEBUG_ENABLE
#ifdef CSR101x_A05
    /* Initialise UART and configure with
     * default baud rate and port configuration.
     */
    DebugInit(UART_BUF_SIZE_BYTES_256, UartDataRxCallback, NULL);

    /* UART Rx threshold is set to 1,
     * so that every byte received will trigger the rx callback.
     */
```

```
    /* Standard setup of CSR102x boards */
    uart.rx  = UART_PIO_RX;
    uart.tx  = UART_PIO_TX;
    uart.rts = UART_PIO_RTS;
    uart.cts = UART_PIO_CTS;

    /* Initialise Default UART communications */
    DebugInit(1, UART_RATE_921K6, 0, &uart);
#endif /* CSR101x_A05 */
#endif /* DEBUG_ENABLE */

    /* Initialise Light Hardware */
    LightHardwareInit();

    /* Read the user keys */
    readUserKeys();
}
```

After the completion of reading the user keys, the NVM is initialized. After the NVM initialization the Connection Manager initialization is done.

When the Connection Manager is initialized it, sends a `CM_INIT_CFM` message to the application which is handled in the `connection_handler.c` file.

When the application receives the confirmation that the Connection Manger is initialized, it:

1. Enables the raw advertisement reports to be sent from Connection Manager.
2. Initializes the GATT server services supported in the application.
3. Initializes the application data structure.
4. Initializes the application mesh handler.

See Table 2-2.

**Table 2-2 Connection Manager initialization sequence of CSRmesh v2.1 Light application**

```
static void handleCmInitCfm(CM_INIT_CFM_T *cm_event_data)
{
    /* Set the CM API to send raw advertising reports to the application*/
    CMObserverEnableRawReports(TRUE);

    /* Initialise the services supported by the application */
    InitAppSupportedServices();

    /* Initialise the application data structure */
    AppDataInit();

#ifdef USE_STATIC_RANDOM_ADDRESS
    GattSetRandomAddress();
#endif /* USE_STATIC_RANDOM_ADDRESS */

    /* Initialise Mesh */
    AppMeshInit();

    /* Start advertising */
    AppSetState(app_state_advertising);
}
```

The mesh initialization function initializes the mesh models and the core stack in the following sequence:

1. Configure the scheduler parameters and then initialise the mesh scheduler.
2. Initialise the model data for models supported by the application. This needs to be called before calling the `readPersistentStore()` function.
3. Read the persistent data from NVM and store onto the application and models data structure.
4. Initialise the CSRmesh stack and the core mesh handler which handles the event from the core mesh stack.
5. Initialise the supported model handlers and then start the CSRmesh.
6. 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.

The code snippet for the above initialization sequence is shown in Table 2-3.

**Table 2-3 Mesh initialization sequence of CSRmesh v2.1 Light application**

```
extern void AppMeshInit(void)
{
    CSRmeshResult result = CSR_MESH_RESULT_FAILURE;

    /* Set LE Config Params */
    setLeConfigParams(&le_params);
    CSRSchedSetConfigParams(&le_params);
    /* Start ADV GATT Scheduler */
    CSRSchedStart();

    initialiseSupportedModelData();

    /* Read persistent storage */
    readPersistentStore();

    /* Register with CSR Mesh */
#ifdef USE_AUTHORISATION_CODE
    result = CSRmeshInit(CSR_MESH_NON_CONFIG_DEVICE_WITH_AUTH_CODE);
#else
    result = CSRmeshInit(CSR_MESH_NON_CONFIG_DEVICE);
#endif /* USE_AUTHORISATION_CODE */

    g_mesh_handler_data.appearance = &appearance;
    g_mesh_handler_data.vid_pid_info = &vid_pid_info;
    MeshHandlerInit(&g_mesh_handler_data);

    if(result == CSR_MESH_RESULT_SUCCESS)
    {
        /* Initialize all the models supported in the application. */
        initializeSupportedModels();

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

        if(result == CSR_MESH_RESULT_SUCCESS)
        {
            /* Post MASP_DEVICE_ID msg in Bearer TX queue */
            if(g_mesh_handler_data.assoc_state == app_state_not_associated)
            {
                if(g_app_nvm_fresh == TRUE)
                {
                    /* As the application's sanity has changed, the
association
                     * information in the core stack needs to be removed.
                     */
                    CSRmeshRemoveNetwork(CSR_MESH_DEFAULT_NETID);
                }
                /* Start sending device UUID adverts */
                InitiateAssociation();
            }
```

```
            else
            {
                DEBUG_STR("Light is associated\r\n");
#ifdef ENABLE_TUNING_MODEL
                TuningModelStart(DEFAULT_TUNING_PROBE_PERIOD,
                                 DEFAULT_TUNING_REPORT_PERIOD);
#endif
#ifdef ENABLE_ASSET_MODEL
                AssetStartBroadcast();
#endif
                /* Initialize the source sequence cache */
                AppInitializeSeqCache();

                /* Light is already associated. Set the colour from NVM */
                RestoreLightState();
            }
            /* Update relay and promiscuous settings as per device state */
            AppUpdateBearerState(&g_mesh_handler_data.bearer_tx_state);

            /* Update the TTL value onto mesh stack */
            CSRmeshSetDefaultTTL(g_mesh_handler_data.ttl_value);
        }
        }
    else
    {
        /* Registration has failed */
    }
}
```

## 2.2 Connection Manager integration

The CSRmesh v2.1 applications have been integrated to work with the Connection Manager. The Connection Manager is a common platform for all the µEnergy 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 technology system in an efficient and simple way.

The Connection Manager by default supports a number of components as mentioned below which could be enabled based on the below mentioned flags as defined in the `csr_mesh_light_csr102x_uenergyprops.xml` file. If these flags are enabled, then the corresponding features of the Connection Manager are enabled to build by default.

**Table 2-4 Connection manager flags to enable features**

```
CM Client Support -> This flag enables the GATT client functionality in the
connection manager.

CM Server Support -> This flag enables the GATT server functionality in the
connection manager.

CM Peripheral Support -> This flag enables the LE peripheral functionality in
the connection manager.

CM Central Support -> This flag enables the LE central functionality in the
connection manager.

CM Observer Support -> This flag enables the LE observer functionality in the
connection manager.
```

In the mesh node applications, the support for the roles is enabled as per the requirement of the application. The base roles of CM Server, CM Peripheral and the CM Observer need to be supported by all applications.

There are a few definitions that need to be defined to configure the Connection Manager, these are defined in the `user_config.h` file, see Table 2-5.

**Table 2-5 Connection Manager configuration in CSRmesh applications**

```
/* Maximum active connections */
#define MAX_CONNECTIONS                         (1)

/* Maximum paired devices */
#define MAX_PAIRED_DEVICES                      (1)

/* Maximum number of server services supported */
#if defined(CSR101x_A05) && !defined(OTAU_BOOTLOADER)
#define MAX_SERVER_SERVICES                     (3)
#else
#define MAX_SERVER_SERVICES                     (4)
#endif

/* Initial diversifier */
#define DIVERSIFIER                             (0)
```

The initialization of the Connection Manager and handling of various events received from the Connection Manager are handled in the `connection_handler.c` file. The Connection Manager initialization is as shown in Table 2-6.

**Table 2-6 Connection Manager initialization**

```
extern void AppCMInit(bool nvm_fresh, uint16* nvm_offset)
{
    CM_INIT_PARAMS_T init_params;

    /* Initialise conn manager initial parameters */
    getCmInitParams(&init_params, nvm_fresh, nvm_offset);

    /* Initialize the connection manager. Wait for the event CM_INIT_CFM
before
     * accessing the CM procedures.
     */
    CMInit(&init_params);
}
```

Events from the Connection Manager are handled as shown in Table 2-7.

**Table 2-7 Connection manager configuration in CSRmesh Applications**

```
static void appHandleConnMgrProcEvent(cm_event event_type,
                                       CM_EVENT_T *cm_event_data)
{
    switch(event_type)
    {
        case CM_INIT_CFM:
            handleCmInitCfm((CM_INIT_CFM_T*)cm_event_data);
        break;

        case CM_RAW_ADV_REPORT_IND:
            handleCmRawAdvReportInd((CM_RAW_ADV_REPORT_IND_T*)cm_event_data);
        break;

       case CM_CONNECTION_NOTIFY:
            handleConnNotify((CM_CONNECTION_NOTIFY_T *)cm_event_data);
        break;

        case CM_BONDING_NOTIFY:
            handleBondNotify((CM_BONDING_NOTIFY_T *)cm_event_data);
        break;

        case CM_ENCRYPTION_NOTIFY:
           handleEncryptionChangeNotify((CM_ENCRYPTION_NOTIFY_T*)cm_event_data);
        break;
        case CM_CONNECTION_UPDATED:
            handleCmConnectionUpdated((CM_CONNECTION_UPDATED_T*)cm_event_data);
            ConnectionParamUpdateEvent(event_type, cm_event_data);
        break;
        case CM_RADIO_EVENT_IND:
             CSRSchedNotifyGattEvent(CSR_SCHED_GATT_CONNECTION_EVENT,NULL,NULL);
        break;
        case CM_CONNECTION_UPDATE_SIGNALLING_IND:   /* FALLTHROUGH */
        case CM_CONNECTION_PARAM_UPDATE_CFM:        /* FALLTHROUGH */
        case CM_CONNECTION_PARAM_UPDATE_IND:        /* FALLTHROUGH */
        case CM_SERVER_ACCESSED:                    /* FALLTHROUGH */
        {
            ConnectionParamUpdateEvent(event_type, cm_event_data);
        }
        break;
#ifndef CSR101x_A05
#ifdef GAIA_OTAU_RELAY_SUPPORT
        case CM_ADV_REPORT_IND:
            handleCmAdvReport((CM_ADV_REPORT_IND_T*)cm_event_data);
            break;

         case CM_DISCOVERY_COMPLETE:
            /* Hanlde the gatt discovery complete indication */
            handleDiscoveryComplete((CM_DISCOVERY_COMPLETE_T*)cm_event_data);
            break;
#endif
        case CM_EARLY_WAKEUP_IND:
            CSRSchedNotifyGattEvent(CSR_SCHED_GATT_EARLY_WAKEUP_EVENT, NULL,
NULL);
        break;
#endif
        default:
        break;
    }
}
```

The Connection Manager events are almost one-on-one mapping of the events received from the firmware in the previous releases of CSRmesh. One noticeable difference in the Connection Manager integration is that the connected devices are differentiated based on the device id unlike the earlier implementations where the application used to differentiate the connected devices based on the connection id.

## 2.3 Initialize Client and Server services

The CSRmesh applications mandatorily implement GAP, GATT and Mesh Control Service services in the server role. The applications also implement OTA and GAIA services in the server role for over the air update procedure on the CSR101x and the CSR102x platforms respectively. The GAIA Client Service is also initialized when the applications want to support the over the air update procedure over CSRmesh LOT relay.

Currently with the introduction of the Connection Manager all the µEnergy applications can share these server services which are stored in a common location. Hence the CSRmesh applications are now configured to use these services through the Connection Manager. To enable these server services, we need to enable the Connection Manager server role and initialize the server services, see Table 2-8.

**Table 2-8 Supported server service initialization**

```
extern void InitAppSupportedServices(void)
{
#ifndef CSR101x_A05
    g_app_nvm_offset = NVM_OFFSET_GAIA_OTA_SERVICE;
    /* Initialize the Gaia OTA service. */
#ifdef RESET_NVM
    GAIAInitServerService(g_gaia_nvm_fresh, &g_app_nvm_offset);
#else
    GAIAInitServerService(g_app_nvm_fresh, &g_app_nvm_offset);
#endif /* RESET_NVM */
#ifdef GAIA_OTAU_SUPPORT
    AppGaiaOtauInit();
#endif /* GAIA_OTAU_SUPPORT */
#ifdef GAIA_OTAU_RELAY_SUPPORT
    GaiaInitClientService(g_app_nvm_fresh, &g_app_nvm_offset);
    GaiaAppClientInit(g_app_nvm_fresh,&g_app_nvm_offset);
#endif
    g_app_nvm_offset = NVM_OFFSET_MESH_APP_SERVICES;
#else
#ifdef OTAU_BOOTLOADER
    /* Initialize the OTA service. */
    OtaInitServerService(g_app_nvm_fresh, &g_app_nvm_offset);
#endif
#endif

    /* Initialize the GATT service. */
    GattExtInitServerService(g_app_nvm_fresh, &g_app_nvm_offset, TRUE);

    /* Initialize the GAP service. */
    GapInitServerServiceNoBond();

    /* Initialize the Mesh control service */
    MeshControlInitServerService(g_app_nvm_fresh, &g_app_nvm_offset);
}
```

To specify the number of services that are included in the application to the Connection Manager, we need to set the number of services supported in the application with the `MAX_SERVER_SERVICES` macro defined in the `user_config.h` file.

# 2.4 Application handlers

## 2.4.1 Advertisement handler

The advertisement handler is responsible for sending the connectable advertisements through mesh core stack to connect as a bridge onto an android or an iOS device. The connectable advertisements once triggered would be sent continually at the advert interval set until a connection is established or a function is called to stop sending the connectable advertisements. These advertisements are sent through the CSRmesh core stack and are scheduled to be sent with the mesh non-connectable advertisements. The below advertisement handler APIs could be called from the application to start and stop the connectable advertisements.

**Table 2-9 Functions to start and stop sending connectable advertisements**

```
/* Function to trigger sending connectable advertisements */
extern void GattTriggerConnectableAdverts(void);

/* Function to stop sending the ongoing connectable advertisements */
extern void GattStopAdverts(void);

/* Connectable Advertisement interval macro */
#define GATT_ADVERT_GROSS_INTERVAL              (1245 * MILLISECOND)
#define GATT_ADVERT_RANDOM                      (10 * MILLISECOND)

#define ADVERT_INTERVAL                         (GATT_ADVERT_GROSS_INTERVAL \
                                                - (GATT_ADVERT_RANDOM/2))
```

## 2.4.2 Connection handlerk

The connection handler is responsible for initialization of the Connection Manager as well as handling the Connection Manager events. The connection handler also handles the application events on connection and disconnection and also informs the scheduler of the connection status with the connection parameter information. The connection handler is also responsible for updating the required connection parameters required for the application when a new connection is established. The application should define the required connection parameters in the `app_conn_param.h` file, see Table 2-10.

**Table 2-10 Connection parameters defined in app_conn_params.h file**

```
/* Minimum and maximum connection interval in number of frames */
#define PREFERRED_MAX_CON_INTERVAL          96 /* 120 ms */
#define PREFERRED_MIN_CON_INTERVAL          72 /*  90 ms */

/* Slave latency in number of connection intervals */
#define PREFERRED_SLAVE_LATENCY             0x0000 /* 0 conn_intervals */

/* Supervision time-out (ms) = PREFERRED_SUPERVISION_TIMEOUT * 10 ms */
#define PREFERRED SUPERVISION TIMEOUT       0x0258 /* 6 seconds */
```

## 2.4.3 Core mesh handler

The core mesh handler is responsible for interaction with the core mesh stack during the association processes as well as handling the core mesh events from the stack. The handler is also responsible for configuring and updating the bearer setting in the core CSRmesh stack.

The core mesh handler provides APIs to initiate the association process and update the bearer settings as well as provides APIs to get the association state of the application as well the current bearer settings of the application. The handler on invoking the `InitiateAssociation()` function starts sending the mesh UUID advertisements periodically for the controller devices to associate with the node application.

The core mesh handler has to be initialized from the application by calling the `MeshHandlerInit()` function. The APIs provided by the core mesh handler are shown in Table 2-11.

**Table 2-11 Core mesh handler APIs**

```
/* The function initialises the core mesh handler */
extern void MeshHandlerInit(MESH_HANDLER_DATA_T* mesh_handler_data);

/* This function kick starts a timer to send periodic CSRmesh UUID Messages
 * and starts blinking Blue LED to visually indicate association ready
status
 */
extern void InitiateAssociation(void);

/* This function updates the relay and promiscuous mode of the GATT and
 * and the LE Advert bearers
 */
extern void AppUpdateBearerState(CSR_MESH_TRANSMIT_STATE_T *p_bearer_state);

/* This function returns the association state. */
extern app_association_state AppGetAssociatedState(void);

/* This function returns the stored bearer state. */
extern CSR_MESH_TRANSMIT_STATE_T AppGetStoredBearerState(void);

/* This function returns the status of the GATT bearer */
extern bool IsGattBearerEnabled(void);

/* This function returns the TTL value stored in the application */
extern uint8 AppGetCurrentTTL(void);

/* This function sets the Tx power onto the firmware based on the power
dbm */
extern void AppSetTxPower(CsrInt8 power);

/* This function initializes the sequence cache with the mesh stack */
extern void AppInitializeSeqCache(CsrBool new_association);

/* This function clears the sequence cache with the mesh stack */
extern void AppClearSeqCache(void);
```

## 2.4.4 Mesh model handlers

To make the applications more modular and to make the effort of inclusion of models much simpler the handling of models is done in the model handlers. The model handlers would initialize the models of the CSRmesh library as well as handle the events received from the models. The model handlers also share the data structure with the main application so that the data can be accessed by both.

This section describes the application modifications required to include a model into the CSRmesh application code.

First a new model supported flag needs to be defined in the user_config.h for inclusion of the model as shown in Table 2-12.

**Table 2-12 Light model flag support in user_config.h file**

```
/* Enable Light model support */
#define ENABLE_LIGHT_MODEL
```

Table 2-13 shows the data structure initialization and the handler initialization.

**Table 2-13 Light model handler initialisation**

```
/* Initialize Light Model Data */
#ifdef ENABLE_LIGHT_MODEL
        LightModelDataInit(&g_light_handler_data);
#endif /* ENABLE_LIGHT_MODEL */


/* Initialize Light Model */
#ifdef ENABLE_LIGHT_MODEL
        LightModelHandlerInit(CSR_MESH_DEFAULT_NETID, light_model_groups,
                              MAX_MODEL_GROUPS);
#endif /* ENABLE_LIGHT_MODEL */
```

There are certain instances where the events from the mesh models would need to be indicated to the application. This can be done by calling a predefined function as done in some models or by passing the mesh model events received along with the relevant data received from the model events. An example of calling a common function and passing the model events is shown in.

**Table 2-14 Sending events from the Sensor model handler to the application**

```
/* Data structure defined to send the information from the Sensor model
handler
 * to the application */
typedef struct
{
    CSRMESH_MODEL_EVENT_T          event_code;
    sensor_type_t                  type;
    sensor_type_t                  type2;
    SENSOR_FORMAT_TEMPERATURE_T    recvd_curr_temp;
    SENSOR_FORMAT_TEMPERATURE_T    recvd_desired_temp;
    CsrUint8                       repeatinterval;
    CsrUint16                      src_id;
    CsrUint8                       tid;
}SENSOR_APP_DATA_T;

/* This function is called from the sensor model handler to the app to
send the event and the received data from the handler */

extern void AppSensorModelHandler(SENSOR_APP_DATA_T sensor_app_data);
```

# 2.5 NVM changes

The NVM handlers defined in each of the applications have been moved onto a common location of `nvm_manager` which would handle complete NVM functionality. The NVM functionality has been updated to support both the CSR101x and CSR102x modules with the same APIs.

On the application level the NVM remains the same for CSRmesh core stack and the application mesh data. The only NVM changes are with the integration of the Connection Manager which stores the server service information onto NVM.

# 3 File organization changes

This section explains the file organizational changes in the application between the CSRmesh v2.0 and CSRmeshv2.0.1 applications to the CSRmesh v2.1 applications.

Table 3-1 provides a file to file comparison of changes from the previous release as well as the new path of the files in the currently modified directory structure.

**Table 3-1 Light example application file structure comparison**

| CSRmesh v2.0 and CSRmesh v2.0.1 Application Files | CSRmesh v2.1 Application Files | Purpose (Code Structural changes and Connection Manager Integration) |
|---|---|---|
| appearance.h | ..\mesh_common\mesh\utils\appearance.h | Contains the appearance value macro for all the CSRmesh applications as it has been made common. Earlier each app had its own appearance.h file. |
| app_data_stream.c | ..\mesh_common\mesh\handlers\data_model\data_model_handler.c | This file made as a common handler and is used for handling the data stream across all applications. . |
| app_data_stream.h | ..\mesh_common\mesh\handlers\data_model\data_model_handler.h | The header file for handling the data model API's. |
| app_debug.h | ..\mesh_common\mesh\utils\app_debug.h | The file has been moved onto a common location so that it can be reused by all the applications. The file contains the debug definitions for printing the debug information. |
| app_gatt.h | ..\mesh_common\mesh\utils\app_gatt.h | The file has been moved onto a common location so that it can be reused by all the applications. |
| app_gatt_db.db | app_gatt_db.db | The OTA database has been removed as we do not support the OTA in this release. |
| app_fw_event_handler.c | ..\mesh_common\mesh\handlers\connection\connection_handler.c | With the integration of the CM, the application would handle the events from the CM which in turn handle the firmware events. Hence the firmware events handler has been replaced by a common connection handler which handles the CM events across all applications. |
| app_fw_event_handler.h | ..\mesh_common\mesh\handlers\connection\connection_handler.h | The file specifies the public functions definitions of the connection handler. |
| app_mesh_event_handler.c | ..\mesh_common\mesh\handlers\core_mesh\core_mesh_handler.c | This common file handles the events from the core mesh stack and is used by all the applications. The file also contains the procedure to start association as well as set and get the bearer related information. |

| CSRmesh v2.0 and CSRmesh v2.0.1 Application Files | CSRmesh v2.1 Application Files | Purpose (Code Structural changes and Connection Manager Integration) |
|---|---|---|
| | app_mesh_handler.c | This file contains the initialization and handling of the core mesh and various supported models in the application. |
| app_mesh_event_handler.c | ..\mesh_common\mesh\handlers\core_mesh\core_mesh_handler.h | The file specifies the public functions definitions of the core mesh handler. |
| battery_hw.c | ..\mesh_common\mesh\drivers\ battery_hw.c | The file has been moved onto a common location so that it can be reused by all the applications. |
| battery_hw.h | ..\mesh_common\mesh\drivers\ battery_hw.h | The file has been moved onto a common location so that it can be reused by all the applications. |
| CSRMeshLight.xip | csr_mesh_light_csr102x.xip | The file is updated with the new file structure and the changes required for implementing the application on the CSR102x platform. The file also contains the modifications to support the Connection Manager. This file is present only if the release supports CSR102x platform. |
| | csr_mesh_light_csr101x.xip | The file is updated with the new file structure and the changes required for implementing the application on the CSR101x platform. The file also contains the modifications to support the Connection Manager. This file is present only if the release supports CSR101x platform. |
| CSRMeshLight.xiw | csr_mesh_light_csr102x.xiw | Project file for 102x platform. This file is present only if the release supports CSR102x platform. |
| | csr_mesh_light_csr101x.xiw | Project file for 101x platform. This file is present only if the release supports CSR101x platform. |
| csr_mesh_light.c | main_app.c | The file has been modified just to process the events received from the firmware. |
| csr_mesh_light.h | main_app.h | The file contains the initializations of the public data used in the application as well as defines the application NVM offset initialization. |
| csr_mesh_light_gatt.c | ..\mesh_common\mesh\handlers\advertisement\advertisement_handler.c | This file contains the implementation of sending the connectable advertisement through the mesh. |
| csr_mesh_light_gatt.h | ..\mesh_common\mesh\handlers\advertisement\advertisement_handler.h | The file specifies the public functions definitions of the advertisement handler. |
| csr_mesh_light_hw.c | app_hw.c | The file contains the light hardware implementations on the CSR101x and CSR102x platforms. |
| csr_mesh_light_hw.h | app_hw.h | No change |
| csr_mesh_light_util.c | app_mesh_handler.c | The group and the nvm handling has been moved to mesh handler. |
| csr_mesh_light_util.h | app_mesh_handler.h | No change |
| csr_ota_db.db | ..\mesh_common\server\csr_ota\ csr_ota_db.db | The CSR OTA Service is supported only on CSR101x platform. |

| CSRmesh v2.0 and CSRmesh v2.0.1 Application Files | CSRmesh v2.1 Application Files | Purpose (Code Structural changes and Connection Manager Integration) |
|---|---|---|
| csr_ota_service.c | ..\mesh_common\server\csr_ota\ csr_ota_service.c | The CSR OTA Service is supported only on CSR101x platform. |
| csr_ota_service.h | ..\mesh_common\server\csr_ota\ csr_ota_service.h | The CSR OTA Service is supported only on CSR101x platform. |
| csr_ota_uuids.h | ..\mesh_common\server\csr_ota\ csr_ota_uuids.h | The CSR OTA Service is supported only on CSR101x platform. |
| CSRmesh_light_csr101x_A05.keyr | csr_mesh_light_csr101x_A05.keyr | NVM start address has been modified as we do not need the bootloader in our build as we do not support OTA through bootloader anymore. |
| fast_pwm.c | ..\mesh_common\mesh\drivers\ fast_pwm.c | No change. |
| fast_pwm.h | ..\mesh_common\mesh\drivers\ fast_pwm.h | No change |
| gap_conn_params.h | app_conn_params.h | No change |
| gap_service.c | ..\mesh_common\server\gap\gap_service .c | The server service file is made common and has been modified to work with the Connection Manager. |
| gap_service.h | ..\mesh_common\server\gap\gap_service .h | The service file header defining the public functions for the service. |
| gap_service_db.db | ..\mesh_common\server\gap\gap_service _db.db | No change |
| gap_uuids.h | ..\mesh_common\server\gap\gap_uuids.h | No change |
| gatt_service.c | ..\mesh_common\server\gatt\gatt_servi ce.c | The server service file is made common and has been modified to work with the Connection Manager. |
| gatt_service.h | ..\mesh_common\server\gatt\gatt_servi ce.h | The service file header defining the public functions for the service. |
| gatt_service_db.db | ..\mesh_common\server\gatt\gatt_servi ce_db.db | No change |
| gatt_service_uuids.h | ..\mesh_common\server\gatt\gatt_servi ce_uuids.h | No change |
| iot_hw.c | ..\mesh_common\mesh\drivers\ iot_hw.c | This file is placed into the common folder so that it can be reused by all applications as well as the file is modified to support the csr102x based IOT board. |
| iot_hw.h | ..\mesh_common\mesh\drivers\ iot_hw.h | New initializations have been added for the csr102x based IOT boards. |
| mesh_control_service.c | ..\mesh_common\server\mesh_control\me sh_control_service.c | The server service file is made common and has been modified to work with the Connection Manager. |
| mesh_control_service.h | ..\mesh_common\server\mesh_control\me sh_control_service.h | The service file header defining the public functions for the service. |
| mesh_control_service_db.db | ..\mesh_common\server\mesh_control\me sh_control_service_db.db | No change |
| mesh_control_service_uids.h | ..\mesh_common\server\mesh_control\me sh_control_service_uuids.h | No change |
| nvm_access.c | ..\mesh_common\components\nvm_manager \nvm_access.c | The file has been modified to include the support for csr102x as well as made common to use across different applications. |
| otau_bootloader.keyr | – | As we do not support OTA through bootloader this file has been removed |

| CSRmesh v2.0 and CSRmesh v2.0.1 Application Files | CSRmesh v2.1 Application Files | Purpose (Code Structural changes and Connection Manager Integration) |
|---|---|---|
| ota_customisation.h | – | As we do not support OTA through bootloader this file has been removed |
| pio_ctrlr_code.asm | `pio_ctrlr_code.asm` | No change |
| user_config.h | `user_config.h` | Has been modified to include Connection Manager based defines as well as added flags to enable and disable each individual model. |
| - | `csr_mesh_light_csr102x.htfp` | Configuration file added for the csr102x platform. This file is present only if the release supports CSR102x platform. |
| - | `csr_mesh_light_csr101x_uenergyprops.xml` | The file defines the Connection Manager components. |
| - | `csr_mesh_light_csr102x_uenergyprops.xml` | The file defines the Connection Manager components. |
| ..\include | `..\mesh_common\mesh\include\` | - |
| ..\libraries | `..\mesh_common\mesh\libraries\csr_101x\` | Libraries for the csr101x platform. |
| | `..\mesh_common\mesh\libraries\csr_102x\` | Libraries for the csr102x platform. |
| - | `..\mesh_common\components\Connection Manager\` | This folder contains the complete Connection Manager implementation. |
| - | `..\mesh_common\interfaces\i2c\` | This folder contains the i2c implementation for csr102x platform. |
| - | `..\mesh_common\peripheral\` | This folder contains the implementation of the connection parameter update procedure. |

# Document references

| Document | Reference |
|---|---|
| *CSRmesh Node API Specification* | 80-CG013-1 / CS-00348851-SP |
| *CSRmesh 2.1 Home Automation Application Note* | 80-CG011-1 / CS-00348212-AN |
| *CSR102x Application Development Using the Connection Manager Application Note* | 80-CU027-1 / CS-00343722-AN |

# Terms and definitions

| | |
|---|---|
| API | Application Programming Interface |
| Bluetooth | Set of technologies providing audio and data transfer over short-range radio connections |
| CM | Connection Manager |
| CSR | Cambridge Silicon Radio |
| NVM | Non Volatile Memory |
| OTA | Over the Air |
| OTAU | Over the Air Update |
| PIO | Programmable Input Output |
| SDK | Software Development Kit |