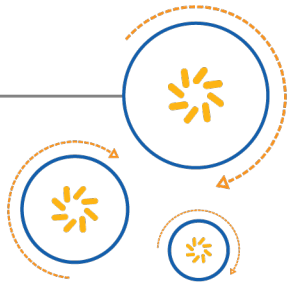
Qualcomm Technologies International, Ltd.



CSRmesh Node

API Reference

80-CG013-1 Rev. AA

February 20, 2018

Document History

Revision	Date	Change Reason
1	OCT 16	Initial release
AA	February 2018	Alternate document number CS-00348851-SP

Contents

1 CSRmesh™ Documentation

Introduction

Qualcomm Technologies International, Ltd. (QTIL) CSRmesh™ connectivity provides a Bluetooth Smart based system that allows multiple Bluetooth Smart devices to associate to a Mesh network and communicate with other devices that are also associated on the network.

Models

CSRmesh™ implements several models which a device can support. Models define the state and behaviour of a device and hence can be used to control and configure Mesh devices.

2 Related Pages

Here is a list of all related documentation pages:

- **Handling 8 bit data types on XAP**

3 Handling 8 bit data types on XAP

On the XAP, a "byte" is 16 bits, not 8 bits. It is not the same as an "octet". This extends to the C compiler: all of the types char, short, and int are 16 bits, and sizeof (int) equals 1. The same holds good for the 8 bit defined types of uint8 and int8.

Any operations performed on the 8 bit variables are executed by the processor as a 16-bit operation. So the result of 255+1 on a uint8 variable will be stored in the memory as 256 instead of 0.

Hence the application developer must take proper care while passing the 8-bit data types as arguments to the CSRmesh API.

In the case of a uint8 variable the upper byte must always be 0. In the case of a int8 variable the upper byte must be either 0 or 0xFF.

While using a signed 8-bit variable which has a negative value must be explicitly sign extended. Example: if int8 a = (int8)0xFF the value will be 255 instead of -1. To correctly store -1 in an int8 type store it as either int8 a = (int8)-1, or int8 a = (int8)0xFFFF.

4 Modules

Here is a list of all modules:

Name	Description
CSRmesh™	
Core Stack	Core Stack API
Models	
Watchdog Model	Watchdog Model API
Client	
Server	
Config Model	Config Model API
Client	
Server	
Group Model	Group Model API
Client	
Server	
Sensor Model	Sensor Model API
Client	
Server	
Actuator Model	Actuator Model API
Client	
Server	
Data Model	Data Model API
Client	
Server	
Bearer Model	Bearer Model API
Client	
Server	
Ping Model	Ping Model API
Client	
Server	
Battery Model	Battery Model API

Name	Description
Client	
Server	
Attention Model	Attention Model API
Client	
Server	
Power Model	Power Model API
Client	
Server	
Light Model	Light Model API
Client	
Server	
Asset Model	Asset Model API
Client	
Server	
Tracker Model	Tracker Model API
Client	
Server	
Time Model	Time Model API
Client	
Server	
Switch Model	Switch Model API
Client	
Server	
Tuning Model	Tuning Model API
Client	
Server	
Extension Model	Extension Model API
Client	
Server	
LargeObjectTransfer Model	LargeObjectTransfer Model API
Client	
Server	
Firmware Model	Firmware Model API
Client	
Server	
Diagnostic Model	Diagnostic Model API
Client	

Name	Description
Server	
Action Model	Action Model API
Client	
Server	
Beacon Model	Beacon Model API
Client	
Server	
BeaconProxy Model	BeaconProxy Model API
Client	
Server	
NVM_Access	

5 CSRmesh™

Detailed Description

5.1 Modules of CSRmesh™

Core Stack

Core Stack API.

Models

6 Core Stack

Detailed Description

Core Stack API.

6.1 Data Structures of Core Stack

`struct CSR_MESH_UUID_T`

128-bit UUID type

`struct CSR_MESH_VID_PID_VERSION_T`

CSRmesh Product ID, Vendor ID and Version Information.

`struct CSR_MESH_CONFORMANCE_SIGNATURE_T`

CSRmesh conformance signature Information.

`struct CSR_MESH_STACK_VERSION_T`

CSRmesh stack version Information.

`struct CSR_MESH_CC_INFO_T`

Client config info.

`struct CSR_MESH_AUTH_CODE_T`

64 bit Authorisation Code type

`struct CSR_MESH_CONFIG_BEARER_PARAM_T`

CSRmesh Scan and Advertising Parameters.

`struct CSR_MESH_CONFIG_MSG_PARAMS_T`

CSRmesh Scan and Advertising Parameters.

`struct CSR_MESH_NETID_LIST_T`

CSR Mesh Network Id List.

`struct CSR_MESH_NW_RELAY_T`

CSR Mesh transmit state.

`struct CSR_MESH_TRANSMIT_STATE_T`

CSR Mesh transmit state.

`struct CSR_MESH_TTL_T`

CSR Mesh TTL.

`struct CSR_MESH_APPEARANCE_T`

CSRmesh Device Appearance. The Appearance is a 24-bit value that is composed of an "organization" and an "organization appearance".

`struct CSR_MESH_DEVICE_APPEARANCE_T`

Device Appearance data type.

`struct CSR_MESH_MASP_DEVICE_IND_T`

Device indication data type.

`struct CSR_MESH_ASSOCIATION_ATTENTION_DATA_T`

Association attention request data type.

`struct CSR_MESH_GROUP_ID_RELATED_DATA_T`

CSR Mesh Group Id Related Data - To provide to app while handling Group model data in core mesh.

`struct CSR_MESH_BEARER_STATE_DATA_T`

CSR Mesh Bearer state Type.

`struct CSR_MESH_SQN_LOOKUP_TABLE_T`

CSR Mesh <<SRC,SEQ>> Lookup Table.

`struct CSR_MESH_SEQ_CACHE_T`

CSR mesh <<SRC,SEQ>> Cache Structure.

`struct CSR_MESH_MESH_ID_DATA_T`

CSR Mesh mesh_id data Type.

`struct CSR_MESH_DEVICE_ID_UPDATE_T`

CSR Mesh device id update data Type.

`struct CSR_MESH_APP_EVENT_DATA_T`

CSR Mesh App Event Data - to provide event, status * app data to app.

`struct CSR_MESH_MASP_DEVICE_DATA_T`

CSRmesh Device Identifier Data and Signature.

`struct CSR_MESH_MASP_DEVICE_SIGN_LIST_NODE_T`

CSRmesh Device Signature Data Linked List node.

`struct CSR_SCHED_SCAN_WINDOW_PARAM_T`

CSR Mesh Scheduling Scan Window Parameter Type.

`struct CSR_SCHED_SCAN_DUTY_PARAM_T`

CSR Mesh Scheduling Scan Duty Parameter Type.

`struct CSR_SCHED_GENERIC_LE_PARAM_T`

CSR Mesh Scheduling Generic LE Parameter Type.

`struct CSR_SCHED_MESH_TX_BUFFER_T`

CSR Mesh Sched TX queue buffer Type.

`struct CSR_SCHED_MESH_TX_PARAM_T`

CSR Mesh Scheduling Tx Parameter Type.

`struct CSR_SCHED_MESH_LE_PARAM_T`

CSR Mesh Scheduling Mesh-LE Parameter Type.

`struct CSR_SCHED_LE_PARAMS_T`

CSR Mesh Scheduling LE Parameter Type.

`struct CSR_SCHED_ADV_PARAMS_T`

CSR Mesh Scheduling LE Advertisment Parameter Type.

`struct CSR_SCHED_ADV_DATA_T`

CSR Mesh Scheduling LE Advertising Data.

`struct CSR_SCHED_GATT_EVENT_DATA_T`

CSR Mesh Scheduling GATT Event Type.

6.2 Macros of Core Stack

```
#define CSR_MESH_NVM_SANITY_PSKEY_SIZE   (1)
```

Size definitions for persistent store values in words.

```
#define CSR_MESH_DEVICE_UUID_PSKEY_SIZE   (8)
```

Device UUID size in words.

```
#define CSR_MESH_DEVICE_AUTHCODE_PSKEY_SIZE   (4)
```

Device Authorization code size in words.

```
#define CSR_MESH_MTL_NW_KEY_AVL_PSKEY_SIZE   (1)
```

Network key avaialble flag size in words.

```
#define CSR_MESH_MTL_MCP_NWKEY_PSKEY_SIZE   (8)
```

Network key size in words.

```
#define CSR_MESH_MASP_RELAY_STATE_PSKEY_SIZE   (1)
```

Association messages relay state flag size in words.

```
#define CSR_MESH_MCP_SEQ_NUM_KEY_PSKEY_SIZE   (CSR_MESH_NO_OF_NWKS * 2)
```

Device last tx sequence number size in words.

```
#define CSR_MESH_CONFIG_LAST_ETAG_PSKEY_SIZE   (4)
```

Device last e-tag size in words.

```
#define CSR_MESH_DEVICE_ID_KEY_PSKEY_SIZE   (1)
```

Device ID size in words.

```
#define CSR_MESH_MTL_RELAY_STATUS_PSKEY_SIZE   (1)
```

Message relay status flag size in words.

```
#define CSR_MESH_DEVICE_DHM_PSKEY_SIZE   (12)
```

DHM Key size in words.

```
#define CSR_MESH_NW_IV_PSKEY_SIZE   (4)
```

Network IV size in words.

```
#define CSR_MESH_RESULT_SUCCESS   ((CSRmeshResult) 0x0000)
```

CSR Mesh Operation status Success.

```
#define CSR_MESH_RESULT_INPROGRESS   ((CSRmeshResult) 0x0001)
```

CSR Mesh Operation status InProgress.

```
#define CSR_MESH_RESULT_MESH_INVALID_STATE   ((CSRmeshResult) 0x0002)
```

CSR Mesh Operation status Mesh_Invalid_State.

```
#define CSR_MESH_RESULT_MODEL_NOT_REGISTERED   ((CSRmeshResult) 0x0003)
```

CSR Mesh Operation statusModel_Not_Registered.

```
#define CSR_MESH_RESULT_MODEL_ALREADY_REGISTERD   ((CSRmeshResult) 0x0004)
```

CSR Mesh Operation status Model_Already_Registered.

```
#define CSR_MESH_RESULT_ROLE_NOT_SUPPORTED   ((CSRmeshResult) 0x0005)
```

CSR Mesh Operation status Role_Not_Supported.

```
#define CSR_MESH_RESULT_INVALID_NWK_ID   ((CSRmeshResult) 0x0006)
```

CSR MeshOperation status Invalid_Network_id.

```
#define CSR_MESH_RESULT_EXCEED_MAX_NO_OF_NWKS   ((CSRmeshResult) 0x0007)
```

CSR MeshOperation status Exceed_Max_No_Of_Network.

```
#define CSR_MESH_RESULT_NOT_READY   ((CSRmeshResult) 0x0008)
```

CSR Mesh Operation statusNot_ready.

```
#define CSR_MESH_RESULT_MASP_ALREADY_ASSOCIATING   ((CSRmeshResult) 0x0009)
```

CSR Mesh Operation status Masp_Already_Associating.

```
#define CSR_MESH_RESULT_API_NOT_SUPPORTED   ((CSRmeshResult) 0x000A)
```

CSR Mesh Operation status Api_Not_Supported.

```
#define CSR_MESH_RESULT_INVALID_ARG   ((CSRmeshResult) 0x000B)
```

CSR Mesh Operation status Invalid argument.

```
#define CSR_MESH_RESULT_RADIO_BUSY   ((CSRmeshResult) 0x000C)
```

CSR Mesh Operation status radio busy.

```
#define CSR_MESH_RESULT_KEY_ALREADY_EXISTS   ((CSRmeshResult) 0x000D)
```

CSR Mesh Operation status key already present.

```
#define CSR_MESH_RESULT_FAILURE   ((CSRmeshResult) 0xFFFF)
```

CSR Mesh Operation status Failure.

```
#define CSR_MESH_MAX_NO_TIMERS   (7)
```

Number of timers required for CSRmesh library to be reserved by the application.

```
#define CSR_MESH_NO_OF_NWKS   (1)
```

Maximum number of mesh network the CSRmesh stack supports. The CSRmesh Stack library is built with this configuration. ** This should not be modified **.

```
#define CSR_MESH_INVALID_NWK_ID   (0xFF)
```

Invalid network id.

```
#define CSR_MESH_SHORT_NAME_LENGTH   (9)
```

short name for the device

```
#define MAX_ADV_DATA_LEN   (31)
```

This constant is used in the main server app to define array that is large enough to hold the advertisement data.

```
#define CSR_MESH_NETWORK_KEY_SIZE_BYTES   (16)
```

CSRmesh Network key in bytes.

```
#define CSR_MESH_MESH_ID_SIZE_BYTES   (16)
```

CSRmesh Mesh_Id size in bytes.

```
#define LE_BEARER_ACTIVE   (1)
```

LE Bearer Type.

```
#define GATT_SERVER_BEARER_ACTIVE   (2)
```

GATT Bearer Type.

```
#define CSR_MESH_DEVICE_SIGN_SIZE_IN_BYTES   (24)
```

Device Signature Size in Words and Bytes.

```
#define MAX_USER_ADV_DATA_LEN   (31)
```

Maximum User Advertising data length.

6.3 Typedefs of Core Stack

```
typedef CsrUint16 CSRmeshResult
```

CSR Mesh operation status Type.

```
typedef void(* CSR_MESH_APP_CB_T) (CSR_MESH_APP_EVENT_DATA_T eventDataCallback)
```

CSRmesh Application callback handler function.

```
typedef void(* CSR_SCHED_NOTIFY_GATT_CB_T) (CsrUint16 ucid, CsrUint8 *mtl_msg,
CsrUint8 length)
```

CSR Mesh Scheduler GATT notify handler function.

```
typedef void(* CSR_SCHED_USER_ADV_NOTIFY_CB_T) (CSR_SCHED_USER_ADV_STATUS_T
status, CsrUint32 time_us)
```

CSR Mesh Application callback handler function.

```
typedef void(* CSR_SCHED_PRIORITY_MSG_CB_T) (void)
```

CSR Mesh Application callback handler function.

6.4 Enumerations of Core Stack

```
enum  CSR_MESH_STACK_PS_KEY {
  CSR_MESH_NVM_SANITY_KEY = 0x00,
CSR_MESH_DEVICE_UUID_KEY = 0x01,
CSR_MESH_DEVICE_AUTHCODE_KEY = 0x02,
CSR_MESH_MTL_NW_KEY_AVL_KEY = 0x03,
  CSR_MESH_MTL_MCP_NWKEY_KEY = 0x04,
CSR_MESH_MASP_RELAY_STATE_KEY = 0x05,
CSR_MESH_MCP_SEQ_NUM_KEY = 0x06,
CSR_MESH_MCP_GEN_SEQ_NUM_KEY = 0x07,
  CSR_MESH_CONFIG_LAST_ETAG_KEY = 0x08,
CSR_MESH_DEVICE_ID_KEY = 0x09,
CSR_MESH_MTL_RELAY_STATUS_KEY = 0x0A,
CSR_MESH_DEVICE_DHM_KEY = 0x0B,
  CSR_MESH_NW_IV_KEY = 0x0C
}
```

Access Keys for Stack parameters to be saved on persistent store.

```
enum  CSR_MESH_CONFIG_FLAG_T { CSR_MESH_CONFIG_DEVICE = 0,
CSR_MESH_NON_CONFIG_DEVICE = 1,
CSR_MESH_CONFIG_DEVICE_WITH_AUTH_CODE = 2,
CSR_MESH_NON_CONFIG_DEVICE_WITH_AUTH_CODE = 3
}
```

Flag determining the type of the device.

```
enum  CSR_MESH_OPERATION_STATUS_T {
  CSR_MESH_OPERATION_SUCCESS = 0x00,
CSR_MESH_OPERATION_STACK_NOT_INITIALIZED = 0x01,
CSR_MESH_OPERATION_NOT_PERMITTED = 0x02,
CSR_MESH_OPERATION_MEMORY_FULL = 0x03,
  CSR_MESH_OPERATION_REQUEST_FOR_INFO = 0x04,
CSR_MESH_OPERATION_GENERIC_FAIL = 0xFF
}
```

Operation status passed with App-Callback to indicate the result of an asynchronous operation or to inform the App that the callback is mader to request info.

```
enum  CSR_MESH_MESSAGE_T { CSR_MESH_MESSAGE_ASSOCIATION,
CSR_MESH_MESSAGE_CONTROL
}
```

CSRmesh Message types.

```
enum  CSR_MESH_EVENT_T {
  CSR_MESH_INIT_EVENT = 0x0001,
CSR_MESH_REGISTER_APP_CB_EVENT = 0x0002,
CSR_MESH_RESET_EVENT = 0x0003,
CSR_MESH_START_EVENT = 0x0004,
  CSR_MESH_STOP_EVENT = 0x0005,
CSR_MESH_TRANSMIT_STATE_EVENT = 0x0006,
CSR_MESH_START_DEVICE_INQUIRY_EVENT = 0x0007,
CSR_MESH_ASSOC_STARTED_EVENT = 0x0008,
  CSR_MESH_ASSOC_COMPLETE_EVENT = 0x0009,
CSR_MESH_SEND_ASSOC_COMPLETE_EVENT = 0x000A,
CSR_MESH_GET_DEVICE_ID_EVENT = 0x000B,
CSR_MESH_GET_DEVICE_UUID_EVENT = 0x000C,
  CSR_MESH_MASP_DEVICE_IND_EVENT = 0x000D,
CSR_MESH_MASP_DEVICE_APPEARANCE_EVENT = 0x000E,
CSR_MESH_NETWORK_ID_LIST_EVENT = 0x000F,
CSR_MESH_SET_MAX_NO_OF_NETWORK_EVENT = 0x0010,
  CSR_MESH_SET_PASSPHRASE_EVENT = 0x0011,
CSR_MESH_SET_NETWORK_KEY_EVENT = 0x0012,
CSR_MESH_CONFIG_RESET_DEVICE_EVENT = 0x0013,
CSR_MESH_CONFIG_SET_PARAMS_EVENT = 0x0014,
  CSR_MESH_CONFIG_GET_PARAMS_EVENT = 0x0015,
CSR_MESH_GET_VID_PID_VERSTION_EVENT = 0x0016,
CSR_MESH_GET_DEVICE_APPEARANCE_EVENT = 0x0017,
CSR_MESH_GROUP_SET_MODEL_GROUPID_EVENT = 0x0018,
  CSR_MESH_SEND_RAW_MCP_MSG_EVENT = 0x0019,
CSR_MESH_SEND_MCP_MSG_EVENT = 0x001A,
CSR_MESH_MCP_REGISTER_MODEL_EVENT = 0x001B,
CSR_MESH_MCP_REGISTER_MODEL_CLIENT_EVENT = 0x001C,
  CSR_MESH_REMOVE_NETWORK_EVENT = 0x001D,
CSR_MESH_GET_DIAG_DATA_EVENT = 0x001E,
CSR_MESH_RESET_DIAG_DATA_EVENT = 0x001F,
CSR_MESH_REGISTER_SNIFFER_APP_CB_EVENT = 0x0020,
  CSR_MESH_GET_MESH_ID_EVENT = 0x0021,
CSR_MESH_GET_NET_ID_FROM_MESH_ID_EVENT = 0x0022,
CSR_MESH_ASSOCIATION_ATTENTION_EVENT = 0x0023,
CSR_MESH_BEARER_STATE_EVENT = 0x0024,
  CSR_MESH_NW_IV_UPDATED_ALREADY = 0x0025,
CSR_MESH_NW_IV_UPDATE_STARTED = 0x0026,
CSR_MESH_NW_IV_WRONG_RSP_RECEIVED = 0x0027,
CSR_MESH_NW_KEY_IV_COMPLETE_EVENT = 0x0028,
  CSR_MESH_NW_KEY_IV_FAILED_EVENT = 0x0029,
CSR_MESH_GET_CONFORMANCE_SIGNATURE_EVENT = 0x0030,
CSR_MESH_GET_NETWORK_KEY_EVENT = 0x0031,
CSR_MESH_GET_DHM_KEY_EVENT = 0x0032,
  CSR_MESH_DEVICE_ID_UPDATE_EVENT = 0x0033,
CSR_MESH_GET_DEFAULT_TTL_EVENT = 0x0034,
CSR_MESH_CONFIG_SET_MSG_PARAMS_EVENT = 0x0035,
CSR_MESH_CONFIG_GET_MSG_PARAMS_EVENT = 0x0036,
  CSR_MESH_SEQ_NUM_ROLL_BACK_EVENT = 0x0037,
CSR_MESH_MASP_DEVICE_CONF_ARGS_EVENT = 0x0038,
CSR_MESH_MASP_CLIENT_CONFIRMATION_EVENT = 0x0039,
```

```
CSR_MESH_INVALID_EVENT = 0xFFFF
}
```

CSRmesh event types.

```
enum  CSR_SCHED_SCAN_TYPE_T { CSR_SCHED_SCAN_WINDOW_PARAM = 0x00,
CSR_SCHED_SCAN_DUTY_PARAM = 0x01
}
```

CSR Mesh Scheduling Scan Parameter Type.

```
enum  CSR_SCHED_GATT_EVENT_T { CSR_SCHED_GATT_CONNECTION_EVENT,
CSR_SCHED_GATT_STATE_CHANGE_EVENT,
CSR_SCHED_GATT_CCCD_STATE_CHANGE_EVENT
}
```

CSR Mesh Scheduling LE Event Type.

```
enum  CSR_SCHED_INCOMING_DATA_EVENT_T { CSR_SCHED_INCOMING_LE_MESH_DATA_EVENT =
1,
CSR_SCHED_INCOMING_GATT_MESH_DATA_EVENT = 2,
CSR_SCHED_SET_LE_ADV_PKT_EVENT = 3
}
```

CSR Mesh Scheduling LE Incoming data event Type.

```
enum  CSRSchedResult {
  CSR_SCHED_RESULT_SUCCESS = 0x0000,
CSR_SCHED_RESULT_INVALID_HANDLE = 0x0001,
CSR_SCHED_RESULT_UNACCEPTABLE_ADV_DURATION = 0x0002,
CSR_SCHED_RESULT_UNACCEPTABLE_ADV_INTERVAL = 0x0003,
  CSR_SCHED_RESULT_INCORRECT_SCAN_PARAM = 0x0004,
CSR_SCHED_RESULT_SCAN_NOT_STARTED = 0x0005
, CSR_SCHED_RESULT_FAILURE = 0xFFFF
}
```

CSR Mesh Scheduling operation status.

6.5 Functions of Core Stack

```
CSRmeshResult CSRmeshInit (CSR_MESH_CONFIG_FLAG_T configFlag)
```

Initialize CSRmesh core mesh stack.

```
CSRmeshResult CSRmeshRegisterAppCallback (CSR_MESH_APP_CB_T callback)
```

Register application callback.

```
CSRmeshResult CSRmeshStart (void)
```

Start the CSRmesh system.

```
CSRmeshResult CSRmeshStop (void)
```

Stop the CSRmesh system.

```
CSRmeshResult CSRmeshReset (void)
```

Reset the CSRmesh library.

```
CSRmeshResult CSRmeshSetDefaultTTL (CsrUint8 ttl)
```

Sets a default ttl value.

```
CSRmeshResult CSRmeshGetDefaultTTL (CSR_MESH_APP_EVENT_DATA_T *eventData)
```

Gets the default ttl value.

```
CSRmeshResult CSRmeshRemoveNetwork (CsrUint8 netId)
```

Remove a network.

```
CSRmeshResult CSRmeshGetDeviceID (CsrUint8 netId, CSR_MESH_APP_EVENT_DATA_T
*eventData)
```

Gets the 16-bit Device Identifier of the CSRmesh device.

```
CSRmeshResult CSRmeshGetDeviceUUID (CSR_MESH_APP_EVENT_DATA_T *eventData)
```

Get the CSRmesh library 128 bit UUID.

```
CSRmeshResult CSRmeshSetTransmitState (CSR_MESH_TRANSMIT_STATE_T
*transmitStateArg, CSR_MESH_APP_EVENT_DATA_T *eventData)
```

CSRmeshSetTransmitState .

```
CSRmeshResult CSRmeshGetTransmitState (CsrUint8 netId,
CSR_MESH_APP_EVENT_DATA_T *eventData)
```

CSRmeshGetTransmitState .

```
CSRmeshResult CSRmeshAssociateToANetwork (CSR_MESH_DEVICE_APPEARANCE_T
*deviceAppearance, CsrUint8 ttl)
```

Advertises a CSRmesh device identification message.

```
void CSRmeshCalculateSHA256Hash (const CsrUint8 *string, CsrUint8 length,
CsrUint16 *hash)
```

This function generates a 32 byte hash.

```
CSRmeshResult CSRmeshSetMeshDuplicateMessageCacheSize (CsrUint8 cacheSize)
```

Controls the depth of the duplicate check cache in MTL.

```
CSRmeshResult CSRmeshSendKeyIVStatusRequest (const CSR_MESH_MASP_DEVICE_DATA_T
*devData, CsrUint8 ttl)
```

Transmit Key-IV request.

```
CSRmeshResult CSRmeshSetSrcSequenceCache (CsrUint8 netId, CSR_MESH_SEQ_CACHE_T
*seqCache)
```

CSRmeshSetSequenceSrcCache.

```
CSRmeshResult CSRmeshPsInit (void)
```

This method is called by the stack to initialize the PS interface if any.

```
CsrBool CSRmeshIsPsReadyForAccess (void)
```

CSRmesh stack calls this function to check if the persistant store contains valid values. The platform must validate the contents of the persistent store. If the persistent store is corrupt or it is brought up for the first time, this function must return FALSE. True otherwise. If this function returns FALSE the stack will reset its PS values and writes them on the store.

```
CSRmeshResult CSRmeshPsRead (CsrUint8 key, CsrUint16 *p_value, CsrUint16 length)
```

This function is called to read a value associated with the key from the persistant store.

```
CSRmeshResult CSRmeshPsWrite (CsrUint8 key, CsrUint16 *p_value, CsrUint16
length)
```

This function is called to write a value associated with the key from the persistant store.

```
CSRmeshResult CSRmeshPsSecureWrite (CsrUint8 key, CsrUint16 *p_value, CsrUint16
length)
```

This function is called to write a value associated with the key from a secure persistant store. This function is called for parameters such as passphrase, network keys and other security keys. If the platform supports a secure storage, this value must be stored on the secure store.

```
CSRmeshResult CSRmeshPsSecureRead (CsrUint8 key, CsrUint16 *valueBuffer,
CsrUint16 length)
```

This function is called to read a value associated with the key from a secure persistant store. If the platform supports a secure storage, this value must be read from the secure store.

```
void CSRmeshPsDeInit (void)
```

This function is called when the CSRmesh stack is reset. This function may reset any variables initialized for the CSRmesh persistent store API and free any allocated buffers.

```
CsrUint16 * CSRmeshPsReadSecureKey (void)
```

The CSRmesh stack obfuscates the secure paramters before writing to the platform persistent store. If the platform supports.

```
CSRSchedResult CSRSchedSetScanDutyCycle (CsrUint16 scan_duty_cycle, CsrUint16
new_scan_slot)
```

Scan duty cycle can be configured using this API.

```
CsrUint16 CSRSchedGetScanDutyCycle (void)
```

Current scan duty cycle can be fetched using this API.

```
CSRSchedResult CSRSchedSetConfigParams (CSR_SCHED_LE_PARAMS_T *le_params)
```

Configure Mesh and Generic LE parameters.

```
CSRSchedResult CSRSchedSendUserAdv (CSR_SCHED_ADV_DATA_T *le_adv_data,
CSR_SCHED_USER_ADV_NOTIFY_CB_T callBack)
```

Transmit Non-Connectable/Connectable Application data.

```
CSRSchedResult CSRSchedGetConfigParams (CSR_SCHED_LE_PARAMS_T *le_params)
```

Get scheduler configuration parameters.

```
void CSRSchedRegisterPriorityMsgCb (CSR_SCHED_PRIORITY_MSG_CB_T cb_ptr)
```

Register callback function to handle radio busy error for priority message. The callback will be called when the radio is available for transmission.

```
CSRSchedResult CSRSchedSendPriorityMsg (CsrUint8 mesh_packet_type, CsrUint8
*payload, CsrUint8 length)
```

This function sends a CSRmesh message with priority.

```
CSRSchedResult CSRSchedHandleIncomingData (CSR_SCHED_INCOMING_DATA_EVENT_T
data_event, CsrUint8 *data, CsrUint8 length, CsrInt8 rssi)
```

Forward Mesh messages to the scheduler.

```
CSRSchedResult CSRSchedEnableListening (CsrBool enable)
```

Starts or stops LE scan operation.

```
CsrBool IsCSRSchedRunning (void)
```

Indicates about the scheduler state.

```
CSRSchedResult CSRSchedStart (void)
```

Start scheduling of LE scan and advertisement.

```
void CSRSchedNotifyGattEvent (CSR_SCHED_GATT_EVENT_T gatt_event_type,
CSR_SCHED_GATT_EVENT_DATA_T *gatt_event_data, CSR_SCHED_NOTIFY_GATT_CB_T
call_back)
```

Initialize CSRmesh core mesh stack.

```
void CsrSchedSetTxPower (CsrUint8 level)
```

Sets the Transmit Power for the Device.

6.6 CSR_MESH_NVM_SANITY_PSKEY_SIZE Macro Definition Documentation

Definition

```
#define CSR_MESH_NVM_SANITY_PSKEY_SIZE   (1)
```

Description

NVM Sanity word size in words

6.7 CSR_MESH_MAX_NO_TIMERS Macro Definition Documentation

Definition

```
#define CSR_MESH_MAX_NO_TIMERS   (7)
```

Description

User application needs to reserve these many timers along with application timers. Required for CSR1010 only. Example:

```
1 #define MAX_APP_TIMERS (3 + CSR_MESH_MAX_NO_TIMERS)
```

6.8 CSR_MESH_STACK_PS_KEY Enumeration Type Documentation

Syntax

```
enum CSR_MESH_STACK_PS_KEY
```

Description

Enumerations

Enumeration	Description
CSR_MESH_NVM_SANITY_KEY	PS access key for sanity word.
CSR_MESH_DEVICE_UUID_KEY	PS access key for Device UUID.
CSR_MESH_DEVICE_AUTHCODE_KEY	PS access key for Device Authorization Code.
CSR_MESH_MTL_NW_KEY_AVL_KEY	PS access key for network key validity flag.
CSR_MESH_MTL_MCP_NWKEY_KEY	PS access key for MCP NWKEY.
CSR_MESH_MASP_RELAY_STATE_KEY	PS access key for Association messages relay state flag.
CSR_MESH_MCP_SEQ_NUM_KEY	PS access key for last stored sequence number.
CSR_MESH_MCP_GEN_SEQ_NUM_KEY	PS access key for intial sequence number.
CSR_MESH_CONFIG_LAST_ETAG_KEY	PS access key for Config Last e-tag.
CSR_MESH_DEVICE_ID_KEY	PS access key for device identifier. Used only when CSRmesh stack supports single network.
CSR_MESH_MTL_RELAY_STATUS_KEY	PS access key for MTL relay status setting. Used only when CSRmesh stack supports single network.
CSR_MESH_DEVICE_DHM_KEY	PS access key for device DHM_KEY. Used only when CSRmesh stack supports single network.
CSR_MESH_NW_IV_KEY	PS access key for network IV. Used only when CSRmesh stack supports single network.

6.9 CSR_MESH_CONFIG_FLAG_T Enumeration Type Documentation

Syntax

```
enum CSR_MESH_CONFIG_FLAG_T
```

Description

Enumerations

Enumeration	Description
CSR_MESH_CONFIG_DEVICE	CSRmesh configuring device type.
CSR_MESH_NON_CONFIG_DEVICE	CSRmesh non-configuring device type.
CSR_MESH_CONFIG_DEVICE_WITH_AUTH_CODE	CSRmesh config device with auth code type.
CSR_MESH_NON_CONFIG_DEVICE_WITH_AUTH_CODE	CSRmesh non-config device with auth code type.

6.10 CSR_MESH_OPERATION_STATUS_T Enumeration Type Documentation

Syntax

```
enum CSR_MESH_OPERATION_STATUS_T
```

Description

Enumerations

Enumeration	Description
CSR_MESH_OPERATION_SUCCESS	Operation status success.
CSR_MESH_OPERATION_STACK_NOT_INITIALIZED	Operation status stack not initialized.

Enumeration	Description
CSR_MESH_OPERATION_NOT_PERMITTED	Operation status operation not permited.
CSR_MESH_OPERATION_MEMORY_FULL	Operation status memory full.
CSR_MESH_OPERATION_REQUEST_FOR_INFO	Operation status request for info to app.
CSR_MESH_OPERATION_GENERIC_FAIL	Operation status generic fail.

6.11 CSR_MESH_MESSAGE_T Enumeration Type Documentation

Syntax

```
enum CSR_MESH_MESSAGE_T
```

Description

Enumerations

Enumeration	Description
CSR_MESH_MESSAGE_ASSOCIATION	CSRmesh Association message.
CSR_MESH_MESSAGE_CONTROL	CSRmesh Control message.

6.12 CSR_MESH_EVENT_T Enumeration Type Documentation

Syntax

```
enum CSR_MESH_EVENT_T
```

Description

Enumerations

Enumeration	Description
CSR_MESH_INIT_EVENT	Type Mesh stack init event.
CSR_MESH_REGISTER_APP_CB_EVENT	Type Register-App event.
CSR_MESH_RESET_EVENT	Type Reset event.
CSR_MESH_START_EVENT	Type Start event.
CSR_MESH_STOP_EVENT	Type Stop event.
CSR_MESH_TRANSMIT_STATE_EVENT	Type Bearer State event.
CSR_MESH_START_DEVICE_INQUIRY_EVENT	Type Device Enquiry event.
CSR_MESH_ASSOC_STARTED_EVENT	Type Assoc Started event.
CSR_MESH_ASSOC_COMPLETE_EVENT	Type Assoc Complete event.
CSR_MESH_SEND_ASSOC_COMPLETE_EVENT	Type Send Assoc Complete event.
CSR_MESH_GET_DEVICE_ID_EVENT	Type Get Device Id event.
CSR_MESH_GET_DEVICE_UUID_EVENT	Type Get Device Uuid event.
CSR_MESH_MASP_DEVICE_IND_EVENT	Type Masp Device Ind event.
CSR_MESH_MASP_DEVICE_APPEARANCE_EVENT	Type Masp Device Appearance event.
CSR_MESH_NETWORK_ID_LIST_EVENT	Type Network Id List event.
CSR_MESH_SET_MAX_NO_OF_NETWORK_EVENT	Type Max No of Network event.
CSR_MESH_SET_PASSPHRASE_EVENT	Type Set Passphrase event.
CSR_MESH_SET_NETWORK_KEY_EVENT	Type Set Network Key event.
CSR_MESH_CONFIG_RESET_DEVICE_EVENT	Type Config Reset Device event.

Enumeration	Description
CSR_MESH_CONFIG_SET_PARAMS_EVENT	Type Config Set Params event.
CSR_MESH_CONFIG_GET_PARAMS_EVENT	Type Config Get Params event.
CSR_MESH_GET_VID_PID_VERSTION_EVENT	Type Get Vid-Pid Version event.
CSR_MESH_GET_DEVICE_APPEARANCE_EVENT	Type Get Device Appearance event.
CSR_MESH_GROUP_SET_MODEL_GROUPID_EVENT	Type Group Set Model Group-id event.
CSR_MESH_SEND_RAW_MCP_MSG_EVENT	Type Send Raw MCP Msg event.
CSR_MESH_SEND_MCP_MSG_EVENT	Type Send MCP Msg event.
CSR_MESH_MCP_REGISTER_MODEL_EVENT	Type MCP Register Model event.
CSR_MESH_MCP_REGISTER_MODEL_CLIENT_EVENT	Type MCP Register Client Model event.
CSR_MESH_REMOVE_NETWORK_EVENT	Type Remove Network event.
CSR_MESH_GET_DIAG_DATA_EVENT	Type Get Diagnostic Data event.
CSR_MESH_RESET_DIAG_DATA_EVENT	Type Reset Diagnostic Data event.
CSR_MESH_REGISTER_SNIFFER_APP_CB_EVENT	Type Register Sniffer App Cb event.
CSR_MESH_GET_MESH_ID_EVENT	Type Get Mesh Id Data event.
CSR_MESH_GET_NET_ID_FROM_MESH_ID_EVENT	Type Get Network id from Mesh Id event.
CSR_MESH_ASSOCIATION_ATTENTION_EVENT	Type Association attention event.
CSR_MESH_BEARER_STATE_EVENT	Type Bearer state is updated.

Enumeration	Description
CSR_MESH_NW_IV_UPDATED_ALREADY	Type nw_iv update status event.
CSR_MESH_NW_IV_UPDATE_STARTED	Type nw_iv update start status event.
CSR_MESH_NW_IV_WRONG_RSP_RECEIVED	Type nw_iv update start status event.
CSR_MESH_NW_KEY_IV_COMPLETE_EVENT	Type Rekeying Complete event.
CSR_MESH_NW_KEY_IV_FAILED_EVENT	Type Rekeying failed event.
CSR_MESH_GET_CONFORMANCE_SIGNATURE_EVENT	Type Get conformance signature event.
CSR_MESH_GET_NETWORK_KEY_EVENT	Type Get Network Key.
CSR_MESH_GET_DHM_KEY_EVENT	Type Get DHM Key.
CSR_MESH_DEVICE_ID_UPDATE_EVENT	Type Device id update event.
CSR_MESH_GET_DEFAULT_TTL_EVENT	Type Get Default ttl event.
CSR_MESH_CONFIG_SET_MSG_PARAMS_EVENT	Type Config Set Message Params event.
CSR_MESH_CONFIG_GET_MSG_PARAMS_EVENT	Type Config Get Message Params event.
CSR_MESH_SEQ_NUM_ROLL_BACK_EVENT	Type Sequence number roll back event.
CSR_MESH_MASP_DEVICE_CONF_ARGS_EVENT	device confirmation arguments event
CSR_MESH_MASP_CLIENT_CONFIRMATION_EVENT	device confirmation arguments event
CSR_MESH_INVALID_EVENT	Type Invalid event.

6.13 CSR_SCHED_SCAN_TYPE_T Enumeration Type Documentation

Syntax

```
enum CSR_SCHED_SCAN_TYPE_T
```

Description

Enumerations

Enumeration	Description
CSR_SCHED_SCAN_WINDOW_PARAM	scan parameter contains scan window and scan interval
CSR_SCHED_SCAN_DUTY_PARAM	scan parameter contains scan duty cycle and min scan slot

6.14 CSR_SCHED_GATT_EVENT_T Enumeration Type Documentation

Syntax

```
enum CSR_SCHED_GATT_EVENT_T
```

Description

Enumerations

Enumeration	Description
CSR_SCHED_GATT_CONNECTION_EVENT	GATT Connection event.
CSR_SCHED_GATT_STATE_CHANGE_EVENT	GATT State change event.
CSR_SCHED_GATT_CCCD_STATE_CHANGE_EVENT	GATT client characteristics configuration descriptor state change event.

6.15 CSR_SCHED_INCOMING_DATA_EVENT_T Enumeration Type Documentation

Syntax

```
enum CSR_SCHED_INCOMING_DATA_EVENT_T
```

Description

Enumerations

Enumeration	Description
CSR_SCHED_INCOMING_LE_MESH_DATA_EVENT	Incoming Mesh data coming from LE ADV event.
CSR_SCHED_INCOMING_GATT_MESH_DATA_EVENT	Incoming Mesh data coming from GATT connection event.
CSR_SCHED_SET_LE_ADV_PKT_EVENT	Incomding Mesh data coming on Gatt Connection and sent over LE ADV event.

6.16 CSRSchedResult Enumeration Type Documentation

Syntax

```
enum CSRSchedResult
```

Description

Enumerations

Enumeration	Description
CSR_SCHED_RESULT_SUCCESS	Operation result success.
CSR_SCHED_RESULT_INVALID_HANDLE	Operation result invalid handle.

Enumeration	Description
CSR_SCHED_RESULT_UNACCEPTABLE_ADV_DURATION	Operation result unacceptable advertisment duration.
CSR_SCHED_RESULT_UNACCEPTABLE_ADV_INTERVAL	Operation result unacceptable advertisment interval.
CSR_SCHED_RESULT_INCORRECT_SCAN_PARAM	Operation result incorrect san params.
CSR_SCHED_RESULT_SCAN_NOT_STARTED	Operation result scan not started.
CSR_SCHED_RESULT_FAILURE	Operation status for failure.

6.17 CSRmeshInit Function Documentation

Syntax

```
CSRmeshResult CSRmeshInit (CSR_MESH_CONFIG_FLAG_T configFlag)
```

Description

Initialize MASP, MCP & MTL Layers.

Parameters

Parameter	Description
configFlag	The configuration flag identifies the role of the device (configuring / non-configuring).

Returns

CSRmeshResult. Refer to CSRmeshResult.

6.18 CSRmeshRegisterAppCallback Function Documentation

Syntax

```
CSRmeshResult CSRmeshRegisterAppCallback (CSR_MESH_APP_CB_T callback)
```

Description

This API registers an application call-back to the stack. All the stack events are notified to this registerd call-back. For supporting mutiple applications a seperate multiplexer module is required. The multiplexer module should take care of notifying the stack event to all the interested applications.

Parameters

Parameter	Description
`callback`	application call-back to register

Returns

CSRmeshResult. Refer to CSRmeshResult.

6.19 CSRmeshStart Function Documentation

Syntax

```
CSRmeshResult CSRmeshStart (void )
```

Description

Initializes Bearer Layer Successful completion is indication that stack is ready for Rx/Tx.

Returns

CSRmeshResult. Refer to CSRmeshResult.

6.20 CSRmeshStop Function Documentation

Syntax

```
CSRmeshResult CSRmeshStop (void )
```

Description

Stops processing the CSRmesh messages and does not allow sending mesh messages. It does not stop LE Scanning. The Scheduler API CSRSchedEnableListening should be called to disable scanning.

Returns

CSRmeshResult. Refer to CSRmeshResult.

6.21 CSRmeshReset Function Documentation

Syntax

```
CSRmeshResult CSRmeshReset (void )
```

Description

Resets the CSRmesh library and clears all data relevant to each layer. The network association info is retained which can be removed only using the config model.

Returns

CSRmeshResult. Refer to CSRmeshResult.

6.22 CSRmeshSetDefaultTTL Function Documentation

Syntax

```
CSRmeshResult CSRmeshSetDefaultTTL (CsrUint8 ttl)
```

Description

This API sets the default ttl value to be used by the core models to send ACK message.

Parameters

Parameter	Description
ttl	Time to live value to be used for ACK messages.

Returns

CSRmeshResult. Refer to CSRmeshResult.

6.23 CSRmeshGetDefaultTTL Function Documentation

Syntax

```
CSRmeshResult CSRmeshGetDefaultTTL (CSR_MESH_APP_EVENT_DATA_T * eventData)
```

Description

This API gets the default ttl value being used by the core models to send ACK message.

Parameters

Parameter	Description
eventData	On successful completion the default ttl will be available to the app as part of eventData param.

Returns

CSRmeshResult. Refer to CSRmeshResult.

6.24 CSRmeshRemoveNetwork Function Documentation

Syntax

```
CSRmeshResult CSRmeshRemoveNetwork (CsrUint8 netId)
```

Description

This API removes the network key associated with the network id from the device.

Parameters

Parameter	Description
netId	Network id of the network to be removed

Returns

CSRmeshResult. Refer to CSRmeshResult.

6.25 CSRmeshGetDeviceID Function Documentation

Syntax

```
CSRmeshResult CSRmeshGetDeviceID (CsrUint8 netId, CSR_MESH_APP_EVENT_DATA_T
* eventData)
```

Description

Parameters

Parameter	Description
netId	
eventData	On successful completion a device id will be available to the app as a field in this 'result parameter'.

Returns

CSRmeshResult. Refer to CSRmeshResult.

6.26 CSRmeshGetDeviceUUID Function Documentation

Syntax

```
CSRmeshResult CSRmeshGetDeviceUUID (CSR_MESH_APP_EVENT_DATA_T * eventData)
```

Description

Parameters

Parameter	Description
eventData	On successful completion a uuid id will be available to the app as a field in this 'result parameter'.

Returns

CSRmeshResult. Refer to CSRmeshResult.

6.27 CSRmeshSetTransmitState Function Documentation

Syntax

```
CSRmeshResult CSRmeshSetTransmitState (CSR_MESH_TRANSMIT_STATE_T
* transmitStateArg, CSR_MESH_APP_EVENT_DATA_T * eventData)
```

Description

This API sets the bearer relay, bearer enable and promiscuous states for Mesh Bearer layer, also sets relay and promiscuous state of MTL. MTL and Mesh Bearer layer based on this configuration will allow reception, transmission of mesh messages and also relay of known and unknown mesh messages. Parameters of this API identifies a 16- bit bitfield for bearer relay active, bearer enabled and bearer promiscuous arguments.

Note: Setting bit '0' in the corresponding bit mask will disable the corresponding functionality if it is already active.

Parameters

Parameter	Description
`transmitStateArg`	structure containing all the transmit state arguments.
`eventData`	On successful completion, transmit state will be available to the app. as a field in this 'result parameter'.

Returns

CSRmeshResult. Refer to CSRmeshResult.

6.28 CSRmeshGetTransmitState Function Documentation

Syntax

```
CSRmeshResult CSRmeshGetTransmitState (CsrUint8 netId,
CSR_MESH_APP_EVENT_DATA_T * eventData)
```

Description

This function reads transmit state arguments of CSRmesh network.

Parameters

Parameter	Description
`netId`	Network id of the network for which the relay is required to be enabled/disabled.
`eventData`	On successful completion transmit state will be available to the app. as a field in this 'result parameter'.

Returns

CSRmeshResult. Refer to CSRmeshResult.

6.29 CSRmeshAssociateToANetwork Function Documentation

Syntax

```
CSRmeshResult CSRmeshAssociateToANetwork (CSR_MESH_DEVICE_APPEARANCE_T
* deviceAppearance, CsrUint8 ttl)
```

Description

Application uses this API to appear as a New Device. On successful execution of this API the device sends MASP_DEVICE_IDENTIFICATION message with its 128bit UUID.

Parameters

Parameter	Description
deviceAppearance	The deviceAppearance of the device. This is a pointer to CSR_MESH_DEVICE_APPEARANCE_T structure.
ttl	Time to live value used for MASP.

Returns

CSRmeshResult. Refer to CSRmeshResult.

6.30 CSRmeshCalculateSHA256Hash Function Documentation

Syntax

```
void CSRmeshCalculateSHA256Hash (const CsrUint8 * string, CsrUint8 length,
CsrUint16 * hash)
```

Description

Application uses this API to generate a 32 byte secure hash.

Parameters

Parameter	Description
string	The data for which hash to be generated.
length	The length of the data.
hash	The generated hash output.

Returns

Nothing.

6.31 CSRmeshSetMeshDuplicateMessageCacheSize Function Documentation

Syntax

```
CSRmeshResult CSRmeshSetMeshDuplicateMessageCacheSize (CsrUint8 cacheSize)
```

Description

Application uses this API to control the depth of message cache in ..MTL (dynamically) [0..255]

Parameters

Parameter	Description
cacheSize	Size of message cache [0 to 255]

Returns

CSRmeshResult. Refer to CSRmeshResult.

6.32 CSRmeshSendKeyIVStatusRequest Function Documentation

Syntax

```
CSRmeshResult CSRmeshSendKeyIVStatusRequest (const CSR_MESH_MASP_DEVICE_DATA_T
* devData, CsrUint8 ttl)
```

Description

This API Transmit KeyIV status request for Newtork re-keying or for IV Update.

Parameters

Parameter	Description
devData	Pointer to MASP device data structure. This needs to be filled with relevant data like Network Id, self (for NON-CONFIG role) or remote (for CONFIG role) Device Id and Device Signature (Only for CONFIG role and NULL for NON-CONFIG role).
ttl	TTL for sending this message.

Returns

CSRmeshResult. Refer to CSRmeshResult.

6.33 CSRmeshSetSrcSequenceCache Function Documentation

Syntax

```
CSRmeshResult CSRmeshSetSrcSequenceCache (CsrUint8 netId, CSR_MESH_SEQ_CACHE_T
* seqCache)
```

Description

This function configures the src , sqn cache in mesh stack.

Parameters

Parameter	Description
netId	Id of the network to which Sequence Table is used.
seqCache	Pointer to the memory allocated for the source, sequence cache. This parameter can be NULL to disable the Sequence Cache for a network. In this parameter, all the data members need to be filled in by the user. Example usage shown here: 1 #define HOME_NW_ID (0) 2 #define NUM_SEQ_LUT_ENTRIES (16) 3 4 CSRmeshResult result; 5 CSR_MESH_SQN_LOOKUP_TABLE_T home_nw_seq_lut[NUM_SEQ_LUT_ENTRIES]; 6 CSR_MESH_SEQ_CACHE_T home_nw_seq_cache; 7 8 home_nw_seq_cache.cached_dev_count = NUM_SEQ_LUT_ENTRIES; 9 home_nw_seq_cache.seq_deviation = 32; 10 home_nw_seq_cache.seq_lookup_table = home_nw_seq_lut; 11 12 result = CSRmeshSetSrcSequenceCache(HOME_NW_ID, &home_nw_seq_cache);

Returns

CSRmeshResult. Refer to CSRmeshResult.

6.34 CSRmeshPsInit Function Documentation

Syntax

```
CSRmeshResult CSRmeshPsInit (void )
```

Description

Returns

CSRmeshResult

6.35 CSRmeshIsPsReadyForAccess Function Documentation

Syntax

```
CsrBool CSRmeshIsPsReadyForAccess (void )
```

Description

Returns

TRUE if the contents of the Persistent store are valid.

6.36 CSRmeshPsRead Function Documentation

Syntax

```
CSRmeshResult CSRmeshPsRead (CsrUint8 key, CsrUint16 * p_value,
CsrUint16 length)
```

Description

Parameters

Parameter	Description
key	Identifier of the Persistent store parameter to be read
p_value	Pointer to a buffer to copy the value
length	Length of the paramter in words

Returns

CSRmeshResult

6.37 CSRmeshPsWrite Function Documentation

Syntax

```
CSRmeshResult CSRmeshPsWrite (CsrUint8 key, CsrUint16 * p_value,
CsrUint16 length)
```

Description

Parameters

Parameter	Description
key	Identifier of the Persistent store parameter to be written
p_value	Pointer to a buffer containing the value to be written
length	Length of the paramter in words

Returns

CSRmeshResult

6.38 CSRmeshPsSecureWrite Function Documentation

Syntax

```
CSRmeshResult CSRmeshPsSecureWrite (CsrUint8 key, CsrUint16 * p_value,
CsrUint16 length)
```

Description

Parameters

Parameter	Description
`key`	Identifier of the Persistent store parameter to be written
`p_value`	Pointer to a buffer containing the value to be written
`length`	Length of the paramter in words

Returns

CSRmeshResult

6.39 CSRmeshPsSecureRead Function Documentation

Syntax

```
CSRmeshResult CSRmeshPsSecureRead (CsrUint8 key, CsrUint16 * valueBuffer,
CsrUint16 length)
```

Description

Parameters

Parameter	Description
key	Identifier of the Persistent store parameter to be written
p_value	Pointer to a buffer to copy the value
length	Length of the paramter in words

Returns

CSRmeshResult

6.40 CSRmeshPsDeInit Function Documentation

Syntax

```
void CSRmeshPsDeInit (void )
```

Description

Returns

CSRmeshResult

6.41 CSRmeshPsReadSecureKey Function Documentation

Syntax

```
CsrUint16* CSRmeshPsReadSecureKey (void )
```

Description

Application uses this API to generate MeshId

Parameters

Parameter	Description
netId	Network id of the network for which MeshId generation is required.
eventData	On successful completion a secret MeshId be available to the app. as a field in this 'result parameter'

Returns

CSRmeshResult

6.42 CSRSchedSetScanDutyCycle Function Documentation

Syntax

```
CSRSchedResult CSRSchedSetScanDutyCycle (CsrUint16 scan_duty_cycle,
CsrUint16 new_scan_slot)
```

Description

Select a pair scan_window, scan_interval, which are a reasonably small multiple of 625us, matching the percentage requested.

Parameters

Parameter	Description
scan_duty_cycle	Range 1-1000. 1 corresponds to 0.1% and 1000 corresponds to 100%
new_scan_slot	Range 4 to 16384.

Returns

CSRSchedResult. Refer to CSRSchedResult.

6.43 CSRSchedGetScanDutyCycle Function Documentation

Syntax

```
CsrUint16 CSRSchedGetScanDutyCycle (void )
```

Description

Get scan duty cycle from scan window and scan interval.

Returns

CsrUint16.

6.44 CSRSchedSetConfigParams Function Documentation

Syntax

```
CSRSchedResult CSRSchedSetConfigParams (CSR_SCHED_LE_PARAMS_T * le_params)
```

Description

Applicaton uses this API to Configure MESH and Generic LE parameters

Parameters

Parameter	Description
le_params	This structure contains and Mesh or Non-mesh scheduling parameters.

Returns

CSRSchedResult. Refer to CSRSchedResult.

6.45 CSRSchedSendUserAdv Function Documentation

Syntax

```
CSRSchedResult CSRSchedSendUserAdv (CSR_SCHED_ADV_DATA_T * le_adv_data,
CSR_SCHED_USER_ADV_NOTIFY_CB_T callBack)
```

Description

Application Uses this API to transmit user advertising data(CONNECTABLE/NON-CONNECTABLE)

Parameters

Parameter	Description
le_adv_data	contains advertising paramters advertising data and scan response data.
callBack	callback function registered by the application. Scheduler uses this callback to notify the application regarding the successful transmission of user adv packet.

Returns

CSRSchedResult. Refer to CSRSchedResult.

6.46 CSRSchedGetConfigParams Function Documentation

Syntax

```
CSRSchedResult CSRSchedGetConfigParams (CSR_SCHED_LE_PARAMS_T * le_params)
```

Description

Application uses this API to Get the current configuration .

Parameters

Parameter	Description
le_params	This structure contains Generic LE and Mesh LE parameters.

Returns

CSRSchedResult. Refer to CSRSchedResult.

6.47 CSRSchedRegisterPriorityMsgCb Function Documentation

Syntax

```
void CSRSchedRegisterPriorityMsgCb (CSR_SCHED_PRIORITY_MSG_CB_T cb_ptr)
```

Description

Parameters

Parameter	Description
cb_ptr	Function pointer for callback function

Returns

none

6.48 CSRSchedSendPriorityMsg Function Documentation

Syntax

```
CSRSchedResult CSRSchedSendPriorityMsg (CsrUint8 mesh_packet_type, CsrUint8
* payload, CsrUint8 length)
```

Description

This function sends the message with priority. The message bypasses the transmit queue and will be advertised immediately. In case the radio busy in a connection event error code CSR_SCHED_RESULT_RADIO_BUSY is returned.

Parameters

Parameter	Description
mesh_packet_type	
payload	CSRmesh message. The message will be sent as a mesh advert.
length	Message length

Returns

CSRSchedResult. Refer to CSRSchedResult.

6.49 CSRSchedHandleIncomingData Function Documentation

Syntax

```
CSRSchedResult CSRSchedHandleIncomingData
(CSR_SCHED_INCOMING_DATA_EVENT_T data_event, CsrUint8 * data, CsrUint8 length,
CsrInt8 rssi)
```

Description

This API should get called during the reception of incoming mesh message

Parameters

Parameter	Description
data_event	Specifies the bearer type over which the mesh data is received.
data	Mesh data received.
length	Length of the mesh data.
rssi	Received signal strength.

Returns

CSRSchedResult. Refer to CSRSchedResult.

6.50 CSRSchedEnableListening Function Documentation

Syntax

```
CSRSchedResult CSRSchedEnableListening (CsrBool enable)
```

Description

Application uses this api to start/stop LE scan operation.

Parameters

Parameter	Description
enable	TRUE / FALSE

Returns

CSRSchedResult. Refer to CSRSchedResult.

6.51 IsCSRSchedRunning Function Documentation

Syntax

```
CsrBool IsCSRSchedRunning (void )
```

Description

Application uses this api to check whether scheduler is ready for sending advtertisements.

Returns

CsrBool.

6.52 CSRSchedStart Function Documentation

Syntax

```
CSRSchedResult CSRSchedStart (void )
```

Description

This API starts scheduling of LE scan and advertisement

Returns

CSRSchedResult. Refer to CSRSchedResult.

6.53 CSRSchedNotifyGattEvent Function Documentation

Syntax

```
void CSRSchedNotifyGattEvent (CSR_SCHED_GATT_EVENT_T gatt_event_type,
CSR_SCHED_GATT_EVENT_DATA_T * gatt_event_data,
CSR_SCHED_NOTIFY_GATT_CB_T call_back)
```

Description

Application uses this API to notify the scheduler regarding the GATT connection state event

Parameters

Parameter	Description
gatt_event_type	GATT event type
gatt_event_data	GATT event data
call_back	callback function to notify mesh data to client

Returns

Nothing

6.54 CsrSchedSetTxPower Function Documentation

Syntax

```
void CsrSchedSetTxPower (CsrUint8 level)
```

Description

Sets the Transmit power for the device. This internally uses LsSetTransmitPowerLevel API without any extra mappings.

```
#if defined(CSR101x) || defined(CSR101x_A05)

// Tx Power level mapping.
// 0 :-18 dBm 1 :-14 dBm 2 :-10 dBm 3 :-06 dBm
// 4 :-02 dBm 5 :+02 dBm 6 :+06 dBm 7 :+08 dBm

#define CSR_MESH_MIN_TX_POWER_LEVEL   (-18)
#define CSR_MESH_MAX_TX_POWER_LEVEL   (8)
#define CSR_MESH_TX_POWER_LEVEL_STEP  (4)

#else

// Tx Power level mapping.
// #define TX_PA_POWER_DBM_0        (-60)
// #define TX_PA_POWER_DBM_1        (-30)
// #define TX_PA_POWER_DBM_2        (-22)
// #define TX_PA_POWER_DBM_3        (-16)
// #define TX_PA_POWER_DBM_4        (-13)
// #define TX_PA_POWER_DBM_5        (-10)
// #define TX_PA_POWER_DBM_6        (-6)
// #define TX_PA_POWER_DBM_7        (-4)
// #define TX_PA_POWER_DBM_8        (-3)
// #define TX_PA_POWER_DBM_9        (-2)
// #define TX_PA_POWER_DBM_10       (-1)
// #define TX_PA_POWER_DBM_11       (0)
// #define TX_PA_POWER_DBM_12       (1)
// #define TX_PA_POWER_DBM_13       (2)
// #define TX_PA_POWER_DBM_14       (3)
// #define TX_PA_POWER_DBM_15       (4)
```

```
30
31 static const CsrInt8 fh_tx_pwr_map[] = { -60, -30, -22, -16, -13, -10,
-6, -4, -3, -2,  -1, 0, 1, 2, 3, 4 };
32
33 #define CSR_MESH_MIN_TX_POWER_LEVEL   (-60)
34 #define CSR_MESH_MAX_TX_POWER_LEVEL   (4)
35 #endif
36

37 //------------------------------------------------------------------------
-
38 //  CsrSchedConvertTxPower
39 //
40 //  DESCRIPTION
41 //      Sets the Transmit Power to the nearest value possible.
42 //
43 //  PARAMETERS
44 //      power Transmit power level in dBm.
45 //
46 //  RETURNS/MODIFIES
47 //      Returns level as per the platform.

48 //------------------------------------------------------------------------
-
49 extern CsrUint8 CsrSchedConvertTxPower(CsrInt8 power)
50 {
51     CsrUint8 level = 0;
52
53     if (power & 0x80)
54     {
55           power |= 0xFF00;
56     }
57
58     // Map Power level in dBm to index
59     if (power <= CSR_MESH_MIN_TX_POWER_LEVEL)
60     {
61           level = LS_MIN_TRANSMIT_POWER_LEVEL;
62     }
63     else if (power >= CSR_MESH_MAX_TX_POWER_LEVEL)
64     {
65           level = LS_MAX_TRANSMIT_POWER_LEVEL;
66     }
67     else
68     {
```

```
   70            for (level = 0; level < (sizeof(fh_tx_pwr_map) - 1); level++)
   71            {
   72                 if ((power >= fh_tx_pwr_map[level]) && (power <
fh_tx_pwr_map[level + 1]))
   73                 {
   74                       break;
   75                 }
   76            }
   77 #else
   78            level = (power - CSR_MESH_MIN_TX_POWER_LEVEL) /
CSR_MESH_TX_POWER_LEVEL_STEP;
   79 #endif
   80     }
   81
   82     return level;
   83 }
```

Parameters

Parameter	Description
`level`	The level of the transmit power to be used. It is not a level in dBm, it's an index value into the transmit power table. Refer to LsSetTransmitPowerLevel in CSR µEnergy® Firmware Library Documentation. An example mapping function from dBm to level is shown below for CSR101x/CSR102x.

Returns

Nothing

7 Models

Detailed Description

7.1 Modules of Models

Watchdog Model

Watchdog Model API.

Config Model

Config Model API.

Group Model

Group Model API.

Sensor Model

Sensor Model API.

Actuator Model

Actuator Model API.

Data Model

Data Model API.

Bearer Model

Bearer Model API.

Ping Model

Ping Model API.

Battery Model

Battery Model API.

Attention Model

Attention Model API.

Power Model

Power Model API.

Light Model

Light Model API.

Asset Model

Asset Model API.

Tracker Model

Tracker Model API.

Time Model

Time Model API.

Switch Model

Switch Model API.

Tuning Model

Tuning Model API.

Extension Model

Extension Model API.

LargeObjectTransfer Model

LargeObjectTransfer Model API.

Firmware Model

Firmware Model API.

Diagnostic Model

Diagnostic Model API.

Action Model

Action Model API.

Beacon Model

Beacon Model API.

BeaconProxy Model

BeaconProxy Model API.

7.2 Data Structures of Models

```
struct  CSRMESH_EVENT_DATA_T
```

CSRmesh Event Data.

7.3 Typedefs of Models

```
typedef CSRmeshResult(* CSRMESH_MODEL_CALLBACK_T) (CSRMESH_MODEL_EVENT_T
event_code, CSRMESH_EVENT_DATA_T *data, CsrUint16 length, void **state_data)
```

MCP Model handler function type.

7.4 Enumerations of Models

```
enum  csr_mesh_boolean_t { csr_mesh_boolean_false = 0,
csr_mesh_boolean_true = 1
}
```

CSRmesh boolean type.

```
enum  csr_mesh_power_state_t { csr_mesh_power_state_off = 0,
csr_mesh_power_state_on = 1,
csr_mesh_power_state_standby = 2,
csr_mesh_power_state_onfromstandby = 3
}
```

CSRmesh power_state type.

```
enum  csr_mesh_device_information_t {
  csr_mesh_device_information_uuid_low = 0,
csr_mesh_device_information_uuid_high = 1,
csr_mesh_device_information_model_low = 2,
csr_mesh_device_information_model_high = 3,
  csr_mesh_device_information_vid_pid_version = 4,
csr_mesh_device_information_appearance = 5,
csr_mesh_device_information_lastetag = 6
}
```

CSRmesh device_information type.

```
enum  csr_mesh_key_properties_t { csr_mesh_key_properties_administrator = 0,
csr_mesh_key_properties_user = 1,
csr_mesh_key_properties_guest = 2,
csr_mesh_key_properties_relayonly = 3
}
```

CSRmesh key_properties type.

```
enum  csr_mesh_month_of_year_t {
  csr_mesh_month_of_year_unknown = 0,
csr_mesh_month_of_year_january = 1,
csr_mesh_month_of_year_february = 2,
csr_mesh_month_of_year_march = 3,
  csr_mesh_month_of_year_april = 4,
csr_mesh_month_of_year_may = 5,
csr_mesh_month_of_year_june = 6,
csr_mesh_month_of_year_july = 7,
  csr_mesh_month_of_year_august = 8,
csr_mesh_month_of_year_september = 9,
csr_mesh_month_of_year_october = 10,
csr_mesh_month_of_year_november = 11,
  csr_mesh_month_of_year_december = 12
}
```

CSRmesh month_of_year type.

```
enum  csr_mesh_timer_mode_t { csr_mesh_timer_mode_programming = 0,
csr_mesh_timer_mode_active = 1,
csr_mesh_timer_mode_partly_random = 2,
csr_mesh_timer_mode_completely_random = 3
}
```

CSRmesh timer_mode type.

```
enum  csr_mesh_remote_code_t {
  csr_mesh_remote_code_number_0 = 0,
csr_mesh_remote_code_number_1 = 1,
csr_mesh_remote_code_number_2 = 2,
csr_mesh_remote_code_number_3 = 3,
  csr_mesh_remote_code_number_4 = 4,
csr_mesh_remote_code_number_5 = 5,
csr_mesh_remote_code_number_6 = 6,
csr_mesh_remote_code_number_7 = 7,
  csr_mesh_remote_code_number_8 = 8,
csr_mesh_remote_code_number_9 = 9,
csr_mesh_remote_code_direction_n = 16,
csr_mesh_remote_code_direction_e = 17,
  csr_mesh_remote_code_direction_s = 18,
csr_mesh_remote_code_direction_w = 19,
csr_mesh_remote_code_direction_ne = 20,
csr_mesh_remote_code_direction_se = 21,
  csr_mesh_remote_code_direction_sw = 22,
csr_mesh_remote_code_direction_nw = 23,
csr_mesh_remote_code_direction_nne = 24,
csr_mesh_remote_code_direction_ene = 25,
  csr_mesh_remote_code_direction_ese = 26,
csr_mesh_remote_code_direction_sse = 27,
csr_mesh_remote_code_direction_ssw = 28,
csr_mesh_remote_code_direction_wsw = 29,
  csr_mesh_remote_code_direction_wnw = 30,
csr_mesh_remote_code_direction_nnw = 31,
csr_mesh_remote_code_select = 32,
csr_mesh_remote_code_channel_up = 33,
  csr_mesh_remote_code_channel_down = 34,
csr_mesh_remote_code_volume_up = 35,
csr_mesh_remote_code_volume_down = 36,
csr_mesh_remote_code_volume_mute = 37,
  csr_mesh_remote_code_menu = 38,
csr_mesh_remote_code_back = 39,
csr_mesh_remote_code_guide = 40,
csr_mesh_remote_code_play = 41,
  csr_mesh_remote_code_pause = 42,
csr_mesh_remote_code_stop = 43,
csr_mesh_remote_code_fast_forward = 44,
csr_mesh_remote_code_fast_rewind = 45,
  csr_mesh_remote_code_skip_forward = 46,
csr_mesh_remote_code_skip_backward = 47
}
```

CSRmesh remote_code type.

```
enum  csr_mesh_sensor_type_t {
  csr_mesh_sensor_type_unknown = 0,
csr_mesh_sensor_type_internal_air_temperature = 1,
csr_mesh_sensor_type_external_air_temperature = 2,
csr_mesh_sensor_type_desired_air_temperature = 3,
  csr_mesh_sensor_type_internal_humidity = 4,
csr_mesh_sensor_type_external_humidity = 5,
csr_mesh_sensor_type_external_dewpoint = 6,
csr_mesh_sensor_type_internal_door = 7,
  csr_mesh_sensor_type_external_door = 8,
csr_mesh_sensor_type_internal_window = 9,
csr_mesh_sensor_type_external_window = 10,
csr_mesh_sensor_type_solar_energy = 11,
  csr_mesh_sensor_type_number_of_activations = 12,
csr_mesh_sensor_type_fridge_temperature = 13,
csr_mesh_sensor_type_desired_fridge_temperature = 14,
csr_mesh_sensor_type_freezer_temperature = 15,
  csr_mesh_sensor_type_desired_freezer_temperature = 16,
csr_mesh_sensor_type_oven_temperature = 17,
csr_mesh_sensor_type_desired_oven_temperature = 18,
csr_mesh_sensor_type_seat_occupied = 19,
  csr_mesh_sensor_type_washing_machine_state = 20,
csr_mesh_sensor_type_dish_washer_state = 21,
csr_mesh_sensor_type_clothes_dryer_state = 22,
csr_mesh_sensor_type_toaster_state = 23,
  csr_mesh_sensor_type_carbon_dioxide = 24,
csr_mesh_sensor_type_carbon_monoxide = 25,
csr_mesh_sensor_type_smoke = 26,
csr_mesh_sensor_type_water_level = 27,
  csr_mesh_sensor_type_hot_water_temperature = 28,
csr_mesh_sensor_type_cold_water_temperature = 29,
csr_mesh_sensor_type_desired_water_temperature = 30,
csr_mesh_sensor_type_cooker_hob_back_left_state = 31,
  csr_mesh_sensor_type_desired_cooker_hob_back_left_state = 32,
csr_mesh_sensor_type_cooker_hob_front_left_state = 33,
csr_mesh_sensor_type_desired_cooker_hob_front_left_state = 34,
csr_mesh_sensor_type_cooker_hob_back_middle_state = 35,
  csr_mesh_sensor_type_desired_cooker_hob_back_middle_state = 36,
csr_mesh_sensor_type_cooker_hob_front_middle_state = 37,
csr_mesh_sensor_type_desired_cooker_hob_front_middle_state = 38,
csr_mesh_sensor_type_cooker_hob_back_right_state = 39,
  csr_mesh_sensor_type_desired_cooker_hob_back_right_state = 40,
csr_mesh_sensor_type_cooker_hob_front_right_state = 41,
csr_mesh_sensor_type_desired_cooker_hob_front_right_state = 42,
csr_mesh_sensor_type_desired_wakeup_alarm_time = 43,
  csr_mesh_sensor_type_desired_second_wakeup_alarm_time = 44,
csr_mesh_sensor_type_passive_infrared_state = 45,
csr_mesh_sensor_type_water_flowing = 46,
csr_mesh_sensor_type_desired_water_flow = 47,
  csr_mesh_sensor_type_audio_level = 48,
csr_mesh_sensor_type_desired_audio_level = 49,
csr_mesh_sensor_type_fan_speed = 50,
csr_mesh_sensor_type_desired_fan_speed = 51,
```

```
  csr_mesh_sensor_type_wind_speed = 52,
csr_mesh_sensor_type_wind_speed_gust = 53,
csr_mesh_sensor_type_wind_direction = 54,
csr_mesh_sensor_type_wind_direction_gust = 55,
  csr_mesh_sensor_type_rain_fall_last_hour = 56,
csr_mesh_sensor_type_rain_fall_today = 57,
csr_mesh_sensor_type_barometric_pressure = 58,
csr_mesh_sensor_type_soil_temperature = 59,
  csr_mesh_sensor_type_soil_moisure = 60,
csr_mesh_sensor_type_window_cover_position = 61,
csr_mesh_sensor_type_desired_window_cover_position = 62,
csr_mesh_sensor_type_generic_1_byte = 63,
  csr_mesh_sensor_type_generic_2_byte = 64,
csr_mesh_sensor_type_generic_1_byte_typed = 250,
csr_mesh_sensor_type_generic_2_byte_typed = 251,
csr_mesh_sensor_type_generic_3_byte_typed = 252
}
```

CSRmesh sensor_type type.

```
enum  csr_mesh_beacon_type_t {
  csr_mesh_beacon_type_csr = 0,
csr_mesh_beacon_type_ibeacon = 1,
csr_mesh_beacon_type_eddystone_url = 2,
csr_mesh_beacon_type_eddystone_uid = 3,
  csr_mesh_beacon_type_lte_direct = 4,
csr_mesh_beacon_type_lumicast = 5
}
```

CSRmesh beacon_type type.

```
enum  csr_mesh_door_state_t { csr_mesh_door_state_open_unlocked = 2,
csr_mesh_door_state_closed_unlocked = 1,
csr_mesh_door_state_closed_locked = 0
}
```

CSRmesh door_state type.

```
enum  csr_mesh_window_state_t { csr_mesh_window_state_closed_insecure = 1,
csr_mesh_window_state_open_insecure = 3,
csr_mesh_window_state_open_secure = 2,
csr_mesh_window_state_closed_secure = 0
}
```

CSRmesh window_state type.

```
enum  csr_mesh_seat_state_t { csr_mesh_seat_state_empty = 0,
csr_mesh_seat_state_continuously_occupied = 2,
csr_mesh_seat_state_occupied = 1
}
```

CSRmesh seat_state type.

```
enum  csr_mesh_appliance_state_t {
  csr_mesh_appliance_state_pre_wash = 3,
csr_mesh_appliance_state_idle = 0,
csr_mesh_appliance_state_post_wash = 5,
csr_mesh_appliance_state_washing = 4,
  csr_mesh_appliance_state_heating_up = 1,
csr_mesh_appliance_state_cooling_down = 2
}
```

CSRmesh appliance_state type.

```
enum  csr_mesh_cooker_hob_state_t {
  csr_mesh_cooker_hob_state_level_2 = 2,
csr_mesh_cooker_hob_state_off_hot = 10,
csr_mesh_cooker_hob_state_level_1 = 1,
csr_mesh_cooker_hob_state_level_7 = 7,
  csr_mesh_cooker_hob_state_level_9 = 9,
csr_mesh_cooker_hob_state_off_cold = 0,
csr_mesh_cooker_hob_state_level_8 = 8,
csr_mesh_cooker_hob_state_level_5 = 5,
  csr_mesh_cooker_hob_state_level_4 = 4,
csr_mesh_cooker_hob_state_level_3 = 3,
csr_mesh_cooker_hob_state_level_6 = 6
}
```

CSRmesh cooker_hob_state type.

```
enum  csr_mesh_movement_state_t { csr_mesh_movement_state_no_movement = 0,
csr_mesh_movement_state_movement_detected = 1
}
```

CSRmesh movement_state type.

```
enum  csr_mesh_forward_backward_t { csr_mesh_forward_backward_forward = 1,
csr_mesh_forward_backward_backwards = 2
}
```

CSRmesh forward_backward type.

```
enum  csr_mesh_direction_t {
  csr_mesh_direction_north_of_north_east = 16,
csr_mesh_direction_north = 0,
csr_mesh_direction_north_east = 32,
csr_mesh_direction_east = 64,
  csr_mesh_direction_north_west = 224,
csr_mesh_direction_north_of_north_west = 240,
csr_mesh_direction_east_of_north_east = 48,
csr_mesh_direction_west = 192,
  csr_mesh_direction_south_east = 96,
csr_mesh_direction_east_of_south_east = 80,
csr_mesh_direction_south_of_south_east = 112,
csr_mesh_direction_south = 128,
  csr_mesh_direction_west_of_north_west = 208,
csr_mesh_direction_south_of_south_west = 144,
csr_mesh_direction_west_of_south_west = 176,
csr_mesh_direction_south_west = 160
}
```

CSRmesh direction type.

```
enum  CSRMESH_MODEL_TYPE_T {
  CSRMESH_WATCHDOG_MODEL = 0,
CSRMESH_CONFIG_MODEL = 1,
CSRMESH_GROUP_MODEL = 2,
CSRMESH_SENSOR_MODEL = 4,
  CSRMESH_ACTUATOR_MODEL = 5,
CSRMESH_DATA_MODEL = 8,
CSRMESH_BEARER_MODEL = 11,
CSRMESH_PING_MODEL = 12,
  CSRMESH_BATTERY_MODEL = 13,
CSRMESH_ATTENTION_MODEL = 14,
CSRMESH_POWER_MODEL = 19,
CSRMESH_LIGHT_MODEL = 20,
  CSRMESH_ASSET_MODEL = 6,
CSRMESH_TRACKER_MODEL = 7,
CSRMESH_TIME_MODEL = 16,
CSRMESH_SWITCH_MODEL = 21,
  CSRMESH_TUNING_MODEL = 28,
CSRMESH_EXTENSION_MODEL = 29,
CSRMESH_LARGEOBJECTTRANSFER_MODEL = 30,
CSRMESH_FIRMWARE_MODEL = 9,
  CSRMESH_DIAGNOSTIC_MODEL = 10,
CSRMESH_ACTION_MODEL = 34,
CSRMESH_BEACON_MODEL = 32,
CSRMESH_BEACONPROXY_MODEL = 33,
  CSRMESH_ALL_MODELS = 255
}
```

CSRmesh Model types.

```
enum  CSRMESH_MODEL_EVENT_T {
  CSRMESH_WATCHDOG_MESSAGE = 0x00,
CSRMESH_WATCHDOG_SET_INTERVAL = 0x01,
CSRMESH_WATCHDOG_INTERVAL = 0x02,
CSRMESH_CONFIG_LAST_SEQUENCE_NUMBER = 0x03,
  CSRMESH_CONFIG_RESET_DEVICE = 0x04,
CSRMESH_CONFIG_SET_DEVICE_IDENTIFIER = 0x05,
CSRMESH_CONFIG_SET_PARAMETERS = 0x06,
CSRMESH_CONFIG_GET_PARAMETERS = 0x07,
  CSRMESH_CONFIG_PARAMETERS = 0x08,
CSRMESH_CONFIG_DISCOVER_DEVICE = 0x09,
CSRMESH_CONFIG_DEVICE_IDENTIFIER = 0x0A,
CSRMESH_CONFIG_GET_INFO = 0x0B,
  CSRMESH_CONFIG_INFO = 0x0C
, CSRMESH_GROUP_GET_NUMBER_OF_MODEL_GROUPIDS = 0x0D,
CSRMESH_GROUP_NUMBER_OF_MODEL_GROUPIDS = 0x0E,
CSRMESH_GROUP_SET_MODEL_GROUPID = 0x0F,
  CSRMESH_GROUP_GET_MODEL_GROUPID = 0x10,
CSRMESH_GROUP_MODEL_GROUPID = 0x11,
CSRMESH_SENSOR_GET_TYPES = 0x20,
CSRMESH_SENSOR_TYPES = 0x21,
  CSRMESH_SENSOR_SET_STATE = 0x22,
CSRMESH_SENSOR_GET_STATE = 0x23,
CSRMESH_SENSOR_STATE = 0x24,
CSRMESH_SENSOR_WRITE_VALUE_NO_ACK = 0x26,
  CSRMESH_SENSOR_WRITE_VALUE = 0x25,
CSRMESH_SENSOR_READ_VALUE = 0x27,
CSRMESH_SENSOR_VALUE = 0x28,
CSRMESH_SENSOR_MISSING = 0x29,
  CSRMESH_ACTUATOR_GET_TYPES = 0x30,
CSRMESH_ACTUATOR_TYPES = 0x31,
CSRMESH_ACTUATOR_SET_VALUE_NO_ACK = 0x32,
CSRMESH_ACTUATOR_SET_VALUE = 0x33,
  CSRMESH_ACTUATOR_VALUE_ACK = 0x35,
CSRMESH_ACTUATOR_GET_VALUE_ACK = 0x34,
CSRMESH_DATA_STREAM_FLUSH = 0x70,
CSRMESH_DATA_STREAM_SEND = 0x71,
  CSRMESH_DATA_STREAM_RECEIVED = 0x72,
CSRMESH_DATA_BLOCK_SEND = 0x73,
CSRMESH_BEARER_SET_STATE = 0x8100,
CSRMESH_BEARER_GET_STATE = 0x8101,
  CSRMESH_BEARER_STATE = 0x8102,
CSRMESH_PING_REQUEST = 0x8200,
CSRMESH_PING_RESPONSE = 0x8201,
CSRMESH_BATTERY_GET_STATE = 0x8300,
  CSRMESH_BATTERY_STATE = 0x8301,
CSRMESH_ATTENTION_SET_STATE = 0x8400,
CSRMESH_ATTENTION_STATE = 0x8401,
CSRMESH_POWER_SET_STATE_NO_ACK = 0x8900,
  CSRMESH_POWER_SET_STATE = 0x8901,
CSRMESH_POWER_TOGGLE_STATE_NO_ACK = 0x8902,
CSRMESH_POWER_TOGGLE_STATE = 0x8903,
CSRMESH_POWER_GET_STATE = 0x8904,
```

```
  CSRMESH_POWER_STATE_NO_ACK = 0x8906,
CSRMESH_POWER_STATE = 0x8905,
CSRMESH_LIGHT_SET_LEVEL_NO_ACK = 0x8A00,
CSRMESH_LIGHT_SET_LEVEL = 0x8A01,
  CSRMESH_LIGHT_SET_RGB_NO_ACK = 0x8A02,
CSRMESH_LIGHT_SET_RGB = 0x8A03,
CSRMESH_LIGHT_SET_POWER_LEVEL_NO_ACK = 0x8A04,
CSRMESH_LIGHT_SET_POWER_LEVEL = 0x8A05,
  CSRMESH_LIGHT_SET_COLOR_TEMP = 0x8A06,
CSRMESH_LIGHT_GET_STATE = 0x8A07,
CSRMESH_LIGHT_STATE_NO_ACK = 0x8A09,
CSRMESH_LIGHT_STATE = 0x8A08,
  CSRMESH_LIGHT_SET_WHITE_NO_ACK = 0x8A0B,
CSRMESH_LIGHT_SET_WHITE = 0x8A0A,
CSRMESH_LIGHT_GET_WHITE = 0x8A0C,
CSRMESH_LIGHT_WHITE_NO_ACK = 0x8A0E,
  CSRMESH_LIGHT_WHITE = 0x8A0D,
CSRMESH_ASSET_SET_STATE = 0x40,
```

```
CSRMESH_ASSET_GET_STATE = 0x41,
CSRMESH_ASSET_STATE = 0x42,
  CSRMESH_ASSET_ANNOUNCE = 0x43,
CSRMESH_TRACKER_FIND = 0x44,
CSRMESH_TRACKER_FOUND = 0x45,
CSRMESH_TRACKER_REPORT = 0x46,
  CSRMESH_TRACKER_CLEAR_CACHE = 0x47,
CSRMESH_TRACKER_SET_PROXIMITY_CONFIG = 0x48,
CSRMESH_TIME_SET_STATE = 0x75,
CSRMESH_TIME_GET_STATE = 0x76,
  CSRMESH_TIME_STATE = 0x77,
CSRMESH_TIME_BROADCAST = 0x7F,
CSRMESH_TUNING_PROBE = 0xEF01,
CSRMESH_TUNING_GET_STATS = 0xEF02,
  CSRMESH_TUNING_STATS = 0xEF03,
CSRMESH_TUNING_ACK_CONFIG = 0xEF04,
CSRMESH_TUNING_SET_CONFIG = 0xEF05,
CSRMESH_EXTENSION_REQUEST = 0xEF10,
  CSRMESH_EXTENSION_CONFLICT = 0xEF11,
CSRMESH_LARGEOBJECTTRANSFER_ANNOUNCE = 0x1A,
CSRMESH_LARGEOBJECTTRANSFER_INTEREST = 0xEF30,
CSRMESH_FIRMWARE_GET_VERSION = 0x78,
  CSRMESH_FIRMWARE_VERSION = 0x79,
CSRMESH_FIRMWARE_UPDATE_REQUIRED = 0x7A,
CSRMESH_FIRMWARE_UPDATE_ACKNOWLEDGED = 0x7b,
CSRMESH_DIAGNOSTIC_STATE = 0xEF40
,
  CSRMESH_ACTION_SET_ACTION = 0x50,
CSRMESH_ACTION_SET_ACTION_ACK = 0x51,
CSRMESH_ACTION_GET_ACTION_STATUS = 0x52,
CSRMESH_ACTION_ACTION_STATUS = 0x53,
  CSRMESH_ACTION_DELETE = 0x54,
CSRMESH_ACTION_DELETE_ACK = 0x55,
CSRMESH_ACTION_GET = 0x56,
CSRMESH_BEACON_SET_STATUS = 0x60,
  CSRMESH_BEACON_GET_BEACON_STATUS = 0x62,
CSRMESH_BEACON_BEACON_STATUS = 0x63,
CSRMESH_BEACON_GET_TYPES = 0x64,
CSRMESH_BEACON_TYPES = 0x65,
  CSRMESH_BEACON_SET_PAYLOAD = 0x66,
CSRMESH_BEACON_PAYLOAD_ACK = 0x67,
CSRMESH_BEACON_GET_PAYLOAD = 0x68,
CSRMESH_BEACONPROXY_ADD = 0x69,
  CSRMESH_BEACONPROXY_REMOVE = 0x6A,
CSRMESH_BEACONPROXY_COMMAND_STATUS_DEVICES = 0x6B,
CSRMESH_BEACONPROXY_GET_STATUS = 0x6C,
CSRMESH_BEACONPROXY_PROXY_STATUS = 0x6D,
  CSRMESH_BEACONPROXY_GET_DEVICES = 0x6E,
CSRMESH_BEACONPROXY_DEVICES = 0x6F
}
```

CSRmesh Model Event Code types.

7.5 csr_mesh_boolean_t Enumeration Type Documentation

Syntax

```
enum csr_mesh_boolean_t
```

Description

Enumerations

Enumeration	Description
csr_mesh_boolean_false	false
csr_mesh_boolean_true	true

7.6 csr_mesh_power_state_t Enumeration Type Documentation

Syntax

```
enum csr_mesh_power_state_t
```

Description

Enumerations

Enumeration	Description
csr_mesh_power_state_off	
csr_mesh_power_state_on	
csr_mesh_power_state_standby	standby
csr_mesh_power_state_onfromstandby	onfromstandby

7.7 csr_mesh_device_information_t Enumeration Type Documentation

Syntax

```
enum csr_mesh_device_information_t
```

Description

Enumerations

Enumeration	Description
csr_mesh_device_information_uuid_low	uuid_low
csr_mesh_device_information_uuid_high	uuid_high
csr_mesh_device_information_model_low	model_low
csr_mesh_device_information_model_high	model_high
csr_mesh_device_information_vid_pid_version	vid_pid_version
csr_mesh_device_information_appearance	appearance
csr_mesh_device_information_lastetag	lastetag

7.8 csr_mesh_key_properties_t Enumeration Type Documentation

Syntax

```
enum csr_mesh_key_properties_t
```

Description

Enumerations

Enumeration	Description
csr_mesh_key_properties_administrator	administrator
csr_mesh_key_properties_user	user
csr_mesh_key_properties_guest	guest
csr_mesh_key_properties_relayonly	relayonly

7.9 csr_mesh_month_of_year_t Enumeration Type Documentation

Syntax

```
enum csr_mesh_month_of_year_t
```

Description

Enumerations

Enumeration	Description
csr_mesh_month_of_year_unknown	unknown
csr_mesh_month_of_year_january	january
csr_mesh_month_of_year_february	february
csr_mesh_month_of_year_march	march
csr_mesh_month_of_year_april	april
csr_mesh_month_of_year_may	may
csr_mesh_month_of_year_june	june
csr_mesh_month_of_year_july	july

Enumeration	Description
csr_mesh_month_of_year_august	august
csr_mesh_month_of_year_septembe r	september
csr_mesh_month_of_year_october	october
csr_mesh_month_of_year_november	november
csr_mesh_month_of_year_december	december

7.10 csr_mesh_timer_mode_t Enumeration Type Documentation

Syntax

```
enum csr_mesh_timer_mode_t
```

Description

Enumerations

Enumeration	Description
csr_mesh_timer_mode_programming	programming
csr_mesh_timer_mode_active	active
csr_mesh_timer_mode_partly_rand om	partly_random
csr_mesh_timer_mode_completely_ random	completely_random

7.11 csr_mesh_remote_code_t Enumeration Type Documentation

Syntax

```
enum csr_mesh_remote_code_t
```

Description

Enumerations

Enumeration	Description
csr_mesh_remote_code_number_0	number_0
csr_mesh_remote_code_number_1	number_1
csr_mesh_remote_code_number_2	number_2
csr_mesh_remote_code_number_3	number_3
csr_mesh_remote_code_number_4	number_4
csr_mesh_remote_code_number_5	number_5
csr_mesh_remote_code_number_6	number_6
csr_mesh_remote_code_number_7	number_7
csr_mesh_remote_code_number_8	number_8
csr_mesh_remote_code_number_9	number_9
csr_mesh_remote_code_direction_n	direction_n
csr_mesh_remote_code_direction_e	direction_e
csr_mesh_remote_code_direction_s	direction_s
csr_mesh_remote_code_direction_w	direction_w

Enumeration	Description
csr_mesh_remote_code_direction_ne	direction_ne
csr_mesh_remote_code_direction_se	direction_se
csr_mesh_remote_code_direction_sw	direction_sw
csr_mesh_remote_code_direction_nw	direction_nw
csr_mesh_remote_code_direction_nne	direction_nne
csr_mesh_remote_code_direction_ene	direction_ene
csr_mesh_remote_code_direction_ese	direction_ese
csr_mesh_remote_code_direction_sse	direction_sse
csr_mesh_remote_code_direction_ssw	direction_ssw
csr_mesh_remote_code_direction_wsw	direction_wsw
csr_mesh_remote_code_direction_wnw	direction_wnw
csr_mesh_remote_code_direction_nnw	direction_nnw
csr_mesh_remote_code_select	select
csr_mesh_remote_code_channel_up	channel_up
csr_mesh_remote_code_channel_down	channel_down
csr_mesh_remote_code_volume_up	volume_up

Enumeration	Description
csr_mesh_remote_code_volume_down	volume_down
csr_mesh_remote_code_volume_mute	volume_mute
csr_mesh_remote_code_menu	menu
csr_mesh_remote_code_back	back
csr_mesh_remote_code_guide	guide
csr_mesh_remote_code_play	play
csr_mesh_remote_code_pause	pause
csr_mesh_remote_code_stop	stop
csr_mesh_remote_code_fast_forward	fast_forward
csr_mesh_remote_code_fast_rewind	fast_rewind
csr_mesh_remote_code_skip_forward	skip_forward
csr_mesh_remote_code_skip_backward	skip_backward

7.12 csr_mesh_sensor_type_t Enumeration Type Documentation

Syntax

```
enum csr_mesh_sensor_type_t
```

Description

Enumerations

Enumeration	Description
csr_mesh_sensor_type_unknown	unknown with values in the format csr_mesh__t
csr_mesh_sensor_type_internal_air_temperature	internal_air_temperature with values in the format csr_mesh_temperature_kelvin_t
csr_mesh_sensor_type_external_air_temperature	external_air_temperature with values in the format csr_mesh_temperature_kelvin_range_t
csr_mesh_sensor_type_desired_air_temperature	desired_air_temperature with values in the format csr_mesh_temperature_kelvin_t
csr_mesh_sensor_type_internal_humidity	internal_humidity with values in the format csr_mesh_percentage_t
csr_mesh_sensor_type_external_humidity	external_humidity with values in the format csr_mesh_percentage_t
csr_mesh_sensor_type_external_dewpoint	external_dewpoint with values in the format csr_mesh_temperature_kelvin_t
csr_mesh_sensor_type_internal_door	internal_door with values in the format csr_mesh_door_state_t
csr_mesh_sensor_type_external_door	external_door with values in the format csr_mesh_door_state_t
csr_mesh_sensor_type_internal_window	internal_window with values in the format csr_mesh_window_state_t
csr_mesh_sensor_type_external_window	external_window with values in the format csr_mesh_window_state_t
csr_mesh_sensor_type_solar_energy	solar_energy with values in the format csr_mesh_watts_per_square_metre_t
csr_mesh_sensor_type_number_of_activations	number_of_activations with values in the format csr_mesh_16_bit_counter_t
csr_mesh_sensor_type_fridge_temperature	fridge_temperature with values in the format csr_mesh_temperature_kelvin_t
csr_mesh_sensor_type_desired_fridge_temperature	desired_fridge_temperature with values in the format csr_mesh_temperature_kelvin_t

Enumeration	Description
csr_mesh_sensor_type_freezer_temperature	freezer_temperature with values in the format csr_mesh_temperature_kelvin_t
csr_mesh_sensor_type_desired_freezer_temperature	desired_freezer_temperature with values in the format csr_mesh_temperature_kelvin_t
csr_mesh_sensor_type_oven_temperature	oven_temperature with values in the format csr_mesh_temperature_kelvin_t
csr_mesh_sensor_type_desired_oven_temperature	desired_oven_temperature with values in the format csr_mesh_temperature_kelvin_t
csr_mesh_sensor_type_seat_occupied	seat_occupied with values in the format csr_mesh_seat_state_t
csr_mesh_sensor_type_washing_machine_state	washing_machine_state with values in the format csr_mesh_appliance_state_t
csr_mesh_sensor_type_dish_washer_state	dish_washer_state with values in the format csr_mesh_appliance_state_t
csr_mesh_sensor_type_clothes_dryer_state	clothes_dryer_state with values in the format csr_mesh_appliance_state_t
csr_mesh_sensor_type_toaster_state	toaster_state with values in the format csr_mesh_appliance_state_t
csr_mesh_sensor_type_carbon_dioxide	carbon_dioxide with values in the format csr_mesh_parts_per_million_t
csr_mesh_sensor_type_carbon_monoxide	carbon_monoxide with values in the format csr_mesh_parts_per_million_t
csr_mesh_sensor_type_smoke	smoke with values in the format csr_mesh_micrograms_per_cubic_metre_t
csr_mesh_sensor_type_water_level	water_level with values in the format csr_mesh_percentage_t
csr_mesh_sensor_type_hot_water_temperature	hot_water_temperature with values in the format csr_mesh_temperature_kelvin_t
csr_mesh_sensor_type_cold_water_temperature	cold_water_temperature with values in the format csr_mesh_temperature_kelvin_t

Enumeration	Description
csr_mesh_sensor_type_desired_water_temperature	desired_water_temperature with values in the format csr_mesh_temperature_kelvin_t
csr_mesh_sensor_type_cooker_hob_back_left_state	cooker_hob_back_left_state with values in the format csr_mesh_cooker_hob_state_t
csr_mesh_sensor_type_desired_cooker_hob_back_left_state	desired_cooker_hob_back_left_state with values in the format csr_mesh_cooker_hob_state_t
csr_mesh_sensor_type_cooker_hob_front_left_state	cooker_hob_front_left_state with values in the format csr_mesh_cooker_hob_state_t
csr_mesh_sensor_type_desired_cooker_hob_front_left_state	desired_cooker_hob_front_left_state with values in the format csr_mesh_cooker_hob_state_t
csr_mesh_sensor_type_cooker_hob_back_middle_state	cooker_hob_back_middle_state with values in the format csr_mesh_cooker_hob_state_t
csr_mesh_sensor_type_desired_cooker_hob_back_middle_state	desired_cooker_hob_back_middle_state with values in the format csr_mesh_cooker_hob_state_t
csr_mesh_sensor_type_cooker_hob_front_middle_state	cooker_hob_front_middle_state with values in the format csr_mesh_cooker_hob_state_t
csr_mesh_sensor_type_desired_cooker_hob_front_middle_state	desired_cooker_hob_front_middle_state with values in the format csr_mesh_cooker_hob_state_t
csr_mesh_sensor_type_cooker_hob_back_right_state	cooker_hob_back_right_state with values in the format csr_mesh_cooker_hob_state_t
csr_mesh_sensor_type_desired_cooker_hob_back_right_state	desired_cooker_hob_back_right_state with values in the format csr_mesh_cooker_hob_state_t
csr_mesh_sensor_type_cooker_hob_front_right_state	cooker_hob_front_right_state with values in the format csr_mesh_cooker_hob_state_t
csr_mesh_sensor_type_desired_cooker_hob_front_right_state	desired_cooker_hob_front_right_state with values in the format csr_mesh_cooker_hob_state_t
csr_mesh_sensor_type_desired_wakeup_alarm_time	desired_wakeup_alarm_time with values in the format csr_mesh_minutes_of_the_day_t
csr_mesh_sensor_type_desired_second_wakeup_alarm_time	desired_second_wakeup_alarm_time with values in the format csr_mesh_minutes_of_the_day_t

Enumeration	Description
csr_mesh_sensor_type_passive_infrared_state	passive_infrared_state with values in the format csr_mesh_movement_state_t
csr_mesh_sensor_type_water_flowing	water_flowing with values in the format csr_mesh_water_flow_rate_t
csr_mesh_sensor_type_desired_water_flow	desired_water_flow with values in the format csr_mesh_water_flow_rate_t
csr_mesh_sensor_type_audio_level	audio_level with values in the format csr_mesh_decibel_t
csr_mesh_sensor_type_desired_audio_level	desired_audio_level with values in the format csr_mesh_decibel_t
csr_mesh_sensor_type_fan_speed	fan_speed with values in the format csr_mesh_percentage_t
csr_mesh_sensor_type_desired_fan_speed	desired_fan_speed with values in the format csr_mesh_forward_backward_t
csr_mesh_sensor_type_wind_speed	wind_speed with values in the format csr_mesh_centimetres_per_second_t
csr_mesh_sensor_type_wind_speed_gust	wind_speed_gust with values in the format csr_mesh_centimetres_per_second_t
csr_mesh_sensor_type_wind_direction	wind_direction with values in the format csr_mesh_direction_t
csr_mesh_sensor_type_wind_direction_gust	wind_direction_gust with values in the format csr_mesh_direction_t
csr_mesh_sensor_type_rain_fall_last_hour	rain_fall_last_hour with values in the format csr_mesh_millimetres_t
csr_mesh_sensor_type_rain_fall_today	rain_fall_today with values in the format csr_mesh_millimetres_t
csr_mesh_sensor_type_barometric_pressure	barometric_pressure with values in the format csr_mesh_air_pressure_t
csr_mesh_sensor_type_soil_temperature	soil_temperature with values in the format csr_mesh_temperature_kelvin_t

Enumeration	Description
csr_mesh_sensor_type_soil_moisure	soil_moisure with values in the format csr_mesh_percentage_t
csr_mesh_sensor_type_window_cover_position	window_cover_position with values in the format csr_mesh_percentage_t
csr_mesh_sensor_type_desired_window_cover_position	desired_window_cover_position with values in the format csr_mesh_percentage_t
csr_mesh_sensor_type_generic_1_byte	generic_1_byte with values in the format csr_mesh_generic_1_byte_t
csr_mesh_sensor_type_generic_2_byte	generic_2_byte with values in the format csr_mesh_generic_2_byte_t
csr_mesh_sensor_type_generic_1_byte_typed	generic_1_byte_typed with values in the format csr_mesh_generic_1_byte_typed_t
csr_mesh_sensor_type_generic_2_byte_typed	generic_2_byte_typed with values in the format csr_mesh_generic_2_byte_typed_t
csr_mesh_sensor_type_generic_3_byte_typed	generic_3_byte_typed with values in the format csr_mesh_generic_3_byte_typed_t

7.13 csr_mesh_beacon_type_t Enumeration Type Documentation

Syntax

```
enum csr_mesh_beacon_type_t
```

Description

Enumerations

Enumeration	Description
csr_mesh_beacon_type_csr	csr
csr_mesh_beacon_type_ibeacon	ibeacon

Enumeration	Description
csr_mesh_beacon_type_eddystone_url	eddystone_url
csr_mesh_beacon_type_eddystone_uid	eddystone_uid
csr_mesh_beacon_type_lte_direct	lte_direct
csr_mesh_beacon_type_lumicast	lumicast

7.14 csr_mesh_door_state_t Enumeration Type Documentation

Syntax

```
enum csr_mesh_door_state_t
```

Description

Enumerations

Enumeration	Description
csr_mesh_door_state_open_unlocked	open_unlocked
csr_mesh_door_state_closed_unlocked	closed_unlocked
csr_mesh_door_state_closed_locked	closed_locked

7.15 csr_mesh_window_state_t Enumeration Type Documentation

Syntax

```
enum csr_mesh_window_state_t
```

Description

Enumerations

Enumeration	Description
csr_mesh_window_state_closed_insecure	closed_insecure
csr_mesh_window_state_open_insecure	open_insecure
csr_mesh_window_state_open_secure	open_secure
csr_mesh_window_state_closed_secure	closed_secure

7.16 csr_mesh_seat_state_t Enumeration Type Documentation

Syntax

```
enum csr_mesh_seat_state_t
```

Description

Enumerations

Enumeration	Description
csr_mesh_seat_state_empty	empty
csr_mesh_seat_state_continuously_occupied	continuously_occupied
csr_mesh_seat_state_occupied	occupied

7.17 csr_mesh_appliance_state_t Enumeration Type Documentation

Syntax

```
enum csr_mesh_appliance_state_t
```

Description

Enumerations

Enumeration	Description
csr_mesh_appliance_state_pre_wash	pre_wash
csr_mesh_appliance_state_idle	idle
csr_mesh_appliance_state_post_wash	post_wash
csr_mesh_appliance_state_washing	washing
csr_mesh_appliance_state_heating_up	heating_up
csr_mesh_appliance_state_cooling_down	cooling_down

7.18 csr_mesh_cooker_hob_state_t Enumeration Type Documentation

Syntax

```
enum csr_mesh_cooker_hob_state_t
```

Description

Enumerations

Enumeration	Description
csr_mesh_cooker_hob_state_level_2	level_2
csr_mesh_cooker_hob_state_off_hot	off_hot
csr_mesh_cooker_hob_state_level_1	level_1
csr_mesh_cooker_hob_state_level_7	level_7
csr_mesh_cooker_hob_state_level_9	level_9
csr_mesh_cooker_hob_state_off_cold	off_cold
csr_mesh_cooker_hob_state_level_8	level_8
csr_mesh_cooker_hob_state_level_5	level_5
csr_mesh_cooker_hob_state_level_4	level_4
csr_mesh_cooker_hob_state_level_3	level_3
csr_mesh_cooker_hob_state_level_6	level_6

7.19 csr_mesh_movement_state_t Enumeration Type Documentation

Syntax

```
enum csr_mesh_movement_state_t
```

Description

Enumerations

Enumeration	Description
csr_mesh_movement_state_no_movement	no_movement
csr_mesh_movement_state_movement_detected	movement_detected

7.20 csr_mesh_forward_backward_t Enumeration Type Documentation

Syntax

```
enum csr_mesh_forward_backward_t
```

Description

Enumerations

Enumeration	Description
csr_mesh_forward_backward_forward	forward
csr_mesh_forward_backward_backwards	backwards

7.21 csr_mesh_direction_t Enumeration Type Documentation

Syntax

```
enum csr_mesh_direction_t
```

Description

Enumerations

Enumeration	Description
csr_mesh_direction_north_of_north_east	north_of_north_east
csr_mesh_direction_north	north
csr_mesh_direction_north_east	north_east
csr_mesh_direction_east	east
csr_mesh_direction_north_west	north_west
csr_mesh_direction_north_of_north_west	north_of_north_west
csr_mesh_direction_east_of_north_east	east_of_north_east
csr_mesh_direction_west	west
csr_mesh_direction_south_east	south_east
csr_mesh_direction_east_of_south_east	east_of_south_east
csr_mesh_direction_south_of_south_east	south_of_south_east
csr_mesh_direction_south	south
csr_mesh_direction_west_of_north_west	west_of_north_west
csr_mesh_direction_south_of_south_west	south_of_south_west
csr_mesh_direction_west_of_south_west	west_of_south_west
csr_mesh_direction_south_west	south_west

7.22 CSRMESH_MODEL_TYPE_T Enumeration Type Documentation

Syntax

```
enum CSRMESH_MODEL_TYPE_T
```

Description

Enumerations

Enumeration	Description
CSRMESH_WATCHDOG_MODEL	Type Watchdog model.
CSRMESH_CONFIG_MODEL	Type Config model.
CSRMESH_GROUP_MODEL	Type Group model.
CSRMESH_SENSOR_MODEL	Type Sensor model.
CSRMESH_ACTUATOR_MODEL	Type Actuator model.
CSRMESH_DATA_MODEL	Type Data model.
CSRMESH_BEARER_MODEL	Type Bearer model.
CSRMESH_PING_MODEL	Type Ping model.
CSRMESH_BATTERY_MODEL	Type Battery model.
CSRMESH_ATTENTION_MODEL	Type Attention model.
CSRMESH_POWER_MODEL	Type Power model.
CSRMESH_LIGHT_MODEL	Type Light model.
CSRMESH_ASSET_MODEL	Type Asset model.
CSRMESH_TRACKER_MODEL	Type Tracker model.
CSRMESH_TIME_MODEL	Type Time model.

Enumeration	Description
CSRMESH_SWITCH_MODEL	Type Switch model.
CSRMESH_TUNING_MODEL	Type Tuning model.
CSRMESH_EXTENSION_MODEL	Type Extension model.
CSRMESH_LARGEOBJECTTRANSFER_MODEL	Type LargeObjectTransfer model.
CSRMESH_FIRMWARE_MODEL	Type Firmware model.
CSRMESH_DIAGNOSTIC_MODEL	Type Diagnostic model.
CSRMESH_ACTION_MODEL	Type Action model.
CSRMESH_BEACON_MODEL	Type Beacon model.
CSRMESH_BEACONPROXY_MODEL	Type BeaconProxy model.
CSRMESH_ALL_MODELS	Type All models.

7.23 CSRMESH_MODEL_EVENT_T Enumeration Type Documentation

Syntax

```
enum CSRMESH_MODEL_EVENT_T
```

Description

Enumerations

Enumeration	Description
CSRMESH_WATCHDOG_MESSAGE	Upon receiving a WATCHDOG_MESSAGE message, if the RspSize field is set to a non-zero value, then the device shall respond with a WATCHDOG_MESSAGE with the RspSize field set to zero, and RspSize -1 octets of additional RandomData.
CSRMESH_WATCHDOG_SET_INTERVAL	Upon receiving a WATCHDOG_SET_INTERVAL message, the device shall save the Interval and ActiveAfterTime fields into the Interval and ActiveAfterTime variables and respond with a WATCHDOG_INTERVAL message with the current variable values.

Enumeration	Description
CSRMESH_WATCHDOG_INTERVAL	Watchdog interval state.
CSRMESH_CONFIG_LAST_SEQUENCE_NUMBER	Upon receiving a CONFIG_LAST_SEQUENCE_NUMBER message from a trusted device, the local device updates the SequenceNumber to at least one higher than the LastSequenceNumber in the message. Note: A trusted device means a device that is not only on the same CSRmesh network, having the same network key, but also interacted with in the past. This message is most useful to check if a device has been reset, for example when the batteries of the device are changed, but it does not remember its last sequence number in non-volatile memory.
CSRMESH_CONFIG_RESET_DEVICE	Upon receiving a CONFIG_RESET_DEVICE message from a trusted device directed at only this device, the local device sets the DeviceID to zero, and forgets all network keys, associated NetworkIVs and other configuration information. The device may act as if it is not associated and use MASP to re-associate with a network. Note: If the CONFIG_RESET_DEVICE message is received on any other destination address than the DeviceID of the local device, it is ignored. This is typically used when selling a device, to remove the device from the network of the seller so that the purchaser can associate the device with their network.
CSRMESH_CONFIG_SET_DEVICE_IDENTIFIER	When the device with a DeviceID of 0x0000 receives a CONFIG_SET_DEVICE_IDENTIFIER message and the DeviceHash of the message matches the DeviceHash of this device, the DeviceID of this device is set to the DeviceID field of this message. Then the device responds with the DEVICE_CONFIG_IDENTIFIER message using the new DeviceID as the source address. Note: This function is not necessary in normal operation of a CSRmesh network as DeviceID is distributed as part of the MASP protocol in the MASP_ID_DISTRIBUTION message.
CSRMESH_CONFIG_SET_PARAMETERS	Upon receiving a CONFIG_SET_PARAMETERS message, where the destination address is the DeviceID of this device, the device saves the TxInterval, TxDuration, RxDutyCycle, TxPower and TTL fields into the TransmitInterval, TransmitDuration, ReceiverDutyCycle, TransmitPower and DefaultTimeToLive state respectively. Then the device responds with a CONFIG_PARAMETERS message with the current configuration model state information.
CSRMESH_CONFIG_GET_PARAMETERS	Upon receiving a CONFIG_GET_PARAMETERS message, where the destination address is the DeviceID of this device, the device will respond with a CONFIG_PARAMETERS message with the current config model state information.
CSRMESH_CONFIG_PARAMETERS	Configuration parameters.
CSRMESH_CONFIG_DISCOVER_DEVICE	Upon receiving a CONFIG_DISCOVER_DEVICE message directed at the 0x0000 group identifier or to DeviceID of this device, the device responds with a CONFIG_DEVICE_IDENTIFIER message.
CSRMESH_CONFIG_DEVICE_IDENTIFIER	Device identifier.

Enumeration	Description
CSRMESH_CONFIG_GET_INFO	Upon receiving a CONFIG_GET_INFO message, directed at the DeviceID of this device, the device responds with a CONFIG_INFO message. The Info field of the CONFIG_GET_INFO message determines the information to be included in the CONFIG_INFO message. The following information values are defined: DeviceUUIDLow (0x00) contains the least significant eight octets of the DeviceUUID state value. DeviceUUIDHigh (0x01) contains the most significant eight octets of the DeviceUUID state value. ModelsLow (0x02) contains the least significant eight octets of the ModelsSupported state value. ModelsHigh (0x03) contains the most significant eight octets of the ModelsSupported state value.
CSRMESH_CONFIG_INFO	Current device information.
CSRMESH_GROUP_GET_NUMBER_OF_MODEL_GROUPIDS	Getting Number of Group IDs: Upon receiving a GROUP_GET_NUMBER_OF_MODEL_GROUPS message, where the destination address is the DeviceID of this device, the device responds with a GROUP_NUMBER_OF_MODEL_GROUPS message with the number of Group IDs that the given model supports on this device.
CSRMESH_GROUP_NUMBER_OF_MODEL_GROUPIDS	Get number of groups supported by the model.
CSRMESH_GROUP_SET_MODEL_GROUPID	Setting Model Group ID: Upon receiving a GROUP_SET_MODEL_GROUPID message, where the destination address is the DeviceID of this device, the device saves the Instance and GroupID fields into the appropriate state value determined by the Model and GroupIndex fields. It then responds with a GROUP_MODEL_GROUPID message with the current state information held for the given model and the GroupIndex values.
CSRMESH_GROUP_GET_MODEL_GROUPID	Getting Model Group ID: Upon receiving a GROUP_GET_MODEL_GROUPID message, where the destination address is the DeviceID of this device, the device responds with a GROUP_MODEL_GROUPID message with the current state information held for the given Model and GroupIndex values.
CSRMESH_GROUP_MODEL_GROUPID	GroupID of a model.
CSRMESH_SENSOR_GET_TYPES	Upon receiving a SENSOR_GET_TYPES message, the device responds with a SENSOR_TYPES message with the list of supported types greater than or equal to the FirstType field. If the device does not support any types greater than or equal to the FirstType field, then it sends a SENSOR_TYPES message with zero-length Types field.
CSRMESH_SENSOR_TYPES	Sensor types.
CSRMESH_SENSOR_SET_STATE	Setting Sensor State: Upon receiving a SENSOR_SET_STATE message, where the destination address is the device ID of this device and the Type field is a supported sensor type, the device saves the RxDutyCycle field and responds with a SENSOR_STATE message with the current state information of the sensor type. If the Type field is not a supported sensor type, the device ignores the message.

Enumeration	Description
CSRMESH_SENSOR_GET_STATE	Getting Sensor State: Upon receiving a SENSOR_GET_STATE message, where the destination address is the deviceID of this device and the Type field is a supported sensor type, the device shall respond with a SENSOR_STATE message with the current state information of the sensor type. Upon receiving a SENSOR_GET_STATE message, where the destination address is the device ID of this device but the Type field is not a supported sensor type, the device shall ignore the message.
CSRMESH_SENSOR_STATE	Current sensor state.
CSRMESH_SENSOR_WRITE_VALUE_NO_ACK	Writing Sensor Value: Upon receiving a SENSOR_WRITE_VALUE message, where the Type field is a supported sensor type, the device saves the value into the current value of the sensor type on this device and responds with a SENSOR_VALUE message with the current value of this sensor type.
CSRMESH_SENSOR_WRITE_VALUE	Writing Sensor Value: Upon receiving a SENSOR_WRITE_VALUE message, where the Type field is a supported sensor type, the device saves the value into the current value of the sensor type on this device and responds with a SENSOR_VALUE message with the current value of this sensor type.
CSRMESH_SENSOR_READ_VALUE	Getting Sensor Value: Upon receiving a SENSOR_READ_VALUE message, where the Type field is a supported sensor type, the device responds with a SENSOR_VALUE message with the value of the sensor type. Proxy Behaviour: Upon receiving a SENSOR_GET_STATE where the destination of the message and the sensor type correspond to a previously received SENSOR_BROADCAST_VALUE or SENSOR_BROADCAST_NEW message, the device responds with a SENSOR_VALUE message with the remembered values.
CSRMESH_SENSOR_VALUE	Current sensor value.
CSRMESH_SENSOR_MISSING	Sensor data is missing. Proxy Behaviour: Upon receiving a SENSOR_MISSING message, the proxy determines if it has remembered this type and value and then writes that type and value to the device that sent the message.
CSRMESH_ACTUATOR_GET_TYPES	Upon receiving an ACTUATOR_GET_TYPES message, the device responds with an ACTUATOR_TYPES message with a list of supported types greater than or equal to the FirstType field. If the device does not support any types greater than or equal to FirstType, it sends an ACTUATOR_TYPES message with a zero length Types field.
CSRMESH_ACTUATOR_TYPES	Actuator types.
CSRMESH_ACTUATOR_SET_VALUE_NO_ACK	Get sensor state. Upon receiving an ACTUATOR_SET_VALUE_NO_ACK message, where the destination address is the device ID of this device and the Type field is a supported actuator type, the device shall immediately use the Value field for the given Type field. The meaning of this actuator value is not defined in this specification. Upon receiving an ACTUATOR_SET_VALUE_NO_ACK message, where the destination address is is the device ID of this device but the Type field is not a supported actuator type, the device shall ignore the message.

Enumeration	Description
CSRMESH_ACTUATOR_SET_VALUE	Get sensor state. Upon receiving an ACTUATOR_SET_VALUE_NO_ACK message, where the destination address is the device ID of this device and the Type field is a supported actuator type, the device shall immediately use the Value field for the given Type field. The meaning of this actuator value is not defined in this specification. Upon receiving an ACTUATOR_SET_VALUE_NO_ACK message, where the destination address is is the device ID of this device but the Type field is not a supported actuator type, the device shall ignore the message.
CSRMESH_ACTUATOR_VALUE_ACK	Current sensor state.
CSRMESH_ACTUATOR_GET_VALUE_ACK	Get Current sensor state.
CSRMESH_DATA_STREAM_FLUSH	Flushing Data: Upon receiving a DATA_STREAM_FLUSH message, the device saves the StreamSN field into the StreamSequenceNumber model state and responds with DATA_STREAM_RECEIVED with the StreamNESN field set to the value of the StreamSequenceNumber model state. The device also flushes all partially received stream data from this peer device.
CSRMESH_DATA_STREAM_SEND	Sending Data: Upon receiving a DATA_STREAM_SEND message, the device first checks if the StreamSN field is the same as the StreamSequenceNumber model state. If these values are the same, the device passes the StreamOctets field up to the application for processing, and increments StreamSequenceNumber by the length of the StreamOctets field. It then responds with a DATA_STREAM_RECEIVED message with the current value of the StreamSequenceNumber. Note: The DATA_STREAM_RECEIVED message is sent even if the StreamSN received is different from the StreamSequenceNumber state. This allows missing packets to be detected and retransmitted by the sending device.
CSRMESH_DATA_STREAM_RECEIVED	Acknowledgement of data received.
CSRMESH_DATA_BLOCK_SEND	A block of data, no acknowledgement. Upon receiving a DATA_BLOCK_SEND message, the device passes the DatagramOctets field up to the application for processing.
CSRMESH_BEARER_SET_STATE	Setting Bearer State: Upon receiving a BEARER_SET_STATE message, where the destination address is the device ID of this device, the device saves the BearerRelayActive, BearerEnabled, and BearerPromiscuous fields into the appropriate state value. Then the device responds with a BEARER_STATE message with the current state information.
CSRMESH_BEARER_GET_STATE	Getting Bearer State: Upon receiving a BEARER_GET_STATE message, where the destination address is the device ID of this device, the device responds with a BEARER_STATE message with the current state information.
CSRMESH_BEARER_STATE	Set bearer state.

Enumeration	Description
CSRMESH_PING_REQUEST	Ping Request: Upon receiving a PING_REQUEST message, the device responds with a PING_RESPONSE message with the TTLAtRx field set to the TTL value from the PING_REQUEST message, and the RSSIAtRx field set to the RSSI value of the PING_REQUEST message. If the bearer used to receive the PING_REQUEST message does not have an RSSI value, then the value 0x00 is used.
CSRMESH_PING_RESPONSE	Ping response.
CSRMESH_BATTERY_GET_STATE	Getting Battery State: Upon receiving a BATTERY_GET_STATE message, the device responds with a BATTERY_STATE message with the current state information.
CSRMESH_BATTERY_STATE	Current battery state.
CSRMESH_ATTENTION_SET_STATE	Setting Flashing State: Upon receiving an ATTENTION_SET_STATE message, the device saves the AttractAttention and AttentionDuration fields into the appropriate state value. It then responds with an ATTENTION_STATE message with the current state information. If the AttractAttention field is set to 0x01 and the AttentionDuration is not 0xFFFF, then any existing attention timer is cancelled and a new attention timer is started that will expire after AttentionDuration milliseconds. If the AttractAttention field is set to 0x01 and the AttentionDuration field is 0xFFFF, then the attention timer is ignored. If the AttractAttention field is set to 0x00, then the attention timer is cancelled if it is already running.
CSRMESH_ATTENTION_STATE	Current battery state.
CSRMESH_POWER_SET_STATE_NO_ACK	Setting Power State: Upon receiving a POWER_SET_STATE_NO_ACK message, the device sets the PowerState state value to the PowerState field. It then responds with a POWER_STATE message with the current state information.
CSRMESH_POWER_SET_STATE	Setting Power State: Upon receiving a POWER_SET_STATE_NO_ACK message, the device sets the PowerState state value to the PowerState field. It then responds with a POWER_STATE message with the current state information.
CSRMESH_POWER_TOGGLE_STATE_NO_ACK	Toggling Power State: Upon receiving a POWER_Toggle_STATE_NO_ACK message, the device sets the PowerState state value as defined: 1.If the current PowerState is 0x00, Off, then PowerState should be set to 0x01, On. 2.If the current PowerState is 0x01, On, then PowerState should be set to 0x00, Off. 3.If the current PowerState is 0x02, Standby, then PowerState should be set to 0x03, OnFromStandby. 4.If the current PowerState is 0x03, OnFromStandby, then PowerState should be set to 0x02, Standby. Then the device responds with a POWER_STATE message with the current state information.

Enumeration	Description
CSRMESH_POWER_TOGGLE_STATE	Toggling Power State: Upon receiving a POWER_Toggle_STATE_NO_ACK message, the device sets the PowerState state value as defined: 1.If the current PowerState is 0x00, Off, then PowerState should be set to 0x01, On. 2.If the current PowerState is 0x01, On, then PowerState should be set to 0x00, Off. 3.If the current PowerState is 0x02, Standby, then PowerState should be set to 0x03, OnFromStandby. 4.If the current PowerState is 0x03, OnFromStandby, then PowerState should be set to 0x02, Standby. Then the device responds with a POWER_STATE message with the current state information.
CSRMESH_POWER_GET_STATE	Getting Power State: Upon receiving a POWER_GET_STATE message, the device responds with a POWER_STATE message with the current state information.
CSRMESH_POWER_STATE_NO_ACK	Current power state.
CSRMESH_POWER_STATE	Current power state.
CSRMESH_LIGHT_SET_LEVEL_NO_ACK	Setting Light Level: Upon receiving a LIGHT_SET_LEVEL_NO_ACK message, the device saves the Level field into the CurrentLevel model state. LevelSDState should be set to Idle. If ACK is requested, the device should respond with a LIGHT_STATE message.
CSRMESH_LIGHT_SET_LEVEL	Setting Light Level: Upon receiving a LIGHT_SET_LEVEL_NO_ACK message, the device saves the Level field into the CurrentLevel model state. LevelSDState should be set to Idle. If ACK is requested, the device should respond with a LIGHT_STATE message.
CSRMESH_LIGHT_SET_RGB_NO_ACK	Setting Light Colour: Upon receiving a LIGHT_SET_RGB_NO_ACK message, the device saves the Level, Red, Green, and Blue fields into the TargetLevel, TargetRed, TargetGreen, and TargetBlue variables respectively. LevelSDState should be set to Attacking. If the Duration field is zero, then the device saves the Level, Red, Green, and Blue fields into the CurrentLevel, CurrentRed, CurrentGreen and CurrentBlue variables, and sets the DeltaLevel, DeltaRed, DeltaGreen, and DeltaBlue variables to zero. If the Duration field is greater than zero, then the device calculates the DeltaLevel, DeltaRed, DeltaGreen, and DeltaBlue levels from the differences between the Current values and the Target values divided by the Duration field, so that over Duration seconds, the CurrentLevel, CurrentRed, CurrentGreen, and CurrentBlue variables are changed smoothly to the TargetLevel, TargetRed, TargetGreen and TargetBlue values. If ACK is requested, the device responds with a LIGHT_STATE message.

Enumeration	Description
CSRMESH_LIGHT_SET_RGB	Setting Light Colour: Upon receiving a LIGHT_SET_RGB_NO_ACK message, the device saves the Level, Red, Green, and Blue fields into the TargetLevel, TargetRed, TargetGreen, and TargetBlue variables respectively. LevelSDState should be set to Attacking. If the Duration field is zero, then the device saves the Level, Red, Green, and Blue fields into the CurrentLevel, CurrentRed, CurrentGreen and CurrentBlue variables, and sets the DeltaLevel, DeltaRed, DeltaGreen, and DeltaBlue variables to zero. If the Duration field is greater than zero, then the device calculates the DeltaLevel, DeltaRed, DeltaGreen, and DeltaBlue levels from the differences between the Current values and the Target values divided by the Duration field, so that over Duration seconds, the CurrentLevel, CurrentRed, CurrentGreen, and CurrentBlue variables are changed smoothly to the TargetLevel, TargetRed, TargetGreen and TargetBlue values. If ACK is requested, the device responds with a LIGHT_STATE message.
CSRMESH_LIGHT_SET_POWER_LEVEL_NO_ACK	Setting Light Power and Light Level: Upon receiving a LIGHT_SET_POWER_LEVEL_NO_ACK message, the device sets the current PowerState to the Power field, the TargetLevel variable to the Level field, the DeltaLevel to the difference between TargetLevel and CurrentLevel divided by the LevelDuration field, saves the Sustain and Decay fields into the LevelSustain and LevelDecay variables, and sets LevelSDState to the Attacking state. If ACK is requested, the device should respond with a LIGHT_STATE message.
CSRMESH_LIGHT_SET_POWER_LEVEL	Setting Light Power and Light Level: Upon receiving a LIGHT_SET_POWER_LEVEL_NO_ACK message, the device sets the current PowerState to the Power field, the TargetLevel variable to the Level field, the DeltaLevel to the difference between TargetLevel and CurrentLevel divided by the LevelDuration field, saves the Sustain and Decay fields into the LevelSustain and LevelDecay variables, and sets LevelSDState to the Attacking state. If ACK is requested, the device should respond with a LIGHT_STATE message.
CSRMESH_LIGHT_SET_COLOR_TEMP	Setting Light Colour Temperature: Upon receiving a LIGHT_SET_COLOR_TEMP message, the device saves the ColorTemperature field into the TargetColorTemperature state variable. If the TempDuration field is zero, the CurrentColorTemperature variable is set to TargetColorTemperature and DeltaColorTemperature is set to zero. If the TempDuration field is greater than zero, then the device calculates the difference between TargetColorTemperature and CurrentColorTemperature, over the TempDuration field and store this into a DeltaColorTemperature state variable, so that over TempDuration seconds, CurrentColorTemperature changes smoothly to TargetColorTemperature. The device then responds with a LIGHT_STATE message.
CSRMESH_LIGHT_GET_STATE	Getting Light State: Upon receiving a LIGHT_GET_STATE message, the device responds with a LIGHT_STATE message.
CSRMESH_LIGHT_STATE_NO_ACK	Current light state.
CSRMESH_LIGHT_STATE	Current light state.
CSRMESH_LIGHT_SET_WHITE_NO_ACK	Setting Light White level.

Enumeration	Description
CSRMESH_LIGHT_SET_WHITE	Setting Light White level.
CSRMESH_LIGHT_GET_WHITE	Setting Light White level.
CSRMESH_LIGHT_WHITE_NO_ACK	Setting Light White level.
CSRMESH_LIGHT_WHITE	Setting Light White level.
CSRMESH_ASSET_SET_STATE	Setting Asset State: Upon receiving an ASSET_SET_STATE message, the device saves the Interval, SideEffects, ToDestination, TxPower, Number of Announcements and AnnounceInterval fields into the appropriate state values. It then responds with an ASSET_STATE message with the current state information.
CSRMESH_ASSET_GET_STATE	Getting Asset State: Upon receiving an ASSET_GET_STATE message, the device responds with an ASSET_STATE message with the current state information.
CSRMESH_ASSET_STATE	Current asset state.
CSRMESH_ASSET_ANNOUNCE	Asset announcement.
CSRMESH_TRACKER_FIND	Finding an Asset: Upon receiving a TRACKER_FIND message, the server checks its tracker cache to see if it has received an ASSET_ANNOUNCE message recently that has the same DeviceID. If it finds one, it will send a TRACKER_FOUND message with the cached information.
CSRMESH_TRACKER_FOUND	Asset found.
CSRMESH_TRACKER_REPORT	Asset report.
CSRMESH_TRACKER_CLEAR_CACHE	Clear tracker cache.
CSRMESH_TRACKER_SET_PROXIMITY_CONFIG	Set tracker proximity config.
CSRMESH_TIME_SET_STATE	Setting Time Broadcast Interval: Upon receiving a TIME_SET_STATE message, the device saves the TimeInterval field into the appropriate state value. It then responds with a TIME_STATE message with the current state information.
CSRMESH_TIME_GET_STATE	Getting Time Broadcast Interval: Upon receiving a TIME_GET_STATE message, the device responds with a TIME_STATE message with the current state information.
CSRMESH_TIME_STATE	Set time broadcast interval.

Enumeration	Description
CSRMESH_TIME_BROADCAST	Synchronise wall clock time from client device: This message is always sent with TTL=0. This message is sent at intervals by the clock master. It is always sent with TTL=0. It is repeated, but the time is updated before each repeat is sent. The clock master repeats the message 5 times, relaying stations repeat it 3 times. When a node receives a clock broadcast its behaviour depends on the current clock state: n MASTER: Ignore broadcasts.n INIT: Start the clock; relay this message. Set state to NO_RELAY if MasterFlag set, otherwise RELAY_MASTER. Start relay timer.n RELAY: Correct clock if required. Relay this message. Set state to NO_RELAY if MasterFlag set, otherwise RELAY_MASTER. Start relay timer.n NO_RELAY: Ignore. State will be reset to RELAY when the relay timer goes off.n RELAY_MASTER: Relay message only if it is from the clock master and set state to NO_RELAY.n The relay timer is by default 1/4 of the clock broadcast interval (15 seconds if the interval is 60 seconds). This means that each node will relay a message only once, and will give priority to messages from the clock master (which always causes the clock to be corrected). Messages from other nodes will only cause clock correction if they exceed the max clock skew (250ms).
CSRMESH_TUNING_PROBE	Tuning Probe: The Tuning Probe message is sent to discover neighbours. This messages is issued by devices wanting to determine their density metrics.n The message is sent in two forms. A short form omitting both ScanDutyCycle and BatteryState with a TTL=0. This allows immediate neighbours to perform various calculations and in turn provide their own PROBE messages. The long version is only provided with TTL>0. This cannot be used for immediate neighbour density determination, but can be used to determine the overall network density. The ability to identify if a node is a potential pinch-point in the Mesh network can be achieved through the comparison of immediate and average number of neighbours within the network. The usage of the PROBE message with TTL=0 or TTL>0 is a way to perform these computations. It is worth noting that the periodicity of these two types of messages are different; messages with TTL>0 is much more infrequent than messages with TTL=0. Furthermore, it is wise not to use messages for TTL>0 and embedded values in the determination of the average values. The AverageNumNeighbour field is fixed point 6.2 format encoded. The ScanDutyCycle is expressing percentage for numbers from 1 to 100 and (x-100)/10 percentage for numbers from 101 to 255.
CSRMESH_TUNING_GET_STATS	Getting Tuning Stats: These messages are aimed at collecting statistics from specific nodes. This message allows for the request of all information or for some of its parts. Responses are multi-parts, each identified with an index (combining a continuation flag - top bit). MissingReplyParts for the required field serves at determining the specific responses one would like to collect. If instead all the information is requested, setting this field to zero will inform the destination device to send all messages. Importantly, response (STATS_RESPONSE) messages will not necessarily come back in order, or all reach the requestor. It is essential to handle these cases in the treatment of the collected responses.

Enumeration	Description
CSRMESH_TUNING_STATS	Current Asset State: Response to the request. The PartNumber indicates the current index of the message, the top bit of this field is used to indicate that more messages are available after this part. For example, a response made up of three messages will have part numbers 0x80, 0x81 and 0x02. Each message has a maximum of two neighbours. The combination of these responses and PROBE (TTL>0) are a means to establish an overall perspective of the entire Mesh Network.
CSRMESH_TUNING_ACK_CONFIG	Current tuning config for a device: This message comes as a response to a SET_CONFIG. Encoding its various fields follow the same convention as the ones exposed in SET_CONFIG.
CSRMESH_TUNING_SET_CONFIG	Setting (or reading) tuning config: Omitted or zero fields mean do not change. This message enforces the state of the recipient. The state is defined by two goals, normal and high traffic, and their associated number of neighbour to decide which cases to follow. Goals are expressed with unit-less values ranging from 0 to 255. These goals relate to metrics calculated on the basis of density computed at the node and across the network. The expectation is for these goals to be maintained through modification of the receive duty cycle. The average of number of neighbours for high and normal traffic is expressed as a ratio, both numbers sharing the same denominator and each representing their respective numerators. The duty cycle encoding follows the same rules as per duty cycle encoding encountered in PROBE message. This message comes in two formats. A fully truncated form containing only the OpCode (thus of length 2) is used to indicate a request for information. This message should be answered by the appropriate ACK_CONFIG. Further interpretations of this message are: 1. Missing ACK field implies that a request for ACK_CONFIG is made. Thus, this is a special case of the fully truncated mode. However, the provided fields are meant to be used in the setting of goals. 2. Individual fields with zero value are meant NOT to be changed in the received element. Same as for missing fields in truncated messages. Furthermore, in order to improve testing, a combination of values for main and high goals are conventionally expected to be used for defining two behaviours: 1. Suspended: Tuning Probe messages (TTL=0) should be sent and statistics maintained, but the duty cycle should not be changed - thus goals will never be achieved. The encoding are: Main Goal = 0x00 and High Goal = 0xFE. 2. Disable: No Tuning Probe message should be sent and statistics should not be gathered - averaged values should decay. The encoding are: Main Goal = 0x00 and High Goal = 0xFF.

Enumeration	Description
CSRMESH_EXTENSION_REQUEST	Request for Extension OpCode to be approved by the whole Mesh. A device wanting to use an OpCode, makes a request to the entire Mesh Network. This message is issued to target identity 0. The device waits some time, proportional to the size of the Mesh network and only after this period, messages using these proposed OpCode are used. Device receiving this message and wanting to oppose the usage of such code will respond to the source node with a CONFLICT. In case no conflict is known and the OpCode is for a message the node is interested in implementing (through comparison with hash value), a record of the OpCode and its mapping is kept. Request messages are relayed in cases of absence of conflict. The hash function is SHA-256, padded as per SHA-256 specifications2, for which the least significant 6 bytes will be used in the message. The range parameter indicates the maximum number of OpCode reserved from the based provided in the Proposed OpCode field. The last OpCode reserved is determined through the sum of the Proposed OpCode with the range value. This range parameter varies from 0 to 127, leaving the top bit free.
CSRMESH_EXTENSION_CONFLICT	Response to a REQUEST - only issued if a conflict is noticed. This message indicates that the proposed OpCode is already in use within the node processing the request message. Nodes receiving conflict extension will process this message and remove the conflicting OpCode from the list of OpCodes to handle. All conflict messages are relayed, processed or not. If a node receiving a REQUEST is able to match the hash of the provider previously assigned to an existing OpCode, but different to the proposed one, it responds with a CONFLICT with a reason combining the top bit with the previously associated range (0x80 | <old range>=""">). In such cases, the previously used OpCode (start of range) will be placed in the ProposedOpCode. Nodes receiving this conflict message with the top bit raised, will discard the initially proposed OpCode and replace it with the proposed code supplied in the conflict message.
CSRMESH_LARGEOBJECTTRANSFER_ANNOUNCE	A node wanting to provide a large object to neighbouring Mesh Nodes issues an ANNOUNCE with the associated content type. This message will have TTL=0, thus will only be answered by its immediate neighbours. The ANNOUNCE has the total size of the packet to be issued. The format and encoding of the large object is subject to the provided type and is out of scope of this document. The destination ID can either be 0, a group or a specific Device ID. In case the destination ID is not zero, only members of the group (associated with the LOT model) or the device with the specified Device ID responds with the intent to download the object for their own consumption. Every other node either ignores or accepts the offer for the purpose of relaying the packet.
CSRMESH_LARGEOBJECTTRANSFER_INTEREST	In case a node is ready to receive the proposed object, it responds with this message. The intended behaviour of the Large Object Transfer is to allow a Peer-to-Peer connection between the consumer and the producer. The consumer uses a ServiceID, part of which is randomly selected. The top 64 bits are 0x1122334455667788, the least significant 63 bits are randomly selected by the consumer node. The most significant bit of the least significant 64 bits is an encoding of the intent to relay the received data. Once this message has been issued, the consumer node starts advertising a connectable service with the 128-bits service composed through the concatenation of the fixed 64 bits and the randomly selected 63 bits. The duration of the advertisement is an implementation decision.

Enumeration	Description
CSRMESH_FIRMWARE_GET_VERSION	Get firmwre verison: Upon receiving a FIRMWARE_GET_VERSION the device reponds with a FIRMWARE_VERSION message containing the current firmware version.
CSRMESH_FIRMWARE_VERSION	Firmware version information.
CSRMESH_FIRMWARE_UPDATE_REQUIRED	Requesting a firmware update. Upon receiving this message, the device moves to a state where it is ready for receiving a firmware update.
CSRMESH_FIRMWARE_UPDATE_ACKNOWLEDGED	Acknowledgement message to the firmware update request.
CSRMESH_DIAGNOSTIC_STATE	When received this message is interpreted as to reconfigure the set of information collected. Statistics gathering can be turned on/off ? in the off mode no measurement of messages count and RSSI measurements will be made. RSSI binning can be stored, such that collection ALL messages? RSSI (MASP/MCP, irrespective of encoding) are split between a given number of bin, each of equal dimensions. Masking of individual broadcast channel can be specified, resulting in the collection of information specifically on the selected channels. A REST bit is also available. When present all the accumulated data will be cleared and all counters restarted. Note that it is possible to change various configurations without the RESET flag, this will result in the continuation of accumulation and therefore incoherent statistics.
CSRMESH_ACTION_SET_ACTION	This message defines the action which needs to be taken. This is a multiple parts message, each part will be acknowledged through an ACTION_SET_ACK. The message's target will store this payload and upon defined trigger, will generate a mesh message using the payload. ACTION_SET_ACTION has several parameters, in particular the destination's device ID. This Device ID can be the device itself, in which case the execution of the payload will be equivalent for this device to receive the message itself, it is up to the implementation to decide whether or not the message should be made visible to the rest of the Mesh. A Transaction ID is also provided. Since an ACTION_SET_ACTION may be composed of several parts, this transaction ID could be incremented with every parts however, since the ACTION_SET_ACTION message has within its ActionID field a part number, the same transaction ID can be re-uses and the combination of both part-number and transaction ID can be used to identify each individual parts. Whilst there is no restriction in the type of Message one can include in the payload, the current implementation does not handle messages which would come as multiple parts and therefore does not handle the embedding of the ACTION_SET_ACTION! The ActionID will be used by the target and node initiating this message as a mean to identify the action in future exchange. Note that there are no inherent mechanisms protecting against clashes if more than one device was elected to be the source of Action provision.
CSRMESH_ACTION_SET_ACTION_ACK	Every parts received is acknowledged, providing both Transaction ID and ActionID (which includes Part Number)
CSRMESH_ACTION_GET_ACTION_STATUS	Request towards a node about the various Actions currently handled. This will be answered through an ACTION_STATUS. A Transaction ID is provided to ensure identification of response with request.

Enumeration	Description
CSRMESH_ACTION_ACTION_STATUS	This provides a bitmask identifying all the currently allocated ActionID. The Action Supported field provides an indication on how many actions this device has capacity for. Note that a device will never have more than 32 actions.
CSRMESH_ACTION_DELETE	This message allows a set of actions to be removed. Actions are identified through a bitmask reflecting the ActionID one wishes to delete. This message will be acknowledged through ACTION_DELETE_ACK.
CSRMESH_ACTION_DELETE_ACK	This message confirms the delete commands and report which actions have been removed. The provided transaction ID will match the one supplied by ACTION_DELETE.
CSRMESH_ACTION_GET	Using this message, it is possible to interrogate a node about the specific payload associated with an Action ID. This will be answered by and ACTION_SET_ACTION message, ACTION_SET_ACTION_ACK will need to be issued in order to collect all parts.
CSRMESH_BEACON_SET_STATUS	May be sent to a beacon to set its status. More than one such command can be sent to set the intervals for different beacon types, which need not be the same. This message also allows for the wakeup intervals to be set if the beacon elects to only be on the mesh sporadically. The time is always with respect to the beacon's internal clock and has no absolute meaning. This message will be answered through BEACON_STATUS.
CSRMESH_BEACON_GET_BEACON_STATUS	Message to be sent to a beacon in order to recover its current status. This message fetch information pertinent to a particular type. The transaction ID will help matching the pertinent BEACON_STATUS message a different transaction ID shall be used for a different type (although the BEACON_STATUS message has the type pertinent to the status and thus could be used in conjunction to the Transaction ID for disambiguation).
CSRMESH_BEACON_BEACON_STATUS	This message issue issued by a Beacon as response to BEACON_SET_STATUS or BEACON_GET_STATUS. Furthermore, a Beacon appearing on the mesh sporadically, will issue such message (with destination ID set to 0) as soon as it re-joins the mesh. In this case, one message per active beacon type should be issued. This message will provide the Payload ID currently in used for the associated type.
CSRMESH_BEACON_GET_TYPES	Message allowing a node to fetch all supported types and associated information by a given Beacon.
CSRMESH_BEACON_TYPES	Provides information on the set of beacon supported by the node. The battery level is reported as well as time since last message.

Enumeration	Description
CSRMESH_BEACON_SET_PAYLOAD	One or more of these packets may be sent to a beacon to set its payload. The content is either the raw advert data, or extra information (such as crypto cycle) which a beacon may require. The payload data is sent as a length and an offset, so the whole payload need not be sent if it has not changed. A beacon can support many beacon types - it can be sent as many different payloads as needed, one for each type. The first byte of the first payload packet contains the length and offset of the payload and the payload ID; this allows a beacon which already has the payload to send an immediate acknowledgement, saving traffic. This ID will be sent back in the 'Payload ACK' message if the beacon has received the whole payload. The payload may have to be split in several parts, in which case only the last part shall be acknowledged. Upon receiving a BEACON_SET_PAYLOAD, the beacon will update its corresponding beacon data if data was previously available, it will be replaced by the provided payload, thus a beacon can only hold one payload per beacon type.
CSRMESH_BEACON_PAYLOAD_ACK	Only one Acknowledgement message will occur for multiple BEACON_SET_PAYLOAD messages sharing the same transaction ID. Only when the last segment is received that such acknowledgement will be issued. Where missing payload messages exist, the list of their indices will be provided in the Ack field.
CSRMESH_BEACON_GET_PAYLOAD	Message allowing a node to retrieve the current payload held on a given beacon. This message shall be answered by one or more BEACON_SET_PAYLOAD messages.
CSRMESH_BEACONPROXY_ADD	Message can be sent to a beacon-proxy to add to the list of devices it manages. This request will be acknowledged using a Proxy Command Status Devices. Up to four device ID can be specified in this message Group can be defined. This message also permits the flushing of pending messages for managed IDs.
CSRMESH_BEACONPROXY_REMOVE	May be sent to a proxy to remove from the list of devices it manages. Any message queued for those devices will be cleared. This request will be acknowledged using a Proxy Command Status Devices. Groups can be used, implying that all its members will be removed from the proxy management. Specifying 0 in the Device Addresses will be interpreted as stopping all Proxy activities on the targeted proxy.
CSRMESH_BEACONPROXY_COMMAND_STATUS_DEVICES	Generic acknowledgement - the transaction ID permits reconciliation with sender.
CSRMESH_BEACONPROXY_GET_STATUS	Fetch the current status - this will be answered by a BEACON_PROXY_STATUS.
CSRMESH_BEACONPROXY_PROXY_STATUS	Provides generic information on the internal states of a Proxy. Including number of managed nodes, groups, states of the various queues.
CSRMESH_BEACONPROXY_GET_DEVICES	Fetch the current set of devices ID managed by a given proxy. This will be answered by one or more BEACON_PROXY_DEVICES messages.
CSRMESH_BEACONPROXY_DEVICES	Provide the list of Devices and Groups currently managed by a given Proxy. This will be a multiple parts message.

8 Watchdog Model

Detailed Description

Watchdog Model API.

The Watchdog Model uses the concept of an interval that allows a device to send out a watchdog message at a known frequency. This message can be used by devices to determine whether the sending device is still connected to the network. For example, a window sensor that only transmits when it detects an alarm condition uses the watchdog message to inform other devices that it is still alive. The periodicity of these watchdog messages can be configured on each device, allowing devices that need shorter reporting intervals to be configured accordingly. After these watchdog messages are sent, the devices can stay active for a short period of time for the capture of responses and for other devices to reconfigure these devices. This is most useful to allow devices that normally never listen for CSRmesh messages to be reconfigured or managed by devices. For example, a temperature sensor may only transmit the current room temperature, and send out a watchdog message once a day; when this single watchdog message is received, the sending device would stay active and another device could check the firmware version and possibly request the sending device to start to update its firmware. This enables the lowest possible power operation for the majority of the time whilst still allowing devices to be reconfigured within a known period of time. Watchdog messages can also be used to ensure that a predetermined pattern of activity is maintained. This is done by issuing watchdog messages where the expected messages should be sent but for some reason are not sent, for example when a home owner is on vacation. This ensures that a similar traffic pattern is maintained and thus offers protection against traffic analysis and occupancy detection. For example, a light switch may send out a few messages between 07:00 and 08:00 in the morning, and a few messages between 19:00 and 22:00. It can simulate this by sending watchdog messages that are sized similarly. It can even simulate the appropriate responses from devices by requesting a response with a similarly appropriate size. Devices can therefore protect against passive eavesdroppers monitoring the occupancy of a building from the quantity, size, or timing of messages.

If a device does not send any messages, but it should normally send some messages, for example because the building is currently unoccupied, the device may send one or more WATCHDOG_MESSAGE s to simulate the typical functionality of the device. The size of the WATCHDOG_MESSAGE s should be the same as the typical MCP messages that the device normally sends, and the RspSize should be set to the size of the typical responses to these messages. The MCP Dst field should also be set to the typical destination identifiers this device uses.

This behaviour prevents a passive eavesdropper from performing coarse traffic analysis to determine if a building is occupied and the coarse location of people in this building<h4> Background BehaviourThe device should transmit a new random WATCHDOG_MESSAGE at least once every Interval state value seconds. A random message is defined as one that has the RspSize field set to the value zero, and the RandomData field set to a random number of random values, from 0 to 9 octets in length. The ActiveAfterTime state value determines for how long a device should listen for the CSRmesh packets after transmitting a WATCHDOG_MESSAGE. This allows devices to be reconfigured at least once every Interval by waiting for a WATCHDOG_MESSAGE to be received from

that device and then sending them another message at this time. A device may transmit a new random WATCHDOG_MESSAGE at similar intervals to the normal operation of the device.

8.1 Modules of Watchdog Model

Client

Server

8.2 Data Structures of Watchdog Model

`struct CSRMESH_WATCHDOG_MESSAGE_T`

CSRmesh Watchdog Model message types.

`struct CSRMESH_WATCHDOG_SET_INTERVAL_T`

Upon receiving a WATCHDOG_SET_INTERVAL message, the device shall save the Interval and ActiveAfterTime fields into the Interval and ActiveAfterTime variables and respond with a WATCHDOG_INTERVAL message with the current variable values.

`struct CSRMESH_WATCHDOG_INTERVAL_T`

Watchdog interval state.

9 Client

Detailed Description

9.1 Functions of Client

```
CSRmeshResult WatchdogModelClientInit (CSRMESH_MODEL_CALLBACK_T app_callback)
```

Initialises Watchdog Model Client functionality.

```
CSRmeshResult WatchdogClientMessage (CsrUint8 nw_id, CsrUint16 dest_id,
CsrUint8 ttl, CSRMESH_WATCHDOG_MESSAGE_T *p_params)
```

Upon receiving a WATCHDOG_MESSAGE message, if the RspSize field is set to a non-zero value, then the device shall respond with a WATCHDOG_MESSAGE with the RspSize field set to zero, and RspSize -1 octets of additional RandomData.

```
CSRmeshResult WatchdogSetInterval (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8
ttl, CSRMESH_WATCHDOG_SET_INTERVAL_T *p_params)
```

Upon receiving a WATCHDOG_SET_INTERVAL message, the device shall save the Interval and ActiveAfterTime fields into the Interval and ActiveAfterTime variables and respond with a WATCHDOG_INTERVAL message with the current variable values.

9.2 WatchdogModelClientInit Function Documentation

Syntax

```
CSRmeshResult WatchdogModelClientInit (CSRMESH_MODEL_CALLBACK_T app_callback)
```

Description

Parameters

Parameter	Description
app_callback	Pointer to the application callback function that will be called when the model client receives a message.

Returns

CSRmeshResult. Refer to CSRmeshResult.

9.3 WatchdogClientMessage Function Documentation

Syntax

```
CSRmeshResult WatchdogClientMessage (CsrUint8 nw_id, CsrUint16 dest_id,
CsrUint8 ttl, CSRMESH_WATCHDOG_MESSAGE_T * p_params)
```

Description

This function packs the given parameters into a CSRmesh message and sends it over the network. When a response is received the application callback function is called to notify the CSRMESH_WATCHDOG_MESSAGE

Parameters

Parameter	Description
nw_id	Network identifier over which the message has to be sent.
dest_id	16-bit identifier of the destination device/group
ttl	TTL value with which the message needs to be sent.
p_params	Pointer to the message parameters of type CSRMESH_WATCHDOG_MESSAGE_T

Returns

CSRmeshResult. Refer to CSRmeshResult.

9.4 WatchdogSetInterval Function Documentation

Syntax

```
CSRmeshResult WatchdogSetInterval (CsrUint8 nw_id, CsrUint16 dest_id,
CsrUint8 ttl, CSRMESH_WATCHDOG_SET_INTERVAL_T * p_params)
```

Description

This function packs the given parameters into a CSRmesh message and sends it over the network. When a response is received the application callback function is called to notify the CSRMESH_WATCHDOG_INTERVAL

Parameters

Parameter	Description
`nw_id`	Network identifier over which the message has to be sent.
`dest_id`	16-bit identifier of the destination device/group
`ttl`	TTL value with which the message needs to be sent.
`p_params`	Pointer to the message parameters of type CSRMESH_WATCHDOG_SET_INTERVAL_T

Returns

CSRmeshResult. Refer to CSRmeshResult.

10 Server

Detailed Description

10.1 Functions of Server

```
CSRmeshResult WatchdogModelInit (CsrUint8 nw_id, CsrUint16 *group_id_list,
CsrUint16 num_groups, CSRMESH_MODEL_CALLBACK_T app_callback)
```

Model Initialisation.

```
CSRmeshResult WatchdogMessage (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8 ttl,
CSRMESH_WATCHDOG_MESSAGE_T *p_params)
```

Upon receiving a WATCHDOG_MESSAGE message, if the RspSize field is set to a non-zero value, then the device shall respond with a WATCHDOG_MESSAGE with the RspSize field set to zero, and RspSize -1 octets of additional RandomData.

```
CSRmeshResult WatchdogInterval (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8
ttl, CSRMESH_WATCHDOG_INTERVAL_T *p_params)
```

Watchdog interval state.

10.2 WatchdogModelInit Function Documentation

Syntax

```
CSRmeshResult WatchdogModelInit (CsrUint8 nw_id, CsrUint16 * group_id_list,
CsrUint16 num_groups, CSRMESH_MODEL_CALLBACK_T app_callback)
```

Description

Registers the model handler with the CSRmesh. Sets the CSRmesh to report num_groups as the maximum number of groups supported for the model

Parameters

Parameter	Description
nw_id	Identifier of the network to which the model has to be registered.
group_id_list	Pointer to a uint16 array to hold assigned group_ids. This must be NULL if no groups are supported
num_groups	Size of the group_id_list. This must be 0 if no groups are supported.
app_callback	Pointer to the application callback function. This function will be called to notify all model specific messages

Returns

CSRmeshResult. Refer to CSRmeshResult.

10.3 WatchdogMessage Function Documentation

Syntax

```
CSRmeshResult WatchdogMessage (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8 ttl,
CSRMESH_WATCHDOG_MESSAGE_T * p_params)
```

Description

This function packs the given parameters into a CSRmesh message and sends it over the network. When a response is received the application callback function is called to notify the CSRMESH_WATCHDOG_MESSAGE

Parameters

Parameter	Description
nw_id	Network identifier over which the message has to be sent.
dest_id	16-bit identifier of the destination device/group
ttl	TTL value with which the message needs to be sent.
p_params	Pointer to the message parameters of type CSRMESH_WATCHDOG_MESSAGE_T

Returns

CSRmeshResult. Refer to CSRmeshResult.

10.4 WatchdogInterval Function Documentation

Syntax

```
CSRmeshResult WatchdogInterval (CsrUint8 nw_id, CsrUint16 dest_id,
CsrUint8 ttl, CSRMESH_WATCHDOG_INTERVAL_T * p_params)
```

Description

This function packs the given parameters into a CSRmesh message and sends it over the network.

Parameters

Parameter	Description
nw_id	Network identifier over which the message has to be sent.
dest_id	16-bit identifier of the destination device/group
ttl	TTL value with which the message needs to be sent.
p_params	Pointer to the message parameters of type CSRMESH_WATCHDOG_INTERVAL_T

Returns

CSRmeshResult. Refer to CSRmeshResult.

11 Config Model

Detailed Description

Config Model API.

The Configuration Model configures the device within MCP. It sets the ability to discover devices within the CSRmesh network, configure the Device ID for a given device, and discover device information such as the set of models a device supports or a device UUID. It also sets the communication parameters for this device including the receive and transmit duty cycles and time-to-live values for transmitted packets.

Many devices implement the configuration model because it provides the primary method of inspecting a device and configuring the device. It has been considered that this model should be made mandatory, however at this time it is not a mandatory model as some devices may not need the configuration model; a device that only transmits asset broadcast messages using the asset model may not need the configuration model to work.

The configuration model has the following states:

- DeviceUUID
- SequenceNumber
- DeviceID
- ModelsSupported
- TransmitInterval
- TransmitDuration
- ReceiverDutyCycle
- TransmitPower
- DefaultTimeToLive
- VendorIdentifier
- ProductIdentifier
- VersionNumber
- Appearance
- LastETag
- Conformance Signature
- Stack version

DeviceUUID is the 128-bit UUID, the unique identifier for the device. This value is fixed and cannot be changed. DeviceUUID is used to generate DeviceHash in some configuration messages.

SequenceNumber is a 24-bit unsigned integer.

DeviceID is a 16-bit value. This value is used as the source address of the device when sending MCP messages and as the destination address when other devices send packets to this device.

ModelsSupported is a 128-bit bit field that has a single bit allocated for each model. If a bit is set, then the server behaviour of the set model should be supported, otherwise the server behaviour of the model is not supported.

It is not possible to determine if a device supports the client behaviour of a model, as this would be typically be determined by the functionality that the device exposes to the user.

The TransmitInterval is a 16-bit unsigned integer in milliseconds. This value determines how often a device retransmits a CSRmesh message.

TransmitDuration is a 16-bit unsigned integer in milliseconds. This value determines how long a device retransmits a CSRmesh message.

The ReceiverDutyCycle is an 8-bit unsigned integer that is 1/255th of a second. This value determines how frequently a device listens for CSRmesh messages from other devices.

The larger this value, the more reliably this device will receive CSRmesh messages, but the device will consume more energy. If this is value is zero, then the device will not listen for messages.

TransmitPower is an 8-bit signed integer in dBm. This value determines how much energy is used when transmitting CSRmesh messages.

The larger this value, the more energy this device will consume but the greater the ranges in which this device will be able to transmit messages.

DefaultTimeToLive is an 8-bit unsigned integer.

VendorIdentifier is a 16-bit unsigned integer. This shall use the same enumerations as the existing Bluetooth SIG Company Identification assigned numbers.

The ProductIdentifier is a 16-bit unsigned integer allocated by the Vendor.

VersionNumber is a 32-bit unsigned integer allocated per product by the vendor. Each new version of the device shall have a different VersionNumber. A new version is defined as any change to the software, hardware or firmware of a device, even if this change has no material impact to the behaviour of the device.

Appearance is a 24-bit unsigned integer using the same the Bluetooth SIG GAP Appearance characteristic values.

LastETag is a 64-bit unsigned integer that should be set to a different value each time the device has a change in its configuration. This value is a read only value, and can be used to quickly determine if a device has changed its configuration without another device reading the full configuration. A configuration change is defined as any change to the device such that the device behaviour changes - for example, changing Groups or the interval for sending Sensor broadcast data.

If a device does not have a DeviceID, that is, the DeviceID is 0x0000, then it may periodically transmit a CONFIG_DEVICE_IDENTIFIER message to the destination address of 0x0000 with its DeviceHash.

11.1 Modules of Config Model

Client

Server

11.2 Data Structures of Config Model

`struct CSRMESH_CONFIG_LAST_SEQUENCE_NUMBER_T`

CSRmesh Config Model message types.

`struct CSRMESH_CONFIG_RESET_DEVICE_T`

Upon receiving a CONFIG_RESET_DEVICE message from a trusted device directed at only this device, the local device sets the DeviceID to zero, and forgets all network keys, associated NetworkIVs and other configuration information. The device may act as if it is not associated and use MASP to re-associate with a network. Note: If the CONFIG_RESET_DEVICE message is received on any other destination address than the DeviceID of the local device, it is ignored. This is typically used when selling a device, to remove the device from the network of the seller so that the purchaser can associate the device with their network.

`struct CSRMESH_CONFIG_SET_DEVICE_IDENTIFIER_T`

When the device with a DeviceID of 0x0000 receives a CONFIG_SET_DEVICE_IDENTIFIER message and the DeviceHash of the message matches the DeviceHash of this device, the DeviceID of this device is set to the DeviceID field of this message. Then the device responds with the DEVICE_CONFIG_IDENTIFIER message using the new DeviceID as the source address. Note: This function is not necessary in normal operation of a CSRmesh network as DeviceID is distributed as part of the MASP protocol in the MASP_ID_DISTRIBUTION message.

`struct CSRMESH_CONFIG_SET_PARAMETERS_T`

Upon receiving a CONFIG_SET_PARAMETERS message, where the destination address is the DeviceID of this device, the device saves the TxInterval, TxDuration, RxDutyCycle, TxPower and TTL fields into the TransmitInterval, TransmitDuration, ReceiverDutyCycle, TransmitPower and DefaultTimeToLive state respectively. Then the device responds with a CONFIG_PARAMETERS message with the current configuration model state information.

`struct CSRMESH_CONFIG_GET_PARAMETERS_T`

Upon receiving a CONFIG_GET_PARAMETERS message, where the destination address is the DeviceID of this device, the device will respond with a CONFIG_PARAMETERS message with the current config model state information.

`struct CSRMESH_CONFIG_PARAMETERS_T`

Configuration parameters.

`struct CSRMESH_CONFIG_DISCOVER_DEVICE_T`

Upon receiving a CONFIG_DISCOVER_DEVICE message directed at the 0x0000 group identifier or to DeviceID of this device, the device responds with a CONFIG_DEVICE_IDENTIFIER message.

`struct CSRMESH_CONFIG_DEVICE_IDENTIFIER_T`

Device identifier.

```
struct  CSRMESH_CONFIG_GET_INFO_T
```

Upon receiving a CONFIG_GET_INFO message, directed at the DeviceID of this device, the device responds with a CONFIG_INFO message. The Info field of the CONFIG_GET_INFO message determines the information to be included in the CONFIG_INFO message. The following information values are defined: DeviceUUIDLow (0x00) contains the least significant eight octets of the DeviceUUID state value. DeviceUUIDHigh (0x01) contains the most significant eight octets of the DeviceUUID state value. ModelsLow (0x02) contains the least significant eight octets of the ModelsSupported state value. ModelsHigh (0x03) contains the most significant eight octets of the ModelsSupported state value.

```
struct  CSRMESH_CONFIG_INFO_T
```

Current device information.

12 Client

Detailed Description

12.1 Functions of Client

```
CSRmeshResult ConfigModelClientInit (CSRMESH_MODEL_CALLBACK_T app_callback)
```

Initialises Config Model Client functionality.

```
CSRmeshResult ConfigLastSequenceNumber (CsrUint8 nw_id, CsrUint16 dest_id,
CsrUint8 ttl, CSRMESH_CONFIG_LAST_SEQUENCE_NUMBER_T *p_params)
```

Upon receiving a CONFIG_LAST_SEQUENCE_NUMBER message from a trusted device, the local device updates the SequenceNumber to at least one higher than the LastSequenceNumber in the message. Note: A trusted device means a device that is not only on the same CSRmesh network, having the same network key, but also interacted with in the past. This message is most useful to check if a device has been reset, for example when the batteries of the device are changed, but it does not remember its last sequence number in non-volatile memory.

```
CSRmeshResult ConfigResetDevice (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8
ttl, CSRMESH_CONFIG_RESET_DEVICE_T *p_params)
```

Upon receiving a CONFIG_RESET_DEVICE message from a trusted device directed at only this device, the local device sets the DeviceID to zero, and forgets all network keys, associated NetworkIVs and other configuration information. The device may act as if it is not associated and use MASP to re-associate with a network. Note: If the CONFIG_RESET_DEVICE message is received on any other destination address than the DeviceID of the local device, it is ignored. This is typically used when selling a device, to remove the device from the network of the seller so that the purchaser can associate the device with their network.

```
CSRmeshResult ConfigSetDeviceIdentifier (CsrUint8 nw_id, CsrUint16 dest_id,
CsrUint8 ttl, CSRMESH_CONFIG_SET_DEVICE_IDENTIFIER_T *p_params)
```

When the device with a DeviceID of 0x0000 receives a CONFIG_SET_DEVICE_IDENTIFIER message and the DeviceHash of the message matches the DeviceHash of this device, the DeviceID of this device is set to the DeviceID field of this message. Then the device responds with the DEVICE_CONFIG_IDENTIFIER message using the new DeviceID as the source address. Note: This function is not necessary in normal operation of a CSRmesh network as DeviceID is distributed as part of the MASP protocol in the MASP_ID_DISTRIBUTION message.

```
CSRmeshResult ConfigSetParameters (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8
ttl, CSRMESH_CONFIG_SET_PARAMETERS_T *p_params)
```

Upon receiving a CONFIG_SET_PARAMETERS message, where the destination address is the DeviceID of this device, the device saves the TxInterval, TxDuration, RxDutyCycle, TxPower and TTL fields into the TransmitInterval, TransmitDuration, ReceiverDutyCycle, TransmitPower and DefaultTimeToLive state respectively. Then the device responds with a CONFIG_PARAMETERS message with the current configuration model state information.

```
CSRmeshResult ConfigGetParameters (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8
ttl, CSRMESH_CONFIG_GET_PARAMETERS_T *p_params)
```

Upon receiving a CONFIG_GET_PARAMETERS message, where the destination address is the DeviceID of this device, the device will respond with a CONFIG_PARAMETERS message with the current config model state information.

```
CSRmeshResult ConfigDiscoverDevice (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8
ttl, CSRMESH_CONFIG_DISCOVER_DEVICE_T *p_params)
```

Upon receiving a CONFIG_DISCOVER_DEVICE message directed at the 0x0000 group identifier or to DeviceID of this device, the device responds with a CONFIG_DEVICE_IDENTIFIER message.

```
CSRmeshResult ConfigGetInfo (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8 ttl,
CSRMESH_CONFIG_GET_INFO_T *p_params)
```

Upon receiving a CONFIG_GET_INFO message, directed at the DeviceID of this device, the device responds with a CONFIG_INFO message. The Info field of the CONFIG_GET_INFO message determines the information to be included in the CONFIG_INFO message. The following information values are defined: DeviceUUIDLow (0x00) contains the least significant eight octets of the DeviceUUID state value. DeviceUUIDHigh (0x01) contains the most significant eight octets of the DeviceUUID state value. ModelsLow (0x02) contains the least significant eight octets of the ModelsSupported state value. ModelsHigh (0x03) contains the most significant eight octets of the ModelsSupported state value.

```
CSRmeshResult ConfigSetMessageParams (CsrUint8 nw_id, CsrUint16 dest_id,
CsrUint8 ttl, CSRMESH_CONFIG_SET_MESSAGE_PARAMS_T *p_params)
```

This function packs the given parameters into a CSRmesh message and sends it over the network. When a response is received the application callback function is called to notify the CSRMESH_CONFIG_MESSAGE_PARAMS.

```
CSRmeshResult ConfigGetMessageParams (CsrUint8 nw_id, CsrUint16 dest_id,
CsrUint8 ttl, CSRMESH_CONFIG_GET_MESSAGE_PARAMS_T *p_params)
```

This function packs the given parameters into a CSRmesh message and sends it over the network. When a response is received the application callback function is called to notify the CSRMESH_CONFIG_MESSAGE_PARAMS.

12.2 ConfigModelClientInit Function Documentation

Syntax

```
CSRmeshResult ConfigModelClientInit (CSRMESH_MODEL_CALLBACK_T app_callback)
```

Description

Parameters

Parameter	Description
app_callback	Pointer to the application callback function that will be called when the model client receives a message.

Returns

CSRmeshResult. Refer to CSRmeshResult.

12.3 ConfigLastSequenceNumber Function Documentation

Syntax

```
CSRmeshResult ConfigLastSequenceNumber (CsrUint8 nw_id, CsrUint16 dest_id,
CsrUint8 ttl, CSRMESH_CONFIG_LAST_SEQUENCE_NUMBER_T * p_params)
```

Description

This function packs the given parameters into a CSRmesh message and sends it over the network.

Parameters

Parameter	Description
nw_id	Network identifier over which the message has to be sent.
dest_id	16-bit identifier of the destination device/group
ttl	TTL value with which the message needs to be sent.
p_params	Pointer to the message parameters of type CSRMESH_CONFIG_LAST_SEQUENCE_NUMBER_T

Returns

CSRmeshResult. Refer to CSRmeshResult.

12.4 ConfigResetDevice Function Documentation

Syntax

```
CSRmeshResult ConfigResetDevice (CsrUint8 nw_id, CsrUint16 dest_id,
CsrUint8 ttl, CSRMESH_CONFIG_RESET_DEVICE_T * p_params)
```

Description

This function packs the given parameters into a CSRmesh message and sends it over the network.

Parameters

Parameter	Description
nw_id	Network identifier over which the message has to be sent.
dest_id	16-bit identifier of the destination device/group
ttl	TTL value with which the message needs to be sent.
p_params	Pointer to the message parameters of type CSRMESH_CONFIG_RESET_DEVICE_T

Returns

CSRmeshResult. Refer to CSRmeshResult.

12.5 ConfigSetDeviceIdentifier Function Documentation

Syntax

```
CSRmeshResult ConfigSetDeviceIdentifier (CsrUint8 nw_id, CsrUint16 dest_id,
CsrUint8 ttl, CSRMESH_CONFIG_SET_DEVICE_IDENTIFIER_T * p_params)
```

Description

This function packs the given parameters into a CSRmesh message and sends it over the network. When a response is received the application callback function is called to notify the CSRMESH_CONFIG_DEVICE_IDENTIFIER

Parameters

Parameter	Description
nw_id	Network identifier over which the message has to be sent.
dest_id	16-bit identifier of the destination device/group
ttl	TTL value with which the message needs to be sent.
p_params	Pointer to the message parameters of type CSRMESH_CONFIG_SET_DEVICE_IDENTIFIER_T

Returns

CSRmeshResult. Refer to CSRmeshResult.

12.6 ConfigSetParameters Function Documentation

Syntax

```
CSRmeshResult ConfigSetParameters (CsrUint8 nw_id, CsrUint16 dest_id,
CsrUint8 ttl, CSRMESH_CONFIG_SET_PARAMETERS_T * p_params)
```

Description

This function packs the given parameters into a CSRmesh message and sends it over the network. When a response is received the application callback function is called to notify the CSRMESH_CONFIG_PARAMETERS

Parameters

Parameter	Description
nw_id	Network identifier over which the message has to be sent.
dest_id	16-bit identifier of the destination device/group
ttl	TTL value with which the message needs to be sent.
p_params	Pointer to the message parameters of type CSRMESH_CONFIG_SET_PARAMETERS_T

Returns

CSRmeshResult. Refer to CSRmeshResult.

12.7 ConfigGetParameters Function Documentation

Syntax

```
CSRmeshResult ConfigGetParameters (CsrUint8 nw_id, CsrUint16 dest_id,
CsrUint8 ttl, CSRMESH_CONFIG_GET_PARAMETERS_T * p_params)
```

Description

This function packs the given parameters into a CSRmesh message and sends it over the network. When a response is received the application callback function is called to notify the CSRMESH_CONFIG_PARAMETERS

Parameters

Parameter	Description
nw_id	Network identifier over which the message has to be sent.
dest_id	16-bit identifier of the destination device/group
ttl	TTL value with which the message needs to be sent.
p_params	Pointer to the message parameters of type CSRMESH_CONFIG_GET_PARAMETERS_T

Returns

CSRmeshResult. Refer to CSRmeshResult.

12.8 ConfigDiscoverDevice Function Documentation

Syntax

```
CSRmeshResult ConfigDiscoverDevice (CsrUint8 nw_id, CsrUint16 dest_id,
CsrUint8 ttl, CSRMESH_CONFIG_DISCOVER_DEVICE_T * p_params)
```

Description

This function packs the given parameters into a CSRmesh message and sends it over the network. When a response is received the application callback function is called to notify the CSRMESH_CONFIG_DEVICE_IDENTIFIER

Parameters

Parameter	Description
nw_id	Network identifier over which the message has to be sent.
dest_id	16-bit identifier of the destination device/group
ttl	TTL value with which the message needs to be sent.
p_params	Pointer to the message parameters of type CSRMESH_CONFIG_DISCOVER_DEVICE_T

Returns

CSRmeshResult. Refer to CSRmeshResult.

12.9 ConfigGetInfo Function Documentation

Syntax

```
CSRmeshResult ConfigGetInfo (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8 ttl,
CSRMESH_CONFIG_GET_INFO_T * p_params)
```

Description

This function packs the given parameters into a CSRmesh message and sends it over the network. When a response is received the application callback function is called to notify the CSRMESH_CONFIG_INFO

Parameters

Parameter	Description
nw_id	Network identifier over which the message has to be sent.
dest_id	16-bit identifier of the destination device/group
ttl	TTL value with which the message needs to be sent.
p_params	Pointer to the message parameters of type CSRMESH_CONFIG_GET_INFO_T

Returns

CSRmeshResult. Refer to CSRmeshResult.

12.10 ConfigSetMessageParams Function Documentation

Syntax

```
CSRmeshResult ConfigSetMessageParams (CsrUint8 nw_id, CsrUint16 dest_id,
CsrUint8 ttl, CSRMESH_CONFIG_SET_MESSAGE_PARAMS_T * p_params)
```

Description

Parameters

Parameter	Description
nw_id	Network identifier over which the message has to be sent.
dest_id	16-bit identifier of the destination device/group
ttl	TTL value with which the message needs to be sent.
p_params	Pointer to the message parameters of type CSRMESH_CONFIG_SET_MESSAGE_PARAMS_T

Returns

CSRmeshResult. Refer to CSRmeshResult.

12.11 ConfigGetMessageParams Function Documentation

Syntax

```
CSRmeshResult ConfigGetMessageParams (CsrUint8 nw_id, CsrUint16 dest_id,
CsrUint8 ttl, CSRMESH_CONFIG_GET_MESSAGE_PARAMS_T * p_params)
```

Description

Parameters

Parameter	Description
nw_id	Network identifier over which the message has to be sent.
dest_id	16-bit identifier of the destination device/group
ttl	TTL value with which the message needs to be sent.
p_params	Pointer to the message parameters of type CSRMESH_CONFIG_GET_MESSAGE_PARAMS_T

Returns

CSRmeshResult. Refer to CSRmeshResult.

13 Server

Detailed Description

13.1 Functions of Server

```
CSRmeshResult ConfigModelInit (CsrUint8 nw_id, CsrUint16 *group_id_list,
CsrUint16 num_groups, CSRMESH_MODEL_CALLBACK_T app_callback)
```

Model Initialisation.

```
CSRmeshResult ConfigParameters (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8
ttl, CSRMESH_CONFIG_PARAMETERS_T *p_params)
```

Configuration parameters.

```
CSRmeshResult ConfigDeviceIdentifier (CsrUint8 nw_id, CsrUint16 dest_id,
CsrUint8 ttl, CSRMESH_CONFIG_DEVICE_IDENTIFIER_T *p_params)
```

Device identifier.

```
CSRmeshResult ConfigInfo (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8 ttl,
CSRMESH_CONFIG_INFO_T *p_params)
```

Current device information.

```
CSRmeshResult ConfigMessageParams (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8
ttl, CSRMESH_CONFIG_MESSAGE_PARAMS_T *p_params)
```

This function packs the given parameters into a CSRmesh message and sends it over the network.

13.2 ConfigModelInit Function Documentation

Syntax

```
CSRmeshResult ConfigModelInit (CsrUint8 nw_id, CsrUint16 * group_id_list,
CsrUint16 num_groups, CSRMESH_MODEL_CALLBACK_T app_callback)
```

Description

Registers the model handler with the CSRmesh. Sets the CSRmesh to report num_groups as the maximum number of groups supported for the model

Parameters

Parameter	Description
nw_id	Identifier of the network to which the model has to be registered.
group_id_list	Pointer to a uint16 array to hold assigned group_ids. This must be NULL if no groups are supported
num_groups	Size of the group_id_list. This must be 0 if no groups are supported.
app_callback	Pointer to the application callback function. This function will be called to notify all model specific messages

Returns

CSRmeshResult. Refer to CSRmeshResult.

13.3 ConfigParameters Function Documentation

Syntax

```
CSRmeshResult ConfigParameters (CsrUint8 nw_id, CsrUint16 dest_id,
CsrUint8 ttl, CSRMESH_CONFIG_PARAMETERS_T * p_params)
```

Description

This function packs the given parameters into a CSRmesh message and sends it over the network.

Parameters

Parameter	Description
nw_id	Network identifier over which the message has to be sent.
dest_id	16-bit identifier of the destination device/group
ttl	TTL value with which the message needs to be sent.
p_params	Pointer to the message parameters of type CSRMESH_CONFIG_PARAMETERS_T

Returns

CSRmeshResult. Refer to CSRmeshResult.

13.4 ConfigDeviceIdentifier Function Documentation

Syntax

```
CSRmeshResult ConfigDeviceIdentifier (CsrUint8 nw_id, CsrUint16 dest_id,
CsrUint8 ttl, CSRMESH_CONFIG_DEVICE_IDENTIFIER_T * p_params)
```

Description

This function packs the given parameters into a CSRmesh message and sends it over the network.

Parameters

Parameter	Description
nw_id	Network identifier over which the message has to be sent.
dest_id	16-bit identifier of the destination device/group
ttl	TTL value with which the message needs to be sent.
p_params	Pointer to the message parameters of type CSRMESH_CONFIG_DEVICE_IDENTIFIER_T

Returns

CSRmeshResult. Refer to CSRmeshResult.

13.5 ConfigInfo Function Documentation

Syntax

```
CSRmeshResult ConfigInfo (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8 ttl,
CSRMESH_CONFIG_INFO_T * p_params)
```

Description

This function packs the given parameters into a CSRmesh message and sends it over the network.

Parameters

Parameter	Description
nw_id	Network identifier over which the message has to be sent.
dest_id	16-bit identifier of the destination device/group
ttl	TTL value with which the message needs to be sent.
p_params	Pointer to the message parameters of type CSRMESH_CONFIG_INFO_T

Returns

CSRmeshResult. Refer to CSRmeshResult.

13.6 ConfigMessageParams Function Documentation

Syntax

```
CSRmeshResult ConfigMessageParams (CsrUint8 nw_id, CsrUint16 dest_id,
CsrUint8 ttl, CSRMESH_CONFIG_MESSAGE_PARAMS_T * p_params)
```

Description

Parameters

Parameter	Description
nw_id	Network identifier over which the message has to be sent.
dest_id	16-bit identifier of the destination device/group
ttl	TTL value with which the message needs to be sent.
p_params	Pointer to the message parameters of type CSRMESH_CONFIG_MESSAGE_PARAMS_T

Returns

CSRmeshResult. Refer to CSRmeshResult.

14 Group Model

Detailed Description

Group Model API.

The Group Model configures the set of groups that a device transmits or processes. The groups are configured for a particular model. Each model can have one or more groups associated with; the groups are organized by an index value per model. For example, a device can support 4 group identifiers for one model, and 2 group identifiers for another model. Some models have no specific operation codes. The support of these models (for example the switch model) implies the support of the group model.It is possible to set the group identifiers for all models in a device using the model identifier 0xFF.

The group model has the following states:

- NumberOfGroups
- NumberOfInstances
- GroupIDs

NumberOfGroups is an array of 8-bit unsigned integers, with one entry per model. The value of each entry is the number of group identifiers that the device supports for a given model.

NumberOfInstances is an array of arrays of 8-bit unsigned integers. The top level array holds an array of instances for each model. The second level array holds the NumberOfInstances values for each GroupID that a device exposes in the NumberOfGroups state value<h5>GroupIDsGroupIDs is an array of arrays of 16-bit unsigned integers. The top level array holds an array of instances for each model. The second level array holds the GroupID values for that model, sized at least as big as the NumberOfGroups state value for that model.

14.1 Modules of Group Model

Client

Server

14.2 Data Structures of Group Model

`struct CSRMESH_GROUP_GET_NUMBER_OF_MODEL_GROUPIDS_T`

CSRmesh Group Model message types.

`struct CSRMESH_GROUP_NUMBER_OF_MODEL_GROUPIDS_T`

Get number of groups supported by the model.

`struct CSRMESH_GROUP_SET_MODEL_GROUPID_T`

Setting Model Group ID: Upon receiving a GROUP_SET_MODEL_GROUPID message, where the destination address is the DeviceID of this device, the device saves the Instance and GroupID fields into the appropriate state value determined by the Model and GroupIndex fields. It then responds with a GROUP_MODEL_GROUPID message with the current state information held for the given model and the GroupIndex values.

`struct CSRMESH_GROUP_GET_MODEL_GROUPID_T`

Getting Model Group ID: Upon receiving a GROUP_GET_MODEL_GROUPID message, where the destination address is the DeviceID of this device, the device responds with a GROUP_MODEL_GROUPID message with the current state information held for the given Model and GroupIndex values.

`struct CSRMESH_GROUP_MODEL_GROUPID_T`

GroupID of a model.

15 Client

Detailed Description

15.1 Functions of Client

```
CSRmeshResult GroupModelClientInit (CSRMESH_MODEL_CALLBACK_T app_callback)
```

Initialises Group Model Client functionality.

```
CSRmeshResult GroupGetNumberOfModelGroupids (CsrUint8 nw_id, CsrUint16 dest_id,
CsrUint8 ttl, CSRMESH_GROUP_GET_NUMBER_OF_MODEL_GROUPIDS_T *p_params)
```

Getting Number of Group IDs: Upon receiving a GROUP_GET_NUMBER_OF_MODEL_GROUPS message, where the destination address is the DeviceID of this device, the device responds with a GROUP_NUMBER_OF_MODEL_GROUPS message with the number of Group IDs that the given model supports on this device.

```
CSRmeshResult GroupSetModelGroupid (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8
ttl, CSRMESH_GROUP_SET_MODEL_GROUPID_T *p_params)
```

Setting Model Group ID: Upon receiving a GROUP_SET_MODEL_GROUPID message, where the destination address is the DeviceID of this device, the device saves the Instance and GroupID fields into the appropriate state value determined by the Model and GroupIndex fields. It then responds with a GROUP_MODEL_GROUPID message with the current state information held for the given model and the GroupIndex values.

```
CSRmeshResult GroupGetModelGroupid (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8
ttl, CSRMESH_GROUP_GET_MODEL_GROUPID_T *p_params)
```

Getting Model Group ID: Upon receiving a GROUP_GET_MODEL_GROUPID message, where the destination address is the DeviceID of this device, the device responds with a GROUP_MODEL_GROUPID message with the current state information held for the given Model and GroupIndex values.

15.2 GroupModelClientInit Function Documentation

Syntax

```
CSRmeshResult GroupModelClientInit (CSRMESH_MODEL_CALLBACK_T app_callback)
```

Description

Parameters

Parameter	Description
app_callback	Pointer to the application callback function that will be called when the model client receives a message.

Returns

CSRmeshResult. Refer to CSRmeshResult.

15.3 GroupGetNumberOfModelGroupids Function Documentation

Syntax

```
CSRmeshResult GroupGetNumberOfModelGroupids (CsrUint8 nw_id, CsrUint16 dest_id,
CsrUint8 ttl, CSRMESH_GROUP_GET_NUMBER_OF_MODEL_GROUPIDS_T * p_params)
```

Description

This function packs the given parameters into a CSRmesh message and sends it over the network. When a response is received the application callback function is called to notify the CSRMESH_GROUP_NUMBER_OF_MODEL_GROUPIDS

Parameters

Parameter	Description
nw_id	Network identifier over which the message has to be sent.
dest_id	16-bit identifier of the destination device/group
ttl	TTL value with which the message needs to be sent.
p_params	Pointer to the message parameters of type CSRMESH_GROUP_GET_NUMBER_OF_MODEL_GROUPIDS_T

Returns

CSRmeshResult. Refer to CSRmeshResult.

15.4 GroupSetModelGroupid Function Documentation

Syntax

```
CSRmeshResult GroupSetModelGroupid (CsrUint8 nw_id, CsrUint16 dest_id,
CsrUint8 ttl, CSRMESH_GROUP_SET_MODEL_GROUPID_T * p_params)
```

Description

This function packs the given parameters into a CSRmesh message and sends it over the network. When a response is received the application callback function is called to notify the CSRMESH_GROUP_MODEL_GROUPID

Parameters

Parameter	Description
nw_id	Network identifier over which the message has to be sent.
dest_id	16-bit identifier of the destination device/group
ttl	TTL value with which the message needs to be sent.
p_params	Pointer to the message parameters of type CSRMESH_GROUP_SET_MODEL_GROUPID_T

Returns

CSRmeshResult. Refer to CSRmeshResult.

15.5 GroupGetModelGroupid Function Documentation

Syntax

```
CSRmeshResult GroupGetModelGroupid (CsrUint8 nw_id, CsrUint16 dest_id,
CsrUint8 ttl, CSRMESH_GROUP_GET_MODEL_GROUPID_T * p_params)
```

Description

This function packs the given parameters into a CSRmesh message and sends it over the network. When a response is received the application callback function is called to notify the CSRMESH_GROUP_MODEL_GROUPID

Parameters

Parameter	Description
`nw_id`	Network identifier over which the message has to be sent.
`dest_id`	16-bit identifier of the destination device/group
`ttl`	TTL value with which the message needs to be sent.
`p_params`	Pointer to the message parameters of type CSRMESH_GROUP_GET_MODEL_GROUPID_T

Returns

CSRmeshResult. Refer to CSRmeshResult.

16 Server

Detailed Description

16.1 Functions of Server

```
CSRmeshResult GroupModelInit (CsrUint8 nw_id, CsrUint16 *group_id_list,
CsrUint16 num_groups, CSRMESH_MODEL_CALLBACK_T app_callback)
```

Model Initialisation.

```
CSRmeshResult GroupNumberOfModelGroupids (CsrUint8 nw_id, CsrUint16 dest_id,
CsrUint8 ttl, CSRMESH_GROUP_NUMBER_OF_MODEL_GROUPIDS_T *p_params)
```

Get number of groups supported by the model.

```
CSRmeshResult GroupModelGroupid (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8
ttl, CSRMESH_GROUP_MODEL_GROUPID_T *p_params)
```

GroupID of a model.

16.2 GroupModelInit Function Documentation

Syntax

```
CSRmeshResult GroupModelInit (CsrUint8 nw_id, CsrUint16 * group_id_list,
CsrUint16 num_groups, CSRMESH_MODEL_CALLBACK_T app_callback)
```

Description

Registers the model handler with the CSRmesh. Sets the CSRmesh to report num_groups as the maximum number of groups supported for the model

Parameters

Parameter	Description
`nw_id`	Identifier of the network to which the model has to be registered.
`group_id_list`	Pointer to a uint16 array to hold assigned group_ids. This must be NULL if no groups are supported

Parameter	Description
num_groups	Size of the group_id_list. This must be 0 if no groups are supported.
app_callback	Pointer to the application callback function. This function will be called to notify all model specific messages

Returns

CSRmeshResult. Refer to CSRmeshResult.

16.3 GroupNumberOfModelGroupids Function Documentation

Syntax

```
CSRmeshResult GroupNumberOfModelGroupids (CsrUint8 nw_id, CsrUint16 dest_id,
CsrUint8 ttl, CSRMESH_GROUP_NUMBER_OF_MODEL_GROUPIDS_T * p_params)
```

Description

This function packs the given parameters into a CSRmesh message and sends it over the network.

Parameters

Parameter	Description
nw_id	Network identifier over which the message has to be sent.
dest_id	16-bit identifier of the destination device/group
ttl	TTL value with which the message needs to be sent.
p_params	Pointer to the message parameters of type CSRMESH_GROUP_NUMBER_OF_MODEL_GROUPIDS_T

Returns

CSRmeshResult. Refer to CSRmeshResult.

16.4 GroupModelGroupid Function Documentation

Syntax

```
CSRmeshResult GroupModelGroupid (CsrUint8 nw_id, CsrUint16 dest_id,
CsrUint8 ttl, CSRMESH_GROUP_MODEL_GROUPID_T * p_params)
```

Description

This function packs the given parameters into a CSRmesh message and sends it over the network.

Parameters

Parameter	Description
`nw_id`	Network identifier over which the message has to be sent.
`dest_id`	16-bit identifier of the destination device/group
`ttl`	TTL value with which the message needs to be sent.
`p_params`	Pointer to the message parameters of type CSRMESH_GROUP_MODEL_GROUPID_T

Returns

CSRmeshResult. Refer to CSRmeshResult.

17 Sensor Model

Detailed Description

Sensor Model API.

The Sensor Model broadcasts values to other devices in the CSRmesh network. These values are either based on the physical measurement of something, for example the current air temperature, or based on a desired value for such a physical measurement. These desired values can be used by devices locally so that a feedback loop is created. For example, a fan can receive both the current air temperature and the desired air temperature from other devices and will turn on when the current air temperature is higher than the desired air temperature. The sensor model logically includes two separate types of devices: a source of sensor values, known as a sensor server, and user of sensor values, known as a client. A device may implement both the source and user of sensor values, known as a combined device. A device may also proxy sensor values from other devices even though it does not generate nor act upon those values itself. The sensor model defines three roles:

- Server
- Client
- Proxy

When a source transmits a sensor value in the CSRmesh network, it transmits the value multiple times over an extended period of time. This allows low duty cycle clients and proxies with reasonable proximity to receive the messages. There are two types of sensor values:immediate values and shared state values. Immediate values typically represent a current physical measurement. While multiple devices in a single room or group may measure the same physical measurement, their different values are significant because the differences convey additional information. Shared state values typically represent an intention or desire. If there are multiple devices within a room or group that need to know these shared state values, they should all work together to use the same value, distributing these values between each other until every device has the same shared state value. Sensor values are always sent within the context of a group. It is the group that has the sensor value, not the individual devices. For example, the desired air temperature may be known by multiple devices, a window, a vent, a fan, an extractor fan, a heater, an air-conditioning unit, or a light; sending a request to read the current desired air temperature can elicit a response from any of these devices. This allows fast replacement or repurposing of equipment by configuring just the group codes for devices. Devices may act as a proxy for sensor data. For example, a device that is able to listen all the time may listen for sensor messages and store the latest values for each type of sensor data for one or more groups. As stated above, the sensor values are associated with a group of devices and proxies do not have to understand the meaning or types of these sensor values or which devices generate these values. The proxies only need to know that the sensor values are associated with a given group. This allows any device to easily help reduce the power consumption of other devices within a room by taking the burden of remembering sensor values and listening to requests and responding with these values, or updating other devices with the latest shared state values. This can be done by any device. A device that needs to know the current state of some sensor data may request the current state of the

value from any other device in the group. A device may also send out a message that it is missing the current state of a value. These messages can highlight missing functionality at the group or network levels. This specification defines a set of sensor types that describe the format and meaning of the sensor values. Multiple sensor types can be sent in a single message, although there are limits to the practical number of sensor values that can fit into a single message. Sensor types based on physical measurements are always in SI units. No derived units are used. For example, temperature is measured as an integer multiple of 0.03125 kelvin. This implies that the resultant temperature must be converted to represent the temperature using the Celsius or Fahrenheit temperature scales. For example, 0 degrees Centigrade would be represented by the value 8741. This also implies that the Celsius temperature can represent accurately a temperature down to 0.1 degrees Centigrade and also a range of values from -273.1 degrees Centigrade to 1774.8 degrees Centigrade. The Fahrenheit scale can also represent temperatures down to 0.1 degrees Fahrenheit accuracy with a range from -459.67 degrees Fahrenheit to 3226.7 degrees Fahrenheit. This enables all devices to be sold worldwide to work with each other and display the temperature in whatever the local unit or user preference is. Additional examples of SI unit representations include: speed measured in centimetres per second, air pressure in hecto-Pascals. Some values such as compass direction are compressed to a single octet representation, yet are still measured in radians. For example, each increment in the value increases the direction by 2pi/256 radians. Some sensor types are based on bit-fields or enumerated values. To allow future extensions of these types a very simple extensibility protocol is used: If a bit is not defined in this specification, it should be transmitted as a zero and ignored upon receipt; if an enumerated value is not defined in this specification, it should not be transmitted and should be ignored upon receipt. A device may support zero or more sensor types. For a given sensor type, there can be only one instance of this sensor type in a device. The set of supported types can therefore be considered to be a set of types. Given that a typical device may only support a very small number of types and these types are sparsely distributed, the method to discover the types supported by a device should be by returning the types supported and not as a bit-field or similar representation. The exact representation of sensor values and the assigned numbers are documented in the mcp.json file included with this specification.

The sensor model has the following states:

- Types
- Shared State Value

The types that a device supports are a property of a device that must be able to be discovered by a configuring device. A device may support zero or more types; there is no limit to the number of types that a device supports except imposed by the number of defined types. The types supported should be considered an ordered set of types. The types are enumerated using a 16-bit unsigned integer. The meaning of these types is defined in the associated mcp.json file.

The shared state value is a variable length value that a device should remember so that the state of the shared state sensor type can be distributed to other devices in the network.

When a device is writing a new sensor value to another device or a set of devices identified by a group, the duty cycle of the destination device(s) may restrict the probability that the message is received. The client should therefore retransmit the SENSOR_WRITE_VALUE and SENSOR_WRITE_VALUE_NO_ACK messages multiple times over a period of time to increase the probability of the message being received by all devices. If SENSOR_WRITE_VALUE is used, then the retransmission of messages may be terminated early if a SENSOR_VALUE message is received from all devices from which such a message is expected.

The value in the SENSOR_VALUE message may be different from the value in the SENSOR_WRITE_VALUE message because another device may also be writing this value at the same time; the value in the SENSOR_VALUE message is always the latest value from the sending device.

For each sensor type that a device supports and for which the period is a non-zero time, if the sensor type is an immediate sensor type, then the device should send SENSOR_VALUE every period of time, with the value in this message as the latest measured value. If a device is expecting a value to be written to a sensor type and no value is written within a reasonable period of time, the sensor can send a SENSOR_MISSING message. This message can be used by a sensor client to start sending sensor values to this device.

17.1 Modules of Sensor Model

Client

Server

17.2 Data Structures of Sensor Model

`struct CSRMESH_SENSOR_GET_TYPES_T`

CSRmesh Sensor Model message types.

`struct CSRMESH_SENSOR_TYPES_T`

Sensor types.

`struct CSRMESH_SENSOR_SET_STATE_T`

Setting Sensor State: Upon receiving a SENSOR_SET_STATE message, where the destination address is the device ID of this device and the Type field is a supported sensor type, the device saves the RxDutyCycle field and responds with a SENSOR_STATE message with the current state information of the sensor type. If the Type field is not a supported sensor type, the device ignores the message.

`struct CSRMESH_SENSOR_GET_STATE_T`

Getting Sensor State: Upon receiving a SENSOR_GET_STATE message, where the destination address is the deviceID of this device and the Type field is a supported sensor type, the device shall respond with a SENSOR_STATE message with the current state information of the sensor type. Upon receiving a SENSOR_GET_STATE message, where the destination address is the device ID of this device but the Type field is not a supported sensor type, the device shall ignore the message.

`struct CSRMESH_SENSOR_STATE_T`

Current sensor state.

`struct CSRMESH_SENSOR_WRITE_VALUE_T`

Writing Sensor Value: Upon receiving a SENSOR_WRITE_VALUE message, where the Type field is a supported sensor type, the device saves the value into the current value of the sensor type on this device and responds with a SENSOR_VALUE message with the current value of this sensor type.

`struct CSRMESH_SENSOR_READ_VALUE_T`

Getting Sensor Value: Upon receiving a SENSOR_READ_VALUE message, where the Type field is a supported sensor type, the device responds with a SENSOR_VALUE message with the value of the sensor type. Proxy Behaviour: Upon receiving a SENSOR_GET_STATE where the destination of the message and the sensor type correspond to a previously received SENSOR_BROADCAST_VALUE or SENSOR_BROADCAST_NEW message, the device responds with a SENSOR_VALUE message with the remembered values.

```
struct  CSRMESH_SENSOR_VALUE_T
```

Current sensor value.

```
struct  CSRMESH_SENSOR_MISSING_T
```

Sensor data is missing. Proxy Behaviour: Upon receiving a SENSOR_MISSING message, the proxy determines if it has remembered this type and value and then writes that type and value to the device that sent the message.

17.3 Typedefs of Sensor Model

```
typedef uint16 SENSOR_FORMAT_TEMPERATURE_T
```

CSRmesh Temperature format.

17.4 Enumerations of Sensor Model

```
enum  sensor_type_t {
  sensor_type_invalid = 0,
sensor_type_internal_air_temperature = 1,
sensor_type_external_air_temperature = 2,
sensor_type_desired_air_temperature = 3,
  sensor_type_internal_humidity = 4,
sensor_type_external_humidity = 5,
sensor_type_external_dewpoint = 6,
sensor_type_internal_door = 7,
  sensor_type_external_door = 8,
sensor_type_internal_window = 9,
sensor_type_external_window = 10,
sensor_type_solar_energy = 11,
  sensor_type_number_of_activations = 12,
sensor_type_fridge_temperature = 13,
sensor_type_desired_fridge_temperature = 14,
sensor_type_freezer_temperature = 15,
  sensor_type_desired_freezer_temperature = 16,
sensor_type_oven_temperature = 17,
sensor_type_desired_oven_temperature = 18,
sensor_type_seat_occupied = 19,
  sensor_type_washing_machine_state = 20,
sensor_type_dish_washer_state = 21,
sensor_type_clothes_dryer_state = 22,
sensor_type_toaster_state = 23,
  sensor_type_carbon_dioxide = 24,
sensor_type_carbon_monoxide = 25,
sensor_type_smoke = 26,
sensor_type_water_level = 27,
  sensor_type_hot_water_temperature = 28,
sensor_type_cold_water_temperature = 29,
sensor_type_desired_water_temperature = 30,
sensor_type_cooker_hob_back_left_state = 31,
  sensor_type_desired_cooker_hob_back_left_state = 32,
sensor_type_cooker_hob_front_left_state = 33,
sensor_type_desired_cooker_hob_front_left_state = 34,
sensor_type_cooker_hob_back_middle_state = 35,
  sensor_type_desired_cooker_hob_back_middle_state = 36,
sensor_type_cooker_hob_front_middle_state = 37,
sensor_type_desired_cooker_hob_front_middle_state = 38,
sensor_type_cooker_hob_back_right_state = 39,
  sensor_type_desired_cooker_hob_back_right_state = 40,
sensor_type_cooker_hob_front_right_state = 41,
sensor_type_desired_cooker_hob_front_right_state = 42,
sensor_type_desired_wakeup_alarm_time = 43,
  sensor_type_desired_second_wakeup_alarm_time = 44,
sensor_type_passive_infrared_state = 45,
sensor_type_water_flowing = 46,
sensor_type_desired_water_flow = 47,
```

```
  sensor_type_audio_level = 48,
sensor_type_desired_audio_level = 49,
sensor_type_fan_speed = 50,
sensor_type_desired_fan_speed = 51,
  sensor_type_wind_speed = 52,
sensor_type_wind_speed_gust = 53,
sensor_type_wind_direction = 54,
sensor_type_wind_direction_gust = 55,
  sensor_type_rain_fall_last_hour = 56,
sensor_type_rain_fall_today = 57,
sensor_type_barometric_pressure = 58,
sensor_type_soil_temperature = 59,
  sensor_type_soil_moisture = 60,
sensor_type_window_cover_position = 61,
sensor_type_desired_window_cover_position = 62
}
```

CSRmesh Sensor Type.

17.5 sensor_type_t Enumeration Type Documentation

Syntax

```
enum sensor_type_t
```

Description

Enumerations

Enumeration	Description
`sensor_type_invalid`	type_undefined
`sensor_type_internal_air_temperature`	temperature_kelvin
`sensor_type_external_air_temperature`	temperature_kelvin_range
`sensor_type_desired_air_temperature`	temperature_kelvin
`sensor_type_internal_humidity`	percentage
`sensor_type_external_humidity`	percentage

Enumeration	Description
sensor_type_external_dewpoint	temperature_kelvin
sensor_type_internal_door	door_state
sensor_type_external_door	door_state
sensor_type_internal_window	window_state
sensor_type_external_window	window_state
sensor_type_solar_energy	watts_per_square_metre
sensor_type_number_of_activations	16_bit_counter
sensor_type_fridge_temperature	temperature_kelvin
sensor_type_desired_fridge_temperature	temperature_kelvin
sensor_type_freezer_temperature	temperature_kelvin
sensor_type_desired_freezer_temperature	temperature_kelvin
sensor_type_oven_temperature	temperature_kelvin
sensor_type_desired_oven_temperature	temperature_kelvin
sensor_type_seat_occupied	seat_state
sensor_type_washing_machine_state	appliance_state
sensor_type_dish_washer_state	appliance_state
sensor_type_clothes_dryer_state	appliance_state
sensor_type_toaster_state	appliance_state
sensor_type_carbon_dioxide	parts_per_million

Enumeration	Description
sensor_type_carbon_monoxide	parts_per_million
sensor_type_smoke	micrograms_per_cubic_metre
sensor_type_water_level	percentage
sensor_type_hot_water_temperature	temperature_kelvin
sensor_type_cold_water_temperature	temperature_kelvin
sensor_type_desired_water_temperature	temperature_kelvin
sensor_type_cooker_hob_back_left_state	cooker_hob_state
sensor_type_desired_cooker_hob_back_left_state	cooker_hob_state
sensor_type_cooker_hob_front_left_state	cooker_hob_state
sensor_type_desired_cooker_hob_front_left_state	cooker_hob_state
sensor_type_cooker_hob_back_middle_state	cooker_hob_state
sensor_type_desired_cooker_hob_back_middle_state	cooker_hob_state
sensor_type_cooker_hob_front_middle_state	cooker_hob_state
sensor_type_desired_cooker_hob_front_middle_state	cooker_hob_state
sensor_type_cooker_hob_back_right_state	cooker_hob_state
sensor_type_desired_cooker_hob_back_right_state	cooker_hob_state

Enumeration	Description
sensor_type_cooker_hob_front_right_state	cooker_hob_state
sensor_type_desired_cooker_hob_front_right_state	cooker_hob_state
sensor_type_desired_wakeup_alarm_time	minutes_of_the_day
sensor_type_desired_second_wakeup_alarm_time	minutes_of_the_day
sensor_type_passive_infrared_state	movement_state
sensor_type_water_flowing	water_flow_rate
sensor_type_desired_water_flow	water_flow_rate
sensor_type_audio_level	decibel
sensor_type_desired_audio_level	decibel
sensor_type_fan_speed	percentage
sensor_type_desired_fan_speed	forward_backward
sensor_type_wind_speed	centimetres_per_second
sensor_type_wind_speed_gust	centimetres_per_second
sensor_type_wind_direction	direction
sensor_type_wind_direction_gust	direction
sensor_type_rain_fall_last_hour	millimetres
sensor_type_rain_fall_today	millimetres
sensor_type_barometric_pressure	air_pressure

Enumeration	Description
sensor_type_soil_temperature	temperature_kelvin
sensor_type_soil_moisture	percentage
sensor_type_window_cover_position	percentage
sensor_type_desired_window_cover_position	percentage

18 Client

Detailed Description

18.1 Functions of Client

```
CSRmeshResult SensorModelClientInit (CSRMESH_MODEL_CALLBACK_T app_callback)
```

Initialises Sensor Model Client functionality.

```
CSRmeshResult SensorGetTypes (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8 ttl,
CSRMESH_SENSOR_GET_TYPES_T *p_params)
```

Upon receiving a SENSOR_GET_TYPES message, the device responds with a SENSOR_TYPES message with the list of supported types greater than or equal to the FirstType field. If the device does not support any types greater than or equal to the FirstType field, then it sends a SENSOR_TYPES message with zero-length Types field.

```
CSRmeshResult SensorSetState (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8 ttl,
CSRMESH_SENSOR_SET_STATE_T *p_params)
```

Setting Sensor State: Upon receiving a SENSOR_SET_STATE message, where the destination address is the device ID of this device and the Type field is a supported sensor type, the device saves the RxDutyCycle field and responds with a SENSOR_STATE message with the current state information of the sensor type. If the Type field is not a supported sensor type, the device ignores the message.

```
CSRmeshResult SensorGetState (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8 ttl,
CSRMESH_SENSOR_GET_STATE_T *p_params)
```

Getting Sensor State: Upon receiving a SENSOR_GET_STATE message, where the destination address is the deviceID of this device and the Type field is a supported sensor type, the device shall respond with a SENSOR_STATE message with the current state information of the sensor type. Upon receiving a SENSOR_GET_STATE message, where the destination address is the device ID of this device but the Type field is not a supported sensor type, the device shall ignore the message.

```
CSRmeshResult SensorWriteValue (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8
ttl, CSRMESH_SENSOR_WRITE_VALUE_T *p_params, bool request_ack)
```

Writing Sensor Value: Upon receiving a SENSOR_WRITE_VALUE message, where the Type field is a supported sensor type, the device saves the value into the current value of the sensor type on this device and responds with a SENSOR_VALUE message with the current value of this sensor type.

```
CSRmeshResult SensorReadValue (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8 ttl,
CSRMESH_SENSOR_READ_VALUE_T *p_params)
```

Getting Sensor Value: Upon receiving a SENSOR_READ_VALUE message, where the Type field is a supported sensor type, the device responds with a SENSOR_VALUE message with the value of the sensor type. Proxy Behaviour: Upon receiving a SENSOR_GET_STATE where the destination of the message and the sensor type correspond to a previously received SENSOR_BROADCAST_VALUE or SENSOR_BROADCAST_NEW message, the device responds with a SENSOR_VALUE message with the remembered values.

18.2 SensorModelClientInit Function Documentation

Syntax

```
CSRmeshResult SensorModelClientInit (CSRMESH_MODEL_CALLBACK_T app_callback)
```

Description

Parameters

Parameter	Description
app_callback	Pointer to the application callback function that will be called when the model client receives a message.

Returns

CSRmeshResult. Refer to CSRmeshResult.

18.3 SensorGetTypes Function Documentation

Syntax

```
CSRmeshResult SensorGetTypes (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8 ttl,
CSRMESH_SENSOR_GET_TYPES_T * p_params)
```

Description

This function packs the given parameters into a CSRmesh message and sends it over the network. When a response is received the application callback function is called to notify the CSRMESH_SENSOR_TYPES

Parameters

Parameter	Description
nw_id	Network identifier over which the message has to be sent.
dest_id	16-bit identifier of the destination device/group
ttl	TTL value with which the message needs to be sent.
p_params	Pointer to the message parameters of type CSRMESH_SENSOR_GET_TYPES_T

Returns

CSRmeshResult. Refer to CSRmeshResult.

18.4 SensorSetState Function Documentation

Syntax

```
CSRmeshResult SensorSetState (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8 ttl,
CSRMESH_SENSOR_SET_STATE_T * p_params)
```

Description

This function packs the given parameters into a CSRmesh message and sends it over the network. When a response is received the application callback function is called to notify the CSRMESH_SENSOR_STATE

Parameters

Parameter	Description
nw_id	Network identifier over which the message has to be sent.
dest_id	16-bit identifier of the destination device/group
ttl	TTL value with which the message needs to be sent.
p_params	Pointer to the message parameters of type CSRMESH_SENSOR_SET_STATE_T

Returns

CSRmeshResult. Refer to CSRmeshResult.

18.5 SensorGetState Function Documentation

Syntax

```
CSRmeshResult SensorGetState (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8 ttl,
CSRMESH_SENSOR_GET_STATE_T * p_params)
```

Description

This function packs the given parameters into a CSRmesh message and sends it over the network. When a response is received the application callback function is called to notify the CSRMESH_SENSOR_STATE

Parameters

Parameter	Description
nw_id	Network identifier over which the message has to be sent.
dest_id	16-bit identifier of the destination device/group
ttl	TTL value with which the message needs to be sent.
p_params	Pointer to the message parameters of type CSRMESH_SENSOR_GET_STATE_T

Returns

CSRmeshResult. Refer to CSRmeshResult.

18.6 SensorWriteValue Function Documentation

Syntax

```
CSRmeshResult SensorWriteValue (CsrUint8 nw_id, CsrUint16 dest_id,
CsrUint8 ttl, CSRMESH_SENSOR_WRITE_VALUE_T * p_params, bool request_ack)
```

Description

This function packs the given parameters into a CSRmesh message and sends it over the network. When a response is received the application callback function is called to notify the CSRMESH_SENSOR_VALUE

Parameters

Parameter	Description
nw_id	Network identifier over which the message has to be sent.
dest_id	16-bit identifier of the destination device/group
ttl	TTL value with which the message needs to be sent.
p_params	Pointer to the message parameters of type CSRMESH_SENSOR_WRITE_VALUE_T
request_ack	TRUE if an acknowledgement is required

Returns

CSRmeshResult. Refer to CSRmeshResult.

18.7 SensorReadValue Function Documentation

Syntax

```
CSRmeshResult SensorReadValue (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8 ttl,
CSRMESH_SENSOR_READ_VALUE_T * p_params)
```

Description

This function packs the given parameters into a CSRmesh message and sends it over the network. When a response is received the application callback function is called to notify the CSRMESH_SENSOR_VALUE

Parameters

Parameter	Description
nw_id	Network identifier over which the message has to be sent.
dest_id	16-bit identifier of the destination device/group
ttl	TTL value with which the message needs to be sent.
p_params	Pointer to the message parameters of type CSRMESH_SENSOR_READ_VALUE_T

Returns

CSRmeshResult. Refer to CSRmeshResult.

19 Server

Detailed Description

19.1 Functions of Server

```
CSRmeshResult SensorModelInit (CsrUint8 nw_id, CsrUint16 *group_id_list,
CsrUint16 num_groups, CSRMESH_MODEL_CALLBACK_T app_callback)
```

Model Initialisation.

```
CSRmeshResult SensorTypes (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8 ttl,
CSRMESH_SENSOR_TYPES_T *p_params)
```

Sensor types.

```
CSRmeshResult SensorState (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8 ttl,
CSRMESH_SENSOR_STATE_T *p_params)
```

Current sensor state.

```
CSRmeshResult SensorValue (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8 ttl,
CSRMESH_SENSOR_VALUE_T *p_params)
```

Current sensor value.

```
CSRmeshResult SensorMissing (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8 ttl,
CSRMESH_SENSOR_MISSING_T *p_params)
```

Sensor data is missing. Proxy Behaviour: Upon receiving a SENSOR_MISSING message, the proxy determines if it has remembered this type and value and then writes that type and value to the device that sent the message.

19.2 SensorModelInit Function Documentation

Syntax

```
CSRmeshResult SensorModelInit (CsrUint8 nw_id, CsrUint16 * group_id_list,
CsrUint16 num_groups, CSRMESH_MODEL_CALLBACK_T app_callback)
```

Description

Registers the model handler with the CSRmesh. Sets the CSRmesh to report num_groups as the maximum number of groups supported for the model

Parameters

Parameter	Description
nw_id	Identifier of the network to which the model has to be registered.
group_id_list	Pointer to a uint16 array to hold assigned group_ids. This must be NULL if no groups are supported
num_groups	Size of the group_id_list. This must be 0 if no groups are supported.
app_callback	Pointer to the application callback function. This function will be called to notify all model specific messages

Returns

CSRmeshResult. Refer to CSRmeshResult.

19.3 SensorTypes Function Documentation

Syntax

```
CSRmeshResult SensorTypes (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8 ttl,
CSRMESH_SENSOR_TYPES_T * p_params)
```

Description

This function packs the given parameters into a CSRmesh message and sends it over the network.

Parameters

Parameter	Description
nw_id	Network identifier over which the message has to be sent.
dest_id	16-bit identifier of the destination device/group
ttl	TTL value with which the message needs to be sent.
p_params	Pointer to the message parameters of type CSRMESH_SENSOR_TYPES_T

Returns

CSRmeshResult. Refer to CSRmeshResult.

19.4 SensorState Function Documentation

Syntax

```
CSRmeshResult SensorState (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8 ttl,
CSRMESH_SENSOR_STATE_T * p_params)
```

Description

This function packs the given parameters into a CSRmesh message and sends it over the network.

Parameters

Parameter	Description
nw_id	Network identifier over which the message has to be sent.
dest_id	16-bit identifier of the destination device/group
ttl	TTL value with which the message needs to be sent.
p_params	Pointer to the message parameters of type CSRMESH_SENSOR_STATE_T

Returns

CSRmeshResult. Refer to CSRmeshResult.

19.5 SensorValue Function Documentation

Syntax

```
CSRmeshResult SensorValue (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8 ttl,
CSRMESH_SENSOR_VALUE_T * p_params)
```

Description

This function packs the given parameters into a CSRmesh message and sends it over the network.

Parameters

Parameter	Description
nw_id	Network identifier over which the message has to be sent.
dest_id	16-bit identifier of the destination device/group
ttl	TTL value with which the message needs to be sent.
p_params	Pointer to the message parameters of type CSRMESH_SENSOR_VALUE_T

Returns

CSRmeshResult. Refer to CSRmeshResult.

19.6 SensorMissing Function Documentation

Syntax

```
CSRmeshResult SensorMissing (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8 ttl,
CSRMESH_SENSOR_MISSING_T * p_params)
```

Description

This function packs the given parameters into a CSRmesh message and sends it over the network.

Parameters

Parameter	Description
nw_id	Network identifier over which the message has to be sent.
dest_id	16-bit identifier of the destination device/group
ttl	TTL value with which the message needs to be sent.
p_params	Pointer to the message parameters of type CSRMESH_SENSOR_MISSING_T

Returns

CSRmeshResult. Refer to CSRmeshResult.

20 Actuator Model

Detailed Description

Actuator Model API.

The Actuator Model sets values on other devices in the CSRmesh network. These values are either based on the physical property of something, for example the current air temperature, or based on an abstract value for such a physical property. There is one type of actuator values: control values. Control values are values that are immediately processed by a device and cannot be read. If the value should be readable, then the value would be better represented as a sensor desired value. This specification defines a set of actuator types that describe the format and meaning of the actuator values, logically from the same namespace as the sensor types. Actuator types based around physical measurements are always in SI units. No derived units are used. For example, temperature is measured as an integer multiple of 0.03125 kelvin. This implies that the resultant temperature must be converted to represent the temperature using the Celsius or Fahrenheit temperature scales. For example, 0 degrees Centigrade would be represented by the value 8741. This also implies that the Celsius temperature scale can represent accurately a temperature down to 0.1 degrees Centigrade and also a range of values from -273.1 degrees Centigrade to 1774.8 degrees Centigrade. The Fahrenheit scale can also represent temperatures down to 0.1 degrees Fahrenheit accuracy with a range from -459.67 degrees Fahrenheit to 3226.7 degrees Fahrenheit. This enables all devices to be sold worldwide to work with each other and display the temperature in whatever the local unit or user preference is. Additional examples of SI unit representations include speed measured in centimetres per second and air pressure in hecto-Pascals. Some values such as compass direction are compressed to a single octet representation, yet are still measured in radians. For example, each increment in the value increases the direction by 2pi/256 radians. Some actuator types are based on bit-fields or enumerated values. To allow for future extensions of these types a very simple extensibility protocol is used: if a bit is not defined in this specification, it shall be transmitted as a zero and ignored upon receipt; if an enumerated value is not defined in this specification, it shall not be transmitted and shall be ignored upon receipt. A device may support zero or more actuator types. For a given actuator type, there can be only one instance of this actuator type in a device. Therefore the collection of types supported is sufficient to describe the behaviour of a specific actuator and the number of instance is not needed. Given that a typical device may only support a very small number of types and these types are sparsely distributed, the method to discover the types supported by a device is by returning the types supported instead of using a bit-field or similar representation.

The actuator model has the following state:

- Types

The types that a device supports are a property of the device that must be able to be discovered by a configuring device. A device may support one or more types; there is no limit to the number of types that a device supports except that imposed by the number of defined types. The types supported should be considered as an ordered set of types.

20.1 Modules of Actuator Model

Client

Server

20.2 Data Structures of Actuator Model

`struct CSRMESH_ACTUATOR_GET_TYPES_T`

CSRmesh Actuator Model message types.

`struct CSRMESH_ACTUATOR_TYPES_T`

Actuator types.

`struct CSRMESH_ACTUATOR_SET_VALUE_T`

Get sensor state. Upon receiving an ACTUATOR_SET_VALUE_NO_ACK message, where the destination address is the device ID of this device and the Type field is a supported actuator type, the device shall immediately use the Value field for the given Type field. The meaning of this actuator value is not defined in this specification. Upon receiving an ACTUATOR_SET_VALUE_NO_ACK message, where the destination address is is the device ID of this device but the Type field is not a supported actuator type, the device shall ignore the message.

`struct CSRMESH_ACTUATOR_VALUE_ACK_T`

Current sensor state.

`struct CSRMESH_ACTUATOR_GET_VALUE_ACK_T`

Get Current sensor state.

21 Client

Detailed Description

21.1 Functions of Client

```
CSRmeshResult ActuatorModelClientInit (CSRMESH_MODEL_CALLBACK_T app_callback)
```

Initialises Actuator Model Client functionality.

```
CSRmeshResult ActuatorGetTypes (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8
ttl, CSRMESH_ACTUATOR_GET_TYPES_T *p_params)
```

Upon receiving an ACTUATOR_GET_TYPES message, the device responds with an ACTUATOR_TYPES message with a list of supported types greater than or equal to the FirstType field. If the device does not support any types greater than or equal to FirstType, it sends an ACTUATOR_TYPES message with a zero length Types field.

```
CSRmeshResult ActuatorSetValue (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8
ttl, CSRMESH_ACTUATOR_SET_VALUE_T *p_params, bool request_ack)
```

Get sensor state. Upon receiving an ACTUATOR_SET_VALUE_NO_ACK message, where the destination address is the device ID of this device and the Type field is a supported actuator type, the device shall immediately use the Value field for the given Type field. The meaning of this actuator value is not defined in this specification. Upon receiving an ACTUATOR_SET_VALUE_NO_ACK message, where the destination address is is the device ID of this device but the Type field is not a supported actuator type, the device shall ignore the message.

```
CSRmeshResult ActuatorGetValueAck (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8
ttl, CSRMESH_ACTUATOR_GET_VALUE_ACK_T *p_params)
```

Get Current sensor state.

21.2 ActuatorModelClientInit Function Documentation

Syntax

```
CSRmeshResult ActuatorModelClientInit (CSRMESH_MODEL_CALLBACK_T app_callback)
```

Description

Parameters

Parameter	Description
app_callback	Pointer to the application callback function that will be called when the model client receives a message.

Returns

CSRmeshResult. Refer to CSRmeshResult.

21.3 ActuatorGetTypes Function Documentation

Syntax

```
CSRmeshResult ActuatorGetTypes (CsrUint8 nw_id, CsrUint16 dest_id,
CsrUint8 ttl, CSRMESH_ACTUATOR_GET_TYPES_T * p_params)
```

Description

This function packs the given parameters into a CSRmesh message and sends it over the network. When a response is received the application callback function is called to notify the CSRMESH_ACTUATOR_TYPES

Parameters

Parameter	Description
nw_id	Network identifier over which the message has to be sent.
dest_id	16-bit identifier of the destination device/group
ttl	TTL value with which the message needs to be sent.
p_params	Pointer to the message parameters of type CSRMESH_ACTUATOR_GET_TYPES_T

Returns

CSRmeshResult. Refer to CSRmeshResult.

21.4 ActuatorSetValue Function Documentation

Syntax

```
CSRmeshResult ActuatorSetValue (CsrUint8 nw_id, CsrUint16 dest_id,
CsrUint8 ttl, CSRMESH_ACTUATOR_SET_VALUE_T * p_params, bool request_ack)
```

Description

This function packs the given parameters into a CSRmesh message and sends it over the network. When a response is received the application callback function is called to notify the CSRMESH_ACTUATOR_VALUE_ACK

Parameters

Parameter	Description
nw_id	Network identifier over which the message has to be sent.
dest_id	16-bit identifier of the destination device/group
ttl	TTL value with which the message needs to be sent.
p_params	Pointer to the message parameters of type CSRMESH_ACTUATOR_SET_VALUE_T
request_ack	TRUE if an acknowledgement is required

Returns

CSRmeshResult. Refer to CSRmeshResult.

21.5 ActuatorGetValueAck Function Documentation

Syntax

```
CSRmeshResult ActuatorGetValueAck (CsrUint8 nw_id, CsrUint16 dest_id,
CsrUint8 ttl, CSRMESH_ACTUATOR_GET_VALUE_ACK_T * p_params)
```

Description

This function packs the given parameters into a CSRmesh message and sends it over the network. When a response is received the application callback function is called to notify the CSRMESH_ACTUATOR_VALUE_ACK

Parameters

Parameter	Description
`nw_id`	Network identifier over which the message has to be sent.
`dest_id`	16-bit identifier of the destination device/group
`ttl`	TTL value with which the message needs to be sent.
`p_params`	Pointer to the message parameters of type CSRMESH_ACTUATOR_GET_VALUE_ACK_T

Returns

CSRmeshResult. Refer to CSRmeshResult.

22 Server

Detailed Description

22.1 Functions of Server

```
CSRmeshResult ActuatorModelInit (CsrUint8 nw_id, CsrUint16 *group_id_list,
CsrUint16 num_groups, CSRMESH_MODEL_CALLBACK_T app_callback)
```

Model Initialisation.

```
CSRmeshResult ActuatorTypes (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8 ttl,
CSRMESH_ACTUATOR_TYPES_T *p_params)
```

Actuator types.

```
CSRmeshResult ActuatorValueAck (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8
ttl, CSRMESH_ACTUATOR_VALUE_ACK_T *p_params)
```

Current sensor state.

22.2 ActuatorModelInit Function Documentation

Syntax

```
CSRmeshResult ActuatorModelInit (CsrUint8 nw_id, CsrUint16 * group_id_list,
CsrUint16 num_groups, CSRMESH_MODEL_CALLBACK_T app_callback)
```

Description

Registers the model handler with the CSRmesh. Sets the CSRmesh to report num_groups as the maximum number of groups supported for the model

Parameters

Parameter	Description
`nw_id`	Identifier of the network to which the model has to be registered.
`group_id_list`	Pointer to a uint16 array to hold assigned group_ids. This must be NULL if no groups are supported

Parameter	Description
num_groups	Size of the group_id_list. This must be 0 if no groups are supported.
app_callback	Pointer to the application callback function. This function will be called to notify all model specific messages

Returns

CSRmeshResult. Refer to CSRmeshResult.

22.3 ActuatorTypes Function Documentation

Syntax

```
CSRmeshResult ActuatorTypes (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8 ttl,
CSRMESH_ACTUATOR_TYPES_T * p_params)
```

Description

This function packs the given parameters into a CSRmesh message and sends it over the network.

Parameters

Parameter	Description
nw_id	Network identifier over which the message has to be sent.
dest_id	16-bit identifier of the destination device/group
ttl	TTL value with which the message needs to be sent.
p_params	Pointer to the message parameters of type CSRMESH_ACTUATOR_TYPES_T

Returns

CSRmeshResult. Refer to CSRmeshResult.

22.4 ActuatorValueAck Function Documentation

Syntax

```
CSRmeshResult ActuatorValueAck (CsrUint8 nw_id, CsrUint16 dest_id,
CsrUint8 ttl, CSRMESH_ACTUATOR_VALUE_ACK_T * p_params)
```

Description

This function packs the given parameters into a CSRmesh message and sends it over the network.

Parameters

Parameter	Description
`nw_id`	Network identifier over which the message has to be sent.
`dest_id`	16-bit identifier of the destination device/group
`ttl`	TTL value with which the message needs to be sent.
`p_params`	Pointer to the message parameters of type CSRMESH_ACTUATOR_VALUE_ACK_T

Returns

CSRmeshResult. Refer to CSRmeshResult.

23 Data Model

Detailed Description

Data Model API.

The Data Model transmits either a stream or blocks of arbitrary data between two devices. For stream data, each octet within the stream has an octet number. This octet number provides the reliable delivery of the stream data as a sequence number. For block data, there is no reliable delivery of messages.

The data model has the following state:

- StreamSequenceNumber

StreamSequenceNumber is an unsigned 16-bit integer measured in octets. This value determines the octet number for the next message received.

Before sending a DATA_STREAM_SEND message, the client should send a DATA_STREAM_FLUSH message to set the current sequence number on the server to a known value. When sending a number of octets, the client should split these octets into the minimum number of DATA_STREAM_SEND messages, and send each message in sequence with the appropriate sequence number. The client then awaits the DATA_STREAM_RECEIVED messages to determine if the data has been successfully received and acknowledged. If a DATA_STREAM_RECEIVED message is received and it has a next expected sequence number within the sent octets, then the DATA_STREAM_SEND message containing that octet number should be sent again. If no DATA_STREAM_RECEIVED messages are received, then the client should resend the DATA_STREAM_SEND message until the device does acknowledge receipt. The client may timeout the reliable delivery of these data stream octets. It should then send a DATA_STREAM_FLUSH message before sending additional stream octets to the server. The length of the time for the timeout is an implementation detail.

The timing of the DATA_STREAM_FLUSH message can be immediately after the timeout occurs, or it can be just before the delivery of the next sequence of octets. This is an implementation detail.

23.1 Modules of Data Model

Client

Server

23.2 Data Structures of Data Model

`struct CSRMESH_DATA_STREAM_FLUSH_T`

CSRmesh Data Model message types.

`struct CSRMESH_DATA_STREAM_SEND_T`

Sending Data: Upon receiving a DATA_STREAM_SEND message, the device first checks if the StreamSN field is the same as the StreamSequenceNumber model state. If these values are the same, the device passes the StreamOctets field up to the application for processing, and increments StreamSequenceNumber by the length of the StreamOctets field. It then responds with a DATA_STREAM_RECEIVED message with the current value of the StreamSequenceNumber. Note: The DATA_STREAM_RECEIVED message is sent even if the StreamSN received is different from the StreamSequenceNumber state. This allows missing packets to be detected and retransmitted by the sending device.

`struct CSRMESH_DATA_STREAM_RECEIVED_T`

Acknowledgement of data received.

`struct CSRMESH_DATA_BLOCK_SEND_T`

A block of data, no acknowledgement. Upon receiving a DATA_BLOCK_SEND message, the device passes the DatagramOctets field up to the application for processing.

24 Client

Detailed Description

24.1 Functions of Client

```
CSRmeshResult DataModelClientInit (CSRMESH_MODEL_CALLBACK_T app_callback)
```

Initialises Data Model Client functionality.

```
CSRmeshResult DataStreamFlush (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8 ttl,
CSRMESH_DATA_STREAM_FLUSH_T *p_params)
```

Flushing Data: Upon receiving a DATA_STREAM_FLUSH message, the device saves the StreamSN field into the StreamSequenceNumber model state and responds with DATA_STREAM_RECEIVED with the StreamNESN field set to the value of the StreamSequenceNumber model state. The device also flushes all partially received stream data from this peer device.

```
CSRmeshResult DataStreamSend (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8 ttl,
CSRMESH_DATA_STREAM_SEND_T *p_params)
```

Sending Data: Upon receiving a DATA_STREAM_SEND message, the device first checks if the StreamSN field is the same as the StreamSequenceNumber model state. If these values are the same, the device passes the StreamOctets field up to the application for processing, and increments StreamSequenceNumber by the length of the StreamOctets field. It then responds with a DATA_STREAM_RECEIVED message with the current value of the StreamSequenceNumber. Note: The DATA_STREAM_RECEIVED message is sent even if the StreamSN received is different from the StreamSequenceNumber state. This allows missing packets to be detected and retransmitted by the sending device.

```
CSRmeshResult DataBlockSend (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8 ttl,
CSRMESH_DATA_BLOCK_SEND_T *p_params)
```

A block of data, no acknowledgement. Upon receiving a DATA_BLOCK_SEND message, the device passes the DatagramOctets field up to the application for processing.

24.2 DataModelClientInit Function Documentation

Syntax

```
CSRmeshResult DataModelClientInit (CSRMESH_MODEL_CALLBACK_T app_callback)
```

Description

Parameters

Parameter	Description
app_callback	Pointer to the application callback function that will be called when the model client receives a message.

Returns

CSRmeshResult. Refer to CSRmeshResult.

24.3 DataStreamFlush Function Documentation

Syntax

```
CSRmeshResult DataStreamFlush (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8 ttl,
CSRMESH_DATA_STREAM_FLUSH_T * p_params)
```

Description

This function packs the given parameters into a CSRmesh message and sends it over the network. When a response is received the application callback function is called to notify the CSRMESH_DATA_STREAM_RECEIVED

Parameters

Parameter	Description
nw_id	Network identifier over which the message has to be sent.
dest_id	16-bit identifier of the destination device/group
ttl	TTL value with which the message needs to be sent.
p_params	Pointer to the message parameters of type CSRMESH_DATA_STREAM_FLUSH_T

Returns

CSRmeshResult. Refer to CSRmeshResult.

24.4 DataStreamSend Function Documentation

Syntax

```
CSRmeshResult DataStreamSend (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8 ttl,
CSRMESH_DATA_STREAM_SEND_T * p_params)
```

Description

This function packs the given parameters into a CSRmesh message and sends it over the network. When a response is received the application callback function is called to notify the CSRMESH_DATA_STREAM_RECEIVED

Parameters

Parameter	Description
nw_id	Network identifier over which the message has to be sent.
dest_id	16-bit identifier of the destination device/group
ttl	TTL value with which the message needs to be sent.
p_params	Pointer to the message parameters of type CSRMESH_DATA_STREAM_SEND_T

Returns

CSRmeshResult. Refer to CSRmeshResult.

24.5 DataBlockSend Function Documentation

Syntax

```
CSRmeshResult DataBlockSend (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8 ttl,
CSRMESH_DATA_BLOCK_SEND_T * p_params)
```

Description

This function packs the given parameters into a CSRmesh message and sends it over the network.

Parameters

Parameter	Description
`nw_id`	Network identifier over which the message has to be sent.
`dest_id`	16-bit identifier of the destination device/group
`ttl`	TTL value with which the message needs to be sent.
`p_params`	Pointer to the message parameters of type CSRMESH_DATA_BLOCK_SEND_T

Returns

CSRmeshResult. Refer to CSRmeshResult.

25 Server

Detailed Description

25.1 Functions of Server

```
CSRmeshResult DataModelInit (CsrUint8 nw_id, CsrUint16 *group_id_list,
CsrUint16 num_groups, CSRMESH_MODEL_CALLBACK_T app_callback)
```

Model Initialisation.

```
CSRmeshResult DataStreamReceived (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8
ttl, CSRMESH_DATA_STREAM_RECEIVED_T *p_params)
```

Acknowledgement of data received.

25.2 DataModelInit Function Documentation

Syntax

```
CSRmeshResult DataModelInit (CsrUint8 nw_id, CsrUint16 * group_id_list,
CsrUint16 num_groups, CSRMESH_MODEL_CALLBACK_T app_callback)
```

Description

Registers the model handler with the CSRmesh. Sets the CSRmesh to report num_groups as the maximum number of groups supported for the model

Parameters

Parameter	Description
`nw_id`	Identifier of the network to which the model has to be registered.
`group_id_list`	Pointer to a uint16 array to hold assigned group_ids. This must be NULL if no groups are supported

Parameter	Description
num_groups	Size of the group_id_list. This must be 0 if no groups are supported.
app_callback	Pointer to the application callback function. This function will be called to notify all model specific messages

Returns

CSRmeshResult. Refer to CSRmeshResult.

25.3 DataStreamReceived Function Documentation

Syntax

```
CSRmeshResult DataStreamReceived (CsrUint8 nw_id, CsrUint16 dest_id,
CsrUint8 ttl, CSRMESH_DATA_STREAM_RECEIVED_T * p_params)
```

Description

This function packs the given parameters into a CSRmesh message and sends it over the network.

Parameters

Parameter	Description
nw_id	Network identifier over which the message has to be sent.
dest_id	16-bit identifier of the destination device/group
ttl	TTL value with which the message needs to be sent.
p_params	Pointer to the message parameters of type CSRMESH_DATA_STREAM_RECEIVED_T

Returns

CSRmeshResult. Refer to CSRmeshResult.

26 Bearer Model

Detailed Description

Bearer Model API.

The Bearer Model controls which bearers are handled by a given device and which bearers are used to relay MTL messages. A device can be set in a promiscuous mode where the device forwards all CSRmesh packets, regardless of whether the network key is known or not. This mode enables the examination of network traffic in busy locations with many overlapping networks and permits simpler deployments so that multiple installations can use their common physical deployment to support their individual traffic.

The Bearer has the following states:

- BearerRelayActive
- BearerEnabled
- Promiscous

BearerRelayActive is a 16-bit bit field. Each bit determines if the given bearer is used to relay MTL packets received on any bearer.

BearerRelayActive does not determine if a message received on a given bearer is relayed, but which bearers relay a message received on any active bearer. The following bits are defined:

- BearerRelayActive
- LE Advertising Bearer=0
- LE GATT Server Bearer=1

BearerEnabled is a 16-bit bit field. Each bit determines if the given bearer is enabled. Enabled in this context means that the bearer can receive mesh messages, and/or the bearer can be used to transmit mesh messages originated by this device.The following bits are defined:

- LE Advertising Bearer=0
- LE GATT Server Bearer=1

BearerPromiscous is a 16-bit bit field. Each bit determines if the given bearer promiscuity is enabled. Enabled in this context means that the bearer can relay all traffic it receives even if the bearer does not match any known network keys. The following bits are defined:

- LE Advertising Bearer=0
- LE GATT Server Bearer=1

26.1 Modules of Bearer Model

Client

Server

26.2 Data Structures of Bearer Model

`struct CSRMESH_BEARER_SET_STATE_T`

CSRmesh Bearer Model message types.

`struct CSRMESH_BEARER_GET_STATE_T`

Getting Bearer State: Upon receiving a BEARER_GET_STATE message, where the destination address is the device ID of this device, the device responds with a BEARER_STATE message with the current state information.

`struct CSRMESH_BEARER_STATE_T`

Set bearer state.

27 Client

Detailed Description

27.1 Functions of Client

```
CSRmeshResult BearerModelClientInit (CSRMESH_MODEL_CALLBACK_T app_callback)
```

Initialises Bearer Model Client functionality.

```
CSRmeshResult BearerSetState (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8 ttl,
CSRMESH_BEARER_SET_STATE_T *p_params)
```

Setting Bearer State: Upon receiving a BEARER_SET_STATE message, where the destination address is the device ID of this device, the device saves the BearerRelayActive, BearerEnabled, and BearerPromiscuous fields into the appropriate state value. Then the device responds with a BEARER_STATE message with the current state information.

```
CSRmeshResult BearerGetState (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8 ttl,
CSRMESH_BEARER_GET_STATE_T *p_params)
```

Getting Bearer State: Upon receiving a BEARER_GET_STATE message, where the destination address is the device ID of this device, the device responds with a BEARER_STATE message with the current state information.

27.2 BearerModelClientInit Function Documentation

Syntax

```
CSRmeshResult BearerModelClientInit (CSRMESH_MODEL_CALLBACK_T app_callback)
```

Description

Parameters

Parameter	Description
app_callback	Pointer to the application callback function that will be called when the model client receives a message.

Returns

CSRmeshResult. Refer to CSRmeshResult.

27.3 BearerSetState Function Documentation

Syntax

```
CSRmeshResult BearerSetState (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8 ttl,
CSRMESH_BEARER_SET_STATE_T * p_params)
```

Description

This function packs the given parameters into a CSRmesh message and sends it over the network. When a response is received the application callback function is called to notify the CSRMESH_BEARER_STATE

Parameters

Parameter	Description
nw_id	Network identifier over which the message has to be sent.
dest_id	16-bit identifier of the destination device/group
ttl	TTL value with which the message needs to be sent.
p_params	Pointer to the message parameters of type CSRMESH_BEARER_SET_STATE_T

Returns

CSRmeshResult. Refer to CSRmeshResult.

27.4 BearerGetState Function Documentation

Syntax

```
CSRmeshResult BearerGetState (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8 ttl,
CSRMESH_BEARER_GET_STATE_T * p_params)
```

Description

This function packs the given parameters into a CSRmesh message and sends it over the network. When a response is received the application callback function is called to notify the CSRMESH_BEARER_STATE

Parameters

Parameter	Description
`nw_id`	Network identifier over which the message has to be sent.
`dest_id`	16-bit identifier of the destination device/group
`ttl`	TTL value with which the message needs to be sent.
`p_params`	Pointer to the message parameters of type CSRMESH_BEARER_GET_STATE_T

Returns

CSRmeshResult. Refer to CSRmeshResult.

28 Server

Detailed Description

28.1 Functions of Server

```
CSRmeshResult BearerModelInit (CsrUint8 nw_id, CsrUint16 *group_id_list,
CsrUint16 num_groups, CSRMESH_MODEL_CALLBACK_T app_callback)
```

Model Initialisation.

```
CSRmeshResult BearerState (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8 ttl,
CSRMESH_BEARER_STATE_T *p_params)
```

Set bearer state.

28.2 BearerModelInit Function Documentation

Syntax

```
CSRmeshResult BearerModelInit (CsrUint8 nw_id, CsrUint16 * group_id_list,
CsrUint16 num_groups, CSRMESH_MODEL_CALLBACK_T app_callback)
```

Description

Registers the model handler with the CSRmesh. Sets the CSRmesh to report num_groups as the maximum number of groups supported for the model

Parameters

Parameter	Description
`nw_id`	Identifier of the network to which the model has to be registered.
`group_id_list`	Pointer to a uint16 array to hold assigned group_ids. This must be NULL if no groups are supported

Parameter	Description
num_groups	Size of the group_id_list. This must be 0 if no groups are supported.
app_callback	Pointer to the application callback function. This function will be called to notify all model specific messages

Returns

CSRmeshResult. Refer to CSRmeshResult.

28.3 BearerState Function Documentation

Syntax

```
CSRmeshResult BearerState (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8 ttl,
CSRMESH_BEARER_STATE_T * p_params)
```

Description

This function packs the given parameters into a CSRmesh message and sends it over the network.

Parameters

Parameter	Description
nw_id	Network identifier over which the message has to be sent.
dest_id	16-bit identifier of the destination device/group
ttl	TTL value with which the message needs to be sent.
p_params	Pointer to the message parameters of type CSRMESH_BEARER_STATE_T

Returns

CSRmeshResult. Refer to CSRmeshResult.

29 Ping Model

Detailed Description

Ping Model API.

The Ping Model ensures that the MTL can provide reliable message delivery to a peer device. It does this by allowing a transmitting device to send a request that a device supporting the ping model will respond to. The ping model optimises the configuration of devices within the network. The mechanisms to optimise the network are out of scope of this specification.

29.1 Modules of Ping Model

Client

Server

29.2 Data Structures of Ping Model

`struct CSRMESH_PING_REQUEST_T`

CSRmesh Ping Model message types.

`struct CSRMESH_PING_RESPONSE_T`

Ping response.

30 Client

Detailed Description

30.1 Functions of Client

```
CSRmeshResult PingModelClientInit (CSRMESH_MODEL_CALLBACK_T app_callback)
```

Initialises Ping Model Client functionality.

```
CSRmeshResult PingRequest (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8 ttl,
CSRMESH_PING_REQUEST_T *p_params)
```

Ping Request: Upon receiving a PING_REQUEST message, the device responds with a PING_RESPONSE message with the TTLAtRx field set to the TTL value from the PING_REQUEST message, and the RSSIAtRx field set to the RSSI value of the PING_REQUEST message. If the bearer used to receive the PING_REQUEST message does not have an RSSI value, then the value 0x00 is used.

30.2 PingModelClientInit Function Documentation

Syntax

```
CSRmeshResult PingModelClientInit (CSRMESH_MODEL_CALLBACK_T app_callback)
```

Description

Parameters

Parameter	Description
app_callback	Pointer to the application callback function that will be called when the model client receives a message.

Returns

CSRmeshResult. Refer to CSRmeshResult.

30.3 PingRequest Function Documentation

Syntax

```
CSRmeshResult PingRequest (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8 ttl,
CSRMESH_PING_REQUEST_T * p_params)
```

Description

This function packs the given parameters into a CSRmesh message and sends it over the network. When a response is received the application callback function is called to notify the CSRMESH_PING_RESPONSE

Parameters

Parameter	Description
nw_id	Network identifier over which the message has to be sent.
dest_id	16-bit identifier of the destination device/group
ttl	TTL value with which the message needs to be sent.
p_params	Pointer to the message parameters of type CSRMESH_PING_REQUEST_T

Returns

CSRmeshResult. Refer to CSRmeshResult.

31 Server

Detailed Description

31.1 Functions of Server

```
CSRmeshResult PingModelInit (CsrUint8 nw_id, CsrUint16 *group_id_list,
CsrUint16 num_groups, CSRMESH_MODEL_CALLBACK_T app_callback)
```

Model Initialisation.

```
CSRmeshResult PingResponse (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8 ttl,
CSRMESH_PING_RESPONSE_T *p_params)
```

Ping response.

31.2 PingModelInit Function Documentation

Syntax

```
CSRmeshResult PingModelInit (CsrUint8 nw_id, CsrUint16 * group_id_list,
CsrUint16 num_groups, CSRMESH_MODEL_CALLBACK_T app_callback)
```

Description

Registers the model handler with the CSRmesh. Sets the CSRmesh to report num_groups as the maximum number of groups supported for the model

Parameters

Parameter	Description
nw_id	Identifier of the network to which the model has to be registered.
group_id_list	Pointer to a uint16 array to hold assigned group_ids. This must be NULL if no groups are supported

Parameter	Description
num_groups	Size of the group_id_list. This must be 0 if no groups are supported.
app_callback	Pointer to the application callback function. This function will be called to notify all model specific messages

Returns

CSRmeshResult. Refer to CSRmeshResult.

31.3 PingResponse Function Documentation

Syntax

```
CSRmeshResult PingResponse (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8 ttl,
CSRMESH_PING_RESPONSE_T * p_params)
```

Description

This function packs the given parameters into a CSRmesh message and sends it over the network.

Parameters

Parameter	Description
nw_id	Network identifier over which the message has to be sent.
dest_id	16-bit identifier of the destination device/group
ttl	TTL value with which the message needs to be sent.
p_params	Pointer to the message parameters of type CSRMESH_PING_RESPONSE_T

Returns

CSRmeshResult. Refer to CSRmeshResult.

32 Battery Model

Detailed Description

Battery Model API.

The Battery Model reports the current battery state of a device that is powered by a battery.

The battery model has the following states:

- BatteryLevel
- BatteryState

BatteryLevel is an unsigned 8-bit value as a percentage. The values 0x00 to 0x64 are valid, and all other values are reserved for future use. The value 0x00 represents a battery that has no energy capacity. The value 0x64 represents a battery with full capacity. Values between these two extremes represent a linear scaling from empty to full capacity.

BatteryState is an 8-bit bit field. Each bit is a Boolean value that determines if a given state is true or false. The value of zero is false, and the value of one is true. The following bits are defined:

- Battery is powering device=0
- Battery is charging=1
- Device is externally powered=2
- Service is required for battery=3
- Battery needs replacement=4

All other bits are reserved.

The device transmits a new BATTERY_STATE message with the current state information when the BatteryLevel changes significantly or the BatteryState changes. The TID field can be set to any value.

The definition of changing significantly is left to the implementation. It may mean when the battery level changes by 1% or 10% or 50% or some other value.

32.1 Modules of Battery Model

Client

Server

32.2 Data Structures of Battery Model

`struct CSRMESH_BATTERY_GET_STATE_T`

CSRmesh Battery Model message types.

`struct CSRMESH_BATTERY_STATE_T`

Current battery state.

33 Client

Detailed Description

33.1 Functions of Client

```
CSRmeshResult BatteryModelClientInit (CSRMESH_MODEL_CALLBACK_T app_callback)
```

Initialises Battery Model Client functionality.

```
CSRmeshResult BatteryGetState (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8 ttl,
CSRMESH_BATTERY_GET_STATE_T *p_params)
```

Getting Battery State: Upon receiving a BATTERY_GET_STATE message, the device responds with a BATTERY_STATE message with the current state information.

33.2 BatteryModelClientInit Function Documentation

Syntax

```
CSRmeshResult BatteryModelClientInit (CSRMESH_MODEL_CALLBACK_T app_callback)
```

Description

Parameters

Parameter	Description
app_callback	Pointer to the application callback function that will be called when the model client receives a message.

Returns

CSRmeshResult. Refer to CSRmeshResult.

33.3 BatteryGetState Function Documentation

Syntax

```
CSRmeshResult BatteryGetState (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8 ttl,
CSRMESH_BATTERY_GET_STATE_T * p_params)
```

Description

This function packs the given parameters into a CSRmesh message and sends it over the network. When a response is received the application callback function is called to notify the CSRMESH_BATTERY_STATE

Parameters

Parameter	Description
nw_id	Network identifier over which the message has to be sent.
dest_id	16-bit identifier of the destination device/group
ttl	TTL value with which the message needs to be sent.
p_params	Pointer to the message parameters of type CSRMESH_BATTERY_GET_STATE_T

Returns

CSRmeshResult. Refer to CSRmeshResult.

34 Server

Detailed Description

34.1 Functions of Server

```
CSRmeshResult BatteryModelInit (CsrUint8 nw_id, CsrUint16 *group_id_list,
CsrUint16 num_groups, CSRMESH_MODEL_CALLBACK_T app_callback)
```

Model Initialisation.

```
CSRmeshResult BatteryState (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8 ttl,
CSRMESH_BATTERY_STATE_T *p_params)
```

Current battery state.

34.2 BatteryModelInit Function Documentation

Syntax

```
CSRmeshResult BatteryModelInit (CsrUint8 nw_id, CsrUint16 * group_id_list,
CsrUint16 num_groups, CSRMESH_MODEL_CALLBACK_T app_callback)
```

Description

Registers the model handler with the CSRmesh. Sets the CSRmesh to report num_groups as the maximum number of groups supported for the model

Parameters

Parameter	Description
`nw_id`	Identifier of the network to which the model has to be registered.
`group_id_list`	Pointer to a uint16 array to hold assigned group_ids. This must be NULL if no groups are supported

Parameter	Description
num_groups	Size of the group_id_list. This must be 0 if no groups are supported.
app_callback	Pointer to the application callback function. This function will be called to notify all model specific messages

Returns

CSRmeshResult. Refer to CSRmeshResult.

34.3 BatteryState Function Documentation

Syntax

```
CSRmeshResult BatteryState (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8 ttl,
CSRMESH_BATTERY_STATE_T * p_params)
```

Description

This function packs the given parameters into a CSRmesh message and sends it over the network.

Parameters

Parameter	Description
nw_id	Network identifier over which the message has to be sent.
dest_id	16-bit identifier of the destination device/group
ttl	TTL value with which the message needs to be sent.
p_params	Pointer to the message parameters of type CSRMESH_BATTERY_STATE_T

Returns

CSRmeshResult. Refer to CSRmeshResult.

35 Attention Model

Detailed Description

Attention Model API.

It is sometimes useful to identify devices that have been installed, especially when they need maintenance. The attention model allows a device to make itself noticeable so that a human can help pinpoint the physical device.

The definition of being noticeable is an implementation detail and may vary from one device to another. For example, a light may flash its light, another device may make a sound, and another may operate a motor.

The attention model has the following states:

- AttractAttention
- AttentionDuration

AttractAttention is a boolean value held in an 8-bit field. A value of 0x00 means that the device is not attracting attention. A value of 0x01 means that the device is attracting attention. Attracting attention in this context means that the device may flash lights, make a sound, operate a motor, or something similar that allows a user to determine which device is attracting attention. All other values are reserved for future use.

AttentionDuration is a 16-bit unsigned integer in milliseconds. This value determines how long a device continues to attract attention. If this value is set to 0xFFFF, then the device attracts attention continuously until it is set not to attract attention.

If a device has been attracting attention for the time set by AttentionDuration, when the attention timer expires, the device will stop attracting attention and may send an ATTENTION_STATE message with the current state information. The TID field can be set to any value.

35.1 Modules of Attention Model

Client

Server

35.2 Data Structures of Attention Model

`struct CSRMESH_ATTENTION_SET_STATE_T`

CSRmesh Attention Model message types.

`struct CSRMESH_ATTENTION_STATE_T`

Current battery state.

36 Client

Detailed Description

36.1 Functions of Client

```
CSRmeshResult AttentionModelClientInit (CSRMESH_MODEL_CALLBACK_T app_callback)
```

Initialises Attention Model Client functionality.

```
CSRmeshResult AttentionSetState (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8
ttl, CSRMESH_ATTENTION_SET_STATE_T *p_params)
```

Setting Flashing State: Upon receiving an ATTENTION_SET_STATE message, the device saves the AttractAttention and AttentionDuration fields into the appropriate state value. It then responds with an ATTENTION_STATE message with the current state information. If the AttractAttention field is set to 0x01 and the AttentionDuration is not 0xFFFF, then any existing attention timer is cancelled and a new attention timer is started that will expire after AttentionDuration milliseconds. If the AttractAttention field is set to 0x01 and the AttentionDuration field is 0xFFFF, then the attention timer is ignored. If the AttractAttention field is set to 0x00, then the attention timer is cancelled if it is already running.

36.2 AttentionModelClientInit Function Documentation

Syntax

```
CSRmeshResult AttentionModelClientInit (CSRMESH_MODEL_CALLBACK_T app_callback)
```

Description

Parameters

Parameter	Description
app_callback	Pointer to the application callback function that will be called when the model client receives a message.

Returns

CSRmeshResult. Refer to CSRmeshResult.

36.3 AttentionSetState Function Documentation

Syntax

```
CSRmeshResult AttentionSetState (CsrUint8 nw_id, CsrUint16 dest_id,
CsrUint8 ttl, CSRMESH_ATTENTION_SET_STATE_T * p_params)
```

Description

This function packs the given parameters into a CSRmesh message and sends it over the network. When a response is received the application callback function is called to notify the CSRMESH_ATTENTION_STATE

Parameters

Parameter	Description
nw_id	Network identifier over which the message has to be sent.
dest_id	16-bit identifier of the destination device/group
ttl	TTL value with which the message needs to be sent.
p_params	Pointer to the message parameters of type CSRMESH_ATTENTION_SET_STATE_T

Returns

CSRmeshResult. Refer to CSRmeshResult.

37 Server

Detailed Description

37.1 Functions of Server

```
CSRmeshResult AttentionModelInit (CsrUint8 nw_id, CsrUint16 *group_id_list,
CsrUint16 num_groups, CSRMESH_MODEL_CALLBACK_T app_callback)
```

Model Initialisation.

```
CSRmeshResult AttentionState (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8 ttl,
CSRMESH_ATTENTION_STATE_T *p_params)
```

Current battery state.

37.2 AttentionModelInit Function Documentation

Syntax

```
CSRmeshResult AttentionModelInit (CsrUint8 nw_id, CsrUint16 * group_id_list,
CsrUint16 num_groups, CSRMESH_MODEL_CALLBACK_T app_callback)
```

Description

Registers the model handler with the CSRmesh. Sets the CSRmesh to report num_groups as the maximum number of groups supported for the model

Parameters

Parameter	Description
nw_id	Identifier of the network to which the model has to be registered.
group_id_list	Pointer to a uint16 array to hold assigned group_ids. This must be NULL if no groups are supported

Parameter	Description
num_groups	Size of the group_id_list. This must be 0 if no groups are supported.
app_callback	Pointer to the application callback function. This function will be called to notify all model specific messages

Returns

CSRmeshResult. Refer to CSRmeshResult.

37.3 AttentionState Function Documentation

Syntax

```
CSRmeshResult AttentionState (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8 ttl,
CSRMESH_ATTENTION_STATE_T * p_params)
```

Description

This function packs the given parameters into a CSRmesh message and sends it over the network.

Parameters

Parameter	Description
nw_id	Network identifier over which the message has to be sent.
dest_id	16-bit identifier of the destination device/group
ttl	TTL value with which the message needs to be sent.
p_params	Pointer to the message parameters of type CSRMESH_ATTENTION_STATE_T

Returns

CSRmeshResult. Refer to CSRmeshResult.

38 Power Model

Detailed Description

Power Model API.

The Power Model controls the turning on and off of power for a device. A device may have multiple power states: on, off, and optionally standby. In addition to these states, a device may also remember if it is on standby before it is turned on so that it returns to standby mode after it is toggled to other power states. The device power can be commanded directly to a given state, or the power state can be toggled to an opposite state. This means when a device is on, it will be toggled to off; when a device is off, it will be toggled to on. If a device supports the standby state, then the device will be toggled from standby to on from standby, and will be toggled from on from standby to standby. The power model can be used in an individual device and at a wall power socket. A wall power socket may have multiple outlets, each of which can be individually switchable. To allow each power socket to be individually switched, each socket is assigned a separate GroupID.

The power model has the following state:

- PowerState

PowerState is an enumerated value held in an 8-bit value. The following values are defined:

- Off=0x00. The device is powered off.
- On=0x01. The device is powered on.
- Standby=0x02. The device appears to be powered off, but is in a standby state so that it can turn on quicker than if it is off.
- OnFromStandby=0x03. The device is powered on from the standby state.

The device may be interacted with directly, for example somebody presses a physical power switch on a device. When the power state changes locally on a device, it may send a POWER_STATE_NO_ACK message with the current state information.

38.1 Modules of Power Model

Client

Server

38.2 Data Structures of Power Model

`struct CSRMESH_POWER_SET_STATE_T`

CSRmesh Power Model message types.

`struct CSRMESH_POWER_TOGGLE_STATE_T`

Toggling Power State: Upon receiving a POWER_Toggle_STATE_NO_ACK message, the device sets the PowerState state value as defined: 1.If the current PowerState is 0x00, Off, then PowerState should be set to 0x01, On. 2.If the current PowerState is 0x01, On, then PowerState should be set to 0x00, Off. 3.If the current PowerState is 0x02, Standby, then PowerState should be set to 0x03, OnFromStandby. 4.If the current PowerState is 0x03, OnFromStandby, then PowerState should be set to 0x02, Standby. Then the device responds with a POWER_STATE message with the current state information.

`struct CSRMESH_POWER_GET_STATE_T`

Getting Power State: Upon receiving a POWER_GET_STATE message, the device responds with a POWER_STATE message with the current state information.

`struct CSRMESH_POWER_STATE_T`

Current power state.

39 Client

Detailed Description

39.1 Functions of Client

```
CSRmeshResult PowerModelClientInit (CSRMESH_MODEL_CALLBACK_T app_callback)
```

Initialises Power Model Client functionality.

```
CSRmeshResult PowerSetState (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8 ttl,
CSRMESH_POWER_SET_STATE_T *p_params, bool request_ack)
```

Setting Power State: Upon receiving a POWER_SET_STATE_NO_ACK message, the device sets the PowerState state value to the PowerState field. It then responds with a POWER_STATE message with the current state information.

```
CSRmeshResult PowerToggleState (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8
ttl, CSRMESH_POWER_TOGGLE_STATE_T *p_params, bool request_ack)
```

Toggling Power State: Upon receiving a POWER_Toggle_STATE_NO_ACK message, the device sets the PowerState state value as defined: 1.If the current PowerState is 0x00, Off, then PowerState should be set to 0x01, On. 2.If the current PowerState is 0x01, On, then PowerState should be set to 0x00, Off. 3.If the current PowerState is 0x02, Standby, then PowerState should be set to 0x03, OnFromStandby. 4.If the current PowerState is 0x03, OnFromStandby, then PowerState should be set to 0x02, Standby. Then the device responds with a POWER_STATE message with the current state information.

```
CSRmeshResult PowerGetState (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8 ttl,
CSRMESH_POWER_GET_STATE_T *p_params)
```

Getting Power State: Upon receiving a POWER_GET_STATE message, the device responds with a POWER_STATE message with the current state information.

39.2 PowerModelClientInit Function Documentation

Syntax

```
CSRmeshResult PowerModelClientInit (CSRMESH_MODEL_CALLBACK_T app_callback)
```

Description

Parameters

Parameter	Description
app_callback	Pointer to the application callback function that will be called when the model client receives a message.

Returns

CSRmeshResult. Refer to CSRmeshResult.

39.3 PowerSetState Function Documentation

Syntax

```
CSRmeshResult PowerSetState (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8 ttl,
CSRMESH_POWER_SET_STATE_T * p_params, bool request_ack)
```

Description

This function packs the given parameters into a CSRmesh message and sends it over the network. When a response is received the application callback function is called to notify the CSRMESH_POWER_STATE

Parameters

Parameter	Description
nw_id	Network identifier over which the message has to be sent.
dest_id	16-bit identifier of the destination device/group
ttl	TTL value with which the message needs to be sent.
p_params	Pointer to the message parameters of type CSRMESH_POWER_SET_STATE_T
request_ack	TRUE if an acknowledgement is required

Returns

CSRmeshResult. Refer to CSRmeshResult.

39.4 PowerToggleState Function Documentation

Syntax

```
CSRmeshResult PowerToggleState (CsrUint8 nw_id, CsrUint16 dest_id,
CsrUint8 ttl, CSRMESH_POWER_TOGGLE_STATE_T * p_params, bool request_ack)
```

Description

This function packs the given parameters into a CSRmesh message and sends it over the network. When a response is received the application callback function is called to notify the CSRMESH_POWER_STATE

Parameters

Parameter	Description
nw_id	Network identifier over which the message has to be sent.
dest_id	16-bit identifier of the destination device/group
ttl	TTL value with which the message needs to be sent.
p_params	Pointer to the message parameters of type CSRMESH_POWER_TOGGLE_STATE_T
request_ack	TRUE if an acknowledgement is required

Returns

CSRmeshResult. Refer to CSRmeshResult.

39.5 PowerGetState Function Documentation

Syntax

```
CSRmeshResult PowerGetState (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8 ttl,
CSRMESH_POWER_GET_STATE_T * p_params)
```

Description

This function packs the given parameters into a CSRmesh message and sends it over the network. When a response is received the application callback function is called to notify the CSRMESH_POWER_STATE

Parameters

Parameter	Description
`nw_id`	Network identifier over which the message has to be sent.
`dest_id`	16-bit identifier of the destination device/group
`ttl`	TTL value with which the message needs to be sent.
`p_params`	Pointer to the message parameters of type CSRMESH_POWER_GET_STATE_T

Returns

CSRmeshResult. Refer to CSRmeshResult.

40 Server

Detailed Description

40.1 Functions of Server

```
CSRmeshResult PowerModelInit (CsrUint8 nw_id, CsrUint16 *group_id_list,
CsrUint16 num_groups, CSRMESH_MODEL_CALLBACK_T app_callback)
```

Model Initialisation.

```
CSRmeshResult PowerState (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8 ttl,
CSRMESH_POWER_STATE_T *p_params, bool request_ack)
```

Current power state.

40.2 PowerModelInit Function Documentation

Syntax

```
CSRmeshResult PowerModelInit (CsrUint8 nw_id, CsrUint16 * group_id_list,
CsrUint16 num_groups, CSRMESH_MODEL_CALLBACK_T app_callback)
```

Description

Registers the model handler with the CSRmesh. Sets the CSRmesh to report num_groups as the maximum number of groups supported for the model

Parameters

Parameter	Description
`nw_id`	Identifier of the network to which the model has to be registered.
`group_id_list`	Pointer to a uint16 array to hold assigned group_ids. This must be NULL if no groups are supported

Parameter	Description
num_groups	Size of the group_id_list. This must be 0 if no groups are supported.
app_callback	Pointer to the application callback function. This function will be called to notify all model specific messages

Returns

CSRmeshResult. Refer to CSRmeshResult.

40.3 PowerState Function Documentation

Syntax

```
CSRmeshResult PowerState (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8 ttl,
CSRMESH_POWER_STATE_T * p_params, bool request_ack)
```

Description

This function packs the given parameters into a CSRmesh message and sends it over the network.

Parameters

Parameter	Description
nw_id	Network identifier over which the message has to be sent.
dest_id	16-bit identifier of the destination device/group
ttl	TTL value with which the message needs to be sent.
p_params	Pointer to the message parameters of type CSRMESH_POWER_STATE_T
request_ack	TRUE if an acknowledgement is required

Returns

CSRmeshResult. Refer to CSRmeshResult.

41 Light Model

Detailed Description

Light Model API.

The Light Model controls devices that typically have dimmable and/or coloured lights. It is also possible to control the power of the lights and have lights change their level and colour slowly. The light model also allows the fine tuning of the colour temperature of the light over time. For example, the light can be set to be bluer in the morning and slowly becomes yellower by the time of the afternoon. Finally, the light model allows the intensity of the light to change based on the received signal strength of certain assets.

The light model has the following states:

- CurrentLevel
- DeltaLevel
- RedLevel
- GreenLevel
- BlueLevel
- DeltaRedLevel
- DeltaGreenLevel
- DeltaBlueLevel
- TargetLevel
- TargetRed
- TargetGreen
- TargetBlue
- LevelSDState
- LevelDuration
- LevelSustain
- LevelDecay
- ColorTemperature
- DeltaColorTemperature
- TargetColorTemperature

CurrentLevel is an unsigned integer representing values from 0x00 to 0xFF. The value 0x00 represents fully off, and the value 0xFF represents fully on. Values between these two extremes represent a linear change in observed brightness from fully off to fully on. For example, the value 0x80 represents half observed brightness.

DeltaLevel represents the desired change in CurrentLevel to TargetLevel over a given period of time.

LevelSDState is an enumeration that represents the following values:

- Idle
- Attacking
- Sustaining
- Decaying

If CurrentLevel is different from TargetLevel, then CurrentLevel should be changed by DeltaLevel. If CurrentLevel reaches TargetLevel, then a LIGHT_STATE_NO_ACK message may be sent. If CurrentRed is different from TargetRed, then CurrentRed should be changed by DeltaRed. If CurrentRed reaches TargetRed, then a LIGHT_STATE_NO_ACK message may be sent. If CurrentGreen is different from TargetGreen, then CurrentGreen should be changed by DeltaGreen. If CurrentGreen reaches TargetGreen, then a LIGHT_STATE_NO_ACK message may be sent. If CurrentBlue is different from TargetBlue, then CurrentBlue should be changed by DeltaBlue. If CurrentBlue reaches TargetBlue, then a LIGHT_STATE_NO_ACK message may be sent. If LevelSDState is Attacking and CurrentLevel is TargetLevel, then LevelSDState should be set to Sustaining and a timer starts for LevelSustain seconds. If LevelSDState is Sustaining and the timer expires, LevelSDState is set to Decaying, TargetLevel is set to zero, DeltaLevel is set to the difference between CurrentLevel and TargetLevel divided by LevelDecay seconds, and a LIGHT_STATE_NO_ACK message may be sent. If LevelSDState is Decaying, and CurrentLevel is the same as TargetLevel, then LevelSDState is set to Idle.

41.1 Modules of Light Model

Client

Server

41.2 Data Structures of Light Model

`struct CSRMESH_LIGHT_SET_LEVEL_T`

CSRmesh Light Model message types.

`struct CSRMESH_LIGHT_SET_RGB_T`

Setting Light Colour: Upon receiving a LIGHT_SET_RGB_NO_ACK message, the device saves the Level, Red, Green, and Blue fields into the TargetLevel, TargetRed, TargetGreen, and TargetBlue variables respectively. LevelSDState should be set to Attacking. If the Duration field is zero, then the device saves the Level, Red, Green, and Blue fields into the CurrentLevel, CurrentRed, CurrentGreen and CurrentBlue variables, and sets the DeltaLevel, DeltaRed, DeltaGreen, and DeltaBlue variables to zero. If the Duration field is greater than zero, then the device calculates the DeltaLevel, DeltaRed, DeltaGreen, and DeltaBlue levels from the differences between the Current values and the Target values divided by the Duration field, so that over Duration seconds, the CurrentLevel, CurrentRed, CurrentGreen, and CurrentBlue variables are changed smoothly to the TargetLevel, TargetRed, TargetGreen and TargetBlue values. If ACK is requested, the device responds with a LIGHT_STATE message.

`struct CSRMESH_LIGHT_SET_POWER_LEVEL_T`

Setting Light Power and Light Level: Upon receiving a LIGHT_SET_POWER_LEVEL_NO_ACK message, the device sets the current PowerState to the Power field, the TargetLevel variable to the Level field, the DeltaLevel to the difference between TargetLevel and CurrentLevel divided by the LevelDuration field, saves the Sustain and Decay fields into the LevelSustain and LevelDecay variables, and sets LevelSDState to the Attacking state. If ACK is requested, the device should respond with a LIGHT_STATE message.

`struct CSRMESH_LIGHT_SET_COLOR_TEMP_T`

Setting Light Colour Temperature: Upon receiving a LIGHT_SET_COLOR_TEMP message, the device saves the ColorTemperature field into the TargetColorTemperature state variable. If the TempDuration field is zero, the CurrentColorTemperature variable is set to TargetColorTemperature and DeltaColorTemperature is set to zero. If the TempDuration field is greater than zero, then the device calculates the difference between TargetColorTemperature and CurrentColorTemperature, over the TempDuration field and store this into a DeltaColorTemperature state variable, so that over TempDuration seconds, CurrentColorTemperature changes smoothly to TargetColorTemperature. The device then responds with a LIGHT_STATE message.

`struct CSRMESH_LIGHT_GET_STATE_T`

Getting Light State: Upon receiving a LIGHT_GET_STATE message, the device responds with a LIGHT_STATE message.

`struct CSRMESH_LIGHT_STATE_T`

Current light state.

`struct CSRMESH_LIGHT_SET_WHITE_T`

Setting Light White level.

`struct CSRMESH_LIGHT_GET_WHITE_T`

Setting Light White level.

`struct CSRMESH_LIGHT_WHITE_T`

Setting Light White level.

42 Client

Detailed Description

42.1 Functions of Client

```
CSRmeshResult LightModelClientInit (CSRMESH_MODEL_CALLBACK_T app_callback)
```

Initialises Light Model Client functionality.

```
CSRmeshResult LightSetLevel (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8 ttl,
CSRMESH_LIGHT_SET_LEVEL_T *p_params, bool request_ack)
```

Setting Light Level: Upon receiving a LIGHT_SET_LEVEL_NO_ACK message, the device saves the Level field into the CurrentLevel model state. LevelSDState should be set to Idle. If ACK is requested, the device should respond with a LIGHT_STATE message.

```
CSRmeshResult LightSetRgb (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8 ttl,
CSRMESH_LIGHT_SET_RGB_T *p_params, bool request_ack)
```

Setting Light Colour: Upon receiving a LIGHT_SET_RGB_NO_ACK message, the device saves the Level, Red, Green, and Blue fields into the TargetLevel, TargetRed, TargetGreen, and TargetBlue variables respectively. LevelSDState should be set to Attacking. If the Duration field is zero, then the device saves the Level, Red, Green, and Blue fields into the CurrentLevel, CurrentRed, CurrentGreen and CurrentBlue variables, and sets the DeltaLevel, DeltaRed, DeltaGreen, and DeltaBlue variables to zero. If the Duration field is greater than zero, then the device calculates the DeltaLevel, DeltaRed, DeltaGreen, and DeltaBlue levels from the differences between the Current values and the Target values divided by the Duration field, so that over Duration seconds, the CurrentLevel, CurrentRed, CurrentGreen, and CurrentBlue variables are changed smoothly to the TargetLevel, TargetRed, TargetGreen and TargetBlue values. If ACK is requested, the device responds with a LIGHT_STATE message.

```
CSRmeshResult LightSetPowerLevel (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8
ttl, CSRMESH_LIGHT_SET_POWER_LEVEL_T *p_params, bool request_ack)
```

Setting Light Power and Light Level: Upon receiving a LIGHT_SET_POWER_LEVEL_NO_ACK message, the device sets the current PowerState to the Power field, the TargetLevel variable to the Level field, the DeltaLevel to the difference between TargetLevel and CurrentLevel divided by the LevelDuration field, saves the Sustain and Decay fields into the LevelSustain and LevelDecay variables, and sets LevelSDState to the Attacking state. If ACK is requested, the device should respond with a LIGHT_STATE message.

```
CSRmeshResult LightSetColorTemp (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8
ttl, CSRMESH_LIGHT_SET_COLOR_TEMP_T *p_params)
```

Setting Light Colour Temperature: Upon receiving a LIGHT_SET_COLOR_TEMP message, the device saves the ColorTemperature field into the TargetColorTemperature state variable. If the TempDuration field is zero, the CurrentColorTemperature variable is set to TargetColorTemperature and DeltaColorTemperature is set to zero. If the TempDuration field is greater than zero, then the device calculates the difference between TargetColorTemperature and CurrentColorTemperature, over the TempDuration field and store this into a DeltaColorTemperature state variable, so that over TempDuration seconds, CurrentColorTemperature changes smoothly to TargetColorTemperature. The device then responds with a LIGHT_STATE message.

```
CSRmeshResult LightGetState (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8 ttl,
CSRMESH_LIGHT_GET_STATE_T *p_params)
```

Getting Light State: Upon receiving a LIGHT_GET_STATE message, the device responds with a LIGHT_STATE message.

```
CSRmeshResult LightSetWhite (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8 ttl,
CSRMESH_LIGHT_SET_WHITE_T *p_params, bool request_ack)
```

Setting Light White level.

```
CSRmeshResult LightGetWhite (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8 ttl,
CSRMESH_LIGHT_GET_WHITE_T *p_params)
```

Setting Light White level.

42.2 LightModelClientInit Function Documentation

Syntax

```
CSRmeshResult LightModelClientInit (CSRMESH_MODEL_CALLBACK_T app_callback)
```

Description

Parameters

Parameter	Description
app_callback	Pointer to the application callback function that will be called when the model client receives a message.

Returns

CSRmeshResult. Refer to CSRmeshResult.

42.3 LightSetLevel Function Documentation

Syntax

```
CSRmeshResult LightSetLevel (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8 ttl,
CSRMESH_LIGHT_SET_LEVEL_T * p_params, bool request_ack)
```

Description

This function packs the given parameters into a CSRmesh message and sends it over the network. When a response is received the application callback function is called to notify the CSRMESH_LIGHT_STATE

Parameters

Parameter	Description
nw_id	Network identifier over which the message has to be sent.
dest_id	16-bit identifier of the destination device/group
ttl	TTL value with which the message needs to be sent.
p_params	Pointer to the message parameters of type CSRMESH_LIGHT_SET_LEVEL_T
request_ack	TRUE if an acknowledgement is required

Returns

CSRmeshResult. Refer to CSRmeshResult.

42.4 LightSetRgb Function Documentation

Syntax

```
CSRmeshResult LightSetRgb (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8 ttl,
CSRMESH_LIGHT_SET_RGB_T * p_params, bool request_ack)
```

Description

This function packs the given parameters into a CSRmesh message and sends it over the network. When a response is received the application callback function is called to notify the CSRMESH_LIGHT_STATE

Parameters

Parameter	Description
nw_id	Network identifier over which the message has to be sent.
dest_id	16-bit identifier of the destination device/group
ttl	TTL value with which the message needs to be sent.
p_params	Pointer to the message parameters of type CSRMESH_LIGHT_SET_RGB_T
request_ack	TRUE if an acknowledgement is required

Returns

CSRmeshResult. Refer to CSRmeshResult.

42.5 LightSetPowerLevel Function Documentation

Syntax

```
CSRmeshResult LightSetPowerLevel (CsrUint8 nw_id, CsrUint16 dest_id,
CsrUint8 ttl, CSRMESH_LIGHT_SET_POWER_LEVEL_T * p_params, bool request_ack)
```

Description

This function packs the given parameters into a CSRmesh message and sends it over the network. When a response is received the application callback function is called to notify the CSRMESH_LIGHT_STATE

Parameters

Parameter	Description
nw_id	Network identifier over which the message has to be sent.
dest_id	16-bit identifier of the destination device/group
ttl	TTL value with which the message needs to be sent.
p_params	Pointer to the message parameters of type CSRMESH_LIGHT_SET_POWER_LEVEL_T
request_ack	TRUE if an acknowledgement is required

Returns

CSRmeshResult. Refer to CSRmeshResult.

42.6 LightSetColorTemp Function Documentation

Syntax

```
CSRmeshResult LightSetColorTemp (CsrUint8 nw_id, CsrUint16 dest_id,
CsrUint8 ttl, CSRMESH_LIGHT_SET_COLOR_TEMP_T * p_params)
```

Description

This function packs the given parameters into a CSRmesh message and sends it over the network. When a response is received the application callback function is called to notify the CSRMESH_LIGHT_STATE

Parameters

Parameter	Description
nw_id	Network identifier over which the message has to be sent.
dest_id	16-bit identifier of the destination device/group
ttl	TTL value with which the message needs to be sent.
p_params	Pointer to the message parameters of type CSRMESH_LIGHT_SET_COLOR_TEMP_T

Returns

CSRmeshResult. Refer to CSRmeshResult.

42.7 LightGetState Function Documentation

Syntax

```
CSRmeshResult LightGetState (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8 ttl,
CSRMESH_LIGHT_GET_STATE_T * p_params)
```

Description

This function packs the given parameters into a CSRmesh message and sends it over the network. When a response is received the application callback function is called to notify the CSRMESH_LIGHT_STATE

Parameters

Parameter	Description
nw_id	Network identifier over which the message has to be sent.
dest_id	16-bit identifier of the destination device/group
ttl	TTL value with which the message needs to be sent.
p_params	Pointer to the message parameters of type CSRMESH_LIGHT_GET_STATE_T

Returns

CSRmeshResult. Refer to CSRmeshResult.

42.8 LightSetWhite Function Documentation

Syntax

```
CSRmeshResult LightSetWhite (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8 ttl,
CSRMESH_LIGHT_SET_WHITE_T * p_params, bool request_ack)
```

Description

This function packs the given parameters into a CSRmesh message and sends it over the network. When a response is received the application callback function is called to notify the CSRMESH_LIGHT_WHITE

Parameters

Parameter	Description
nw_id	Network identifier over which the message has to be sent.
dest_id	16-bit identifier of the destination device/group
ttl	TTL value with which the message needs to be sent.
p_params	Pointer to the message parameters of type CSRMESH_LIGHT_SET_WHITE_T
request_ack	TRUE if an acknowledgement is required

Returns

CSRmeshResult. Refer to CSRmeshResult.

42.9 LightGetWhite Function Documentation

Syntax

```
CSRmeshResult LightGetWhite (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8 ttl,
CSRMESH_LIGHT_GET_WHITE_T * p_params)
```

Description

This function packs the given parameters into a CSRmesh message and sends it over the network. When a response is received the application callback function is called to notify the CSRMESH_LIGHT_WHITE

Parameters

Parameter	Description
nw_id	Network identifier over which the message has to be sent.
dest_id	16-bit identifier of the destination device/group
ttl	TTL value with which the message needs to be sent.
p_params	Pointer to the message parameters of type CSRMESH_LIGHT_GET_WHITE_T

Returns

CSRmeshResult. Refer to CSRmeshResult.

43 Server

Detailed Description

43.1 Functions of Server

```
CSRmeshResult LightModelInit (CsrUint8 nw_id, CsrUint16 *group_id_list,
CsrUint16 num_groups, CSRMESH_MODEL_CALLBACK_T app_callback)
```

Model Initialisation.

```
CSRmeshResult LightState (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8 ttl,
CSRMESH_LIGHT_STATE_T *p_params, bool request_ack)
```

Current light state.

```
CSRmeshResult LightWhite (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8 ttl,
CSRMESH_LIGHT_WHITE_T *p_params, bool request_ack)
```

Setting Light White level.

43.2 LightModelInit Function Documentation

Syntax

```
CSRmeshResult LightModelInit (CsrUint8 nw_id, CsrUint16 * group_id_list,
CsrUint16 num_groups, CSRMESH_MODEL_CALLBACK_T app_callback)
```

Description

Registers the model handler with the CSRmesh. Sets the CSRmesh to report num_groups as the maximum number of groups supported for the model

Parameters

Parameter	Description
`nw_id`	Identifier of the network to which the model has to be registered.
`group_id_list`	Pointer to a uint16 array to hold assigned group_ids. This must be NULL if no groups are supported

Parameter	Description
num_groups	Size of the group_id_list. This must be 0 if no groups are supported.
app_callback	Pointer to the application callback function. This function will be called to notify all model specific messages

Returns

CSRmeshResult. Refer to CSRmeshResult.

43.3 LightState Function Documentation

Syntax

```
CSRmeshResult LightState (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8 ttl,
CSRMESH_LIGHT_STATE_T * p_params, bool request_ack)
```

Description

This function packs the given parameters into a CSRmesh message and sends it over the network.

Parameters

Parameter	Description
nw_id	Network identifier over which the message has to be sent.
dest_id	16-bit identifier of the destination device/group
ttl	TTL value with which the message needs to be sent.
p_params	Pointer to the message parameters of type CSRMESH_LIGHT_STATE_T
request_ack	TRUE if an acknowledgement is required

Returns

CSRmeshResult. Refer to CSRmeshResult.

43.4 LightWhite Function Documentation

Syntax

```
CSRmeshResult LightWhite (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8 ttl,
CSRMESH_LIGHT_WHITE_T * p_params, bool request_ack)
```

Description

This function packs the given parameters into a CSRmesh message and sends it over the network.

Parameters

Parameter	Description
nw_id	Network identifier over which the message has to be sent.
dest_id	16-bit identifier of the destination device/group
ttl	TTL value with which the message needs to be sent.
p_params	Pointer to the message parameters of type CSRMESH_LIGHT_WHITE_T
request_ack	TRUE if an acknowledgement is required

Returns

CSRmeshResult. Refer to CSRmeshResult.

44 Asset Model

Detailed Description

Asset Model API.

The Asset Model announces the presence of an asset within a mesh network.

The Asset model has the following states:

- Interval
- SideEffects
- ToDestinationID

The Interval variable is an unsigned 16-bit integer in seconds representing times from 1 second to 65535 seconds. The value 0x0000 means that the asset will never transmit an asset broadcast message. Some devices will not be able to support every possible Interval value and therefore their Interval value will be rounded to the nearest supported Interval. The rounding of Interval is implementation specific and not specified further.

The SideEffects variable is a 16-bit bit field that represents one or more possible side-effects that this asset would like to be observed. The following side effect bits are defined:

- Light=0
- Audio=1
- Movement=2
- Stationary=3
- Human=4

ToDestionation state is a GroupID that all asset broadcast messages are sent to.

If the Interval state has a non-zero value then the device sends an ASSET_ANNOUNCE message once every Interval to the ToDestinationID with the SideEffects value. This message is sent with TTL 0, repeated Ã¢â‚¬Ëœ numberOfAnnouncesÃ¢â‚¬â„¢ times with a gap of Ã¢â‚¬Ëœ announceIntervalÃ¢â‚¬â„¢ between each repeat.

44.1 Modules of Asset Model

Client

Server

44.2 Data Structures of Asset Model

`struct CSRMESH_ASSET_SET_STATE_T`

CSRmesh Asset Model message types.

`struct CSRMESH_ASSET_GET_STATE_T`

Getting Asset State: Upon receiving an ASSET_GET_STATE message, the device responds with an ASSET_STATE message with the current state information.

`struct CSRMESH_ASSET_STATE_T`

Current asset state.

`struct CSRMESH_ASSET_ANNOUNCE_T`

Asset announcement.

45 Client

Detailed Description

45.1 Functions of Client

```
CSRmeshResult AssetModelClientInit (CSRMESH_MODEL_CALLBACK_T app_callback)
```

Initialises Asset Model Client functionality.

```
CSRmeshResult AssetSetState (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8 ttl,
CSRMESH_ASSET_SET_STATE_T *p_params)
```

Setting Asset State: Upon receiving an ASSET_SET_STATE message, the device saves the Interval, SideEffects, ToDestination, TxPower, Number of Announcements and AnnounceInterval fields into the appropriate state values. It then responds with an ASSET_STATE message with the current state information.

```
CSRmeshResult AssetGetState (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8 ttl,
CSRMESH_ASSET_GET_STATE_T *p_params)
```

Getting Asset State: Upon receiving an ASSET_GET_STATE message, the device responds with an ASSET_STATE message with the current state information.

45.2 AssetModelClientInit Function Documentation

Syntax

```
CSRmeshResult AssetModelClientInit (CSRMESH_MODEL_CALLBACK_T app_callback)
```

Description

Parameters

Parameter	Description
app_callback	Pointer to the application callback function that will be called when the model client receives a message.

Returns

CSRmeshResult. Refer to CSRmeshResult.

45.3 AssetSetState Function Documentation

Syntax

```
CSRmeshResult AssetSetState (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8 ttl,
CSRMESH_ASSET_SET_STATE_T * p_params)
```

Description

This function packs the given parameters into a CSRmesh message and sends it over the network. When a response is received the application callback function is called to notify the CSRMESH_ASSET_STATE

Parameters

Parameter	Description
nw_id	Network identifier over which the message has to be sent.
dest_id	16-bit identifier of the destination device/group
ttl	TTL value with which the message needs to be sent.
p_params	Pointer to the message parameters of type CSRMESH_ASSET_SET_STATE_T

Returns

CSRmeshResult. Refer to CSRmeshResult.

45.4 AssetGetState Function Documentation

Syntax

```
CSRmeshResult AssetGetState (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8 ttl,
CSRMESH_ASSET_GET_STATE_T * p_params)
```

Description

This function packs the given parameters into a CSRmesh message and sends it over the network. When a response is received the application callback function is called to notify the CSRMESH_ASSET_STATE

Parameters

Parameter	Description
`nw_id`	Network identifier over which the message has to be sent.
`dest_id`	16-bit identifier of the destination device/group
`ttl`	TTL value with which the message needs to be sent.
`p_params`	Pointer to the message parameters of type CSRMESH_ASSET_GET_STATE_T

Returns

CSRmeshResult. Refer to CSRmeshResult.

46 Server

Detailed Description

46.1 Functions of Server

```
CSRmeshResult AssetModelInit (CsrUint8 nw_id, CsrUint16 *group_id_list,
CsrUint16 num_groups, CSRMESH_MODEL_CALLBACK_T app_callback)
```

Model Initialisation.

```
CSRmeshResult AssetState (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8 ttl,
CSRMESH_ASSET_STATE_T *p_params)
```

Current asset state.

```
CSRmeshResult AssetAnnounce (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8 ttl,
CSRMESH_ASSET_ANNOUNCE_T *p_params)
```

Asset announcement.

46.2 AssetModelInit Function Documentation

Syntax

```
CSRmeshResult AssetModelInit (CsrUint8 nw_id, CsrUint16 * group_id_list,
CsrUint16 num_groups, CSRMESH_MODEL_CALLBACK_T app_callback)
```

Description

Registers the model handler with the CSRmesh. Sets the CSRmesh to report num_groups as the maximum number of groups supported for the model

Parameters

Parameter	Description
`nw_id`	Identifier of the network to which the model has to be registered.
`group_id_list`	Pointer to a uint16 array to hold assigned group_ids. This must be NULL if no groups are supported

Parameter	Description
num_groups	Size of the group_id_list. This must be 0 if no groups are supported.
app_callback	Pointer to the application callback function. This function will be called to notify all model specific messages

Returns

CSRmeshResult. Refer to CSRmeshResult.

46.3 AssetState Function Documentation

Syntax

```
CSRmeshResult AssetState (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8 ttl,
CSRMESH_ASSET_STATE_T * p_params)
```

Description

This function packs the given parameters into a CSRmesh message and sends it over the network.

Parameters

Parameter	Description
nw_id	Network identifier over which the message has to be sent.
dest_id	16-bit identifier of the destination device/group
ttl	TTL value with which the message needs to be sent.
p_params	Pointer to the message parameters of type CSRMESH_ASSET_STATE_T

Returns

CSRmeshResult. Refer to CSRmeshResult.

46.4 AssetAnnounce Function Documentation

Syntax

```
CSRmeshResult AssetAnnounce (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8 ttl,
CSRMESH_ASSET_ANNOUNCE_T * p_params)
```

Description

This function packs the given parameters into a CSRmesh message and sends it over the network.

Parameters

Parameter	Description
`nw_id`	Network identifier over which the message has to be sent.
`dest_id`	16-bit identifier of the destination device/group
`ttl`	TTL value with which the message needs to be sent.
`p_params`	Pointer to the message parameters of type CSRMESH_ASSET_ANNOUNCE_T

Returns

CSRmeshResult. Refer to CSRmeshResult.

47 Tracker Model

Detailed Description

Tracker Model API.

The Tracker Model finds assets within a mesh network. When a device wishes to find an asset it sends a find message into the mesh network. Any device that has recently heard from an asset that is trying to be found responds with an appropriate message indicating when and how loud it heard this message. It is possible that a device that has only heard a message with a weak signal, delays its response or does not respond at all if it subsequently hears another device saying that it had heard the device at a stronger signal level more recently. This optimises the number of messages transmitted by devices in the network when an asset has been moving around the network. A device that is listening for and recording asset presence messages do not need to save every presence message that it receives. It could only save a subset of messages based on various heuristics such as signal strength or number of devices or frequency of message receipt.

The tracker model has a tracker cache, with a list of the last seen asset broadcasts with associated data. Such associated data includes the received signal strength and the time when that asset broadcast was received. It is impossible to store all possible asset broadcast messages received on a device and therefore it is assumed that devices implementing the tracker model uses a least-recently-used algorithm to remove old and/or weak data. The actual algorithm used is an implementation issue.

When a tracker device hears an ASSET_ANNOUNCE it first listens for other TRACKER_REPORT reports for the same asset to determine if the cached information is either louder or newer than other devices. If the server does not receive any other TRACKER_REPORT message or the RSSI and Age fields of those messages are louder or younger than the information in its tracker cache, then it does not send a TRACKER_REPORT message for the asset. The timing of sending a TRACKER_REPORT message is a function of the age and RSSI values in the tracker cache. The RSSI heard is converted to a delay in milliseconds using the formula: (DelayOffset - RSSI) * DelayFactor. RSSI is a negative number usually between -40 and -100. DelayOffset is 60 by default and DelayFactor is 30. This gives a delay of 3 to 4.8 seconds before a REPORT is sent.

47.1 Modules of Tracker Model

Client

Server

47.2 Data Structures of Tracker Model

`struct CSRMESH_TRACKER_FIND_T`

CSRmesh Tracker Model message types.

`struct CSRMESH_TRACKER_FOUND_T`

Asset found.

`struct CSRMESH_TRACKER_REPORT_T`

Asset report.

`struct CSRMESH_TRACKER_SET_PROXIMITY_CONFIG_T`

Set tracker proximity config.

48 Client

Detailed Description

48.1 Functions of Client

```
CSRmeshResult TrackerModelClientInit (CSRMESH_MODEL_CALLBACK_T app_callback)
```

Initialises Tracker Model Client functionality.

```
CSRmeshResult TrackerFind (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8 ttl,
CSRMESH_TRACKER_FIND_T *p_params)
```

Finding an Asset: Upon receiving a TRACKER_FIND message, the server checks its tracker cache to see if it has received an ASSET_ANNOUNCE message recently that has the same DeviceID. If it finds one, it will send a TRACKER_FOUND message with the cached information.

```
CSRmeshResult TrackerClearCache (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8
ttl)
```

Clear tracker cache.

```
CSRmeshResult TrackerSetProximityConfig (CsrUint8 nw_id, CsrUint16 dest_id,
CsrUint8 ttl, CSRMESH_TRACKER_SET_PROXIMITY_CONFIG_T *p_params)
```

Set tracker proximity config.

48.2 TrackerModelClientInit Function Documentation

Syntax

```
CSRmeshResult TrackerModelClientInit (CSRMESH_MODEL_CALLBACK_T app_callback)
```

Description

Parameters

Parameter	Description
app_callback	Pointer to the application callback function that will be called when the model client receives a message.

Returns

CSRmeshResult. Refer to CSRmeshResult.

48.3 TrackerFind Function Documentation

Syntax

```
CSRmeshResult TrackerFind (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8 ttl,
CSRMESH_TRACKER_FIND_T * p_params)
```

Description

This function packs the given parameters into a CSRmesh message and sends it over the network. When a response is received the application callback function is called to notify the CSRMESH_TRACKER_FOUND

Parameters

Parameter	Description
nw_id	Network identifier over which the message has to be sent.
dest_id	16-bit identifier of the destination device/group
ttl	TTL value with which the message needs to be sent.
p_params	Pointer to the message parameters of type CSRMESH_TRACKER_FIND_T

Returns

CSRmeshResult. Refer to CSRmeshResult.

48.4 TrackerClearCache Function Documentation

Syntax

```
CSRmeshResult TrackerClearCache (CsrUint8 nw_id, CsrUint16 dest_id,
CsrUint8 ttl)
```

Description

This function packs the given parameters into a CSRmesh message and sends it over the network.

Parameters

Parameter	Description
nw_id	Network identifier over which the message has to be sent.
dest_id	16-bit identifier of the destination device/group
ttl	TTL value with which the message needs to be sent.

Returns

CSRmeshResult. Refer to CSRmeshResult.

48.5 TrackerSetProximityConfig Function Documentation

Syntax

```
CSRmeshResult TrackerSetProximityConfig (CsrUint8 nw_id, CsrUint16 dest_id,
CsrUint8 ttl, CSRMESH_TRACKER_SET_PROXIMITY_CONFIG_T * p_params)
```

Description

This function packs the given parameters into a CSRmesh message and sends it over the network.

Parameters

Parameter	Description
nw_id	Network identifier over which the message has to be sent.
dest_id	16-bit identifier of the destination device/group

Parameter	Description
`ttl`	TTL value with which the message needs to be sent.
`p_params`	Pointer to the message parameters of type CSRMESH_TRACKER_SET_PROXIMITY_CONFIG_T

Returns

CSRmeshResult. Refer to CSRmeshResult.

49 Server

Detailed Description

49.1 Functions of Server

```
CSRmeshResult TrackerModelInit (CsrUint8 nw_id, CsrUint16 *group_id_list,
CsrUint16 num_groups, CSRMESH_MODEL_CALLBACK_T app_callback)
```

Model Initialisation.

```
CSRmeshResult TrackerFound (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8 ttl,
CSRMESH_TRACKER_FOUND_T *p_params)
```

Asset found.

```
CSRmeshResult TrackerReport (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8 ttl,
CSRMESH_TRACKER_REPORT_T *p_params)
```

Asset report.

49.2 TrackerModelInit Function Documentation

Syntax

```
CSRmeshResult TrackerModelInit (CsrUint8 nw_id, CsrUint16 * group_id_list,
CsrUint16 num_groups, CSRMESH_MODEL_CALLBACK_T app_callback)
```

Description

Registers the model handler with the CSRmesh. Sets the CSRmesh to report num_groups as the maximum number of groups supported for the model

Parameters

Parameter	Description
`nw_id`	Identifier of the network to which the model has to be registered.
`group_id_list`	Pointer to a uint16 array to hold assigned group_ids. This must be NULL if no groups are supported

Parameter	Description
num_groups	Size of the group_id_list. This must be 0 if no groups are supported.
app_callback	Pointer to the application callback function. This function will be called to notify all model specific messages

Returns

CSRmeshResult. Refer to CSRmeshResult.

49.3 TrackerFound Function Documentation

Syntax

```
CSRmeshResult TrackerFound (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8 ttl,
CSRMESH_TRACKER_FOUND_T * p_params)
```

Description

This function packs the given parameters into a CSRmesh message and sends it over the network.

Parameters

Parameter	Description
nw_id	Network identifier over which the message has to be sent.
dest_id	16-bit identifier of the destination device/group
ttl	TTL value with which the message needs to be sent.
p_params	Pointer to the message parameters of type CSRMESH_TRACKER_FOUND_T

Returns

CSRmeshResult. Refer to CSRmeshResult.

49.4 TrackerReport Function Documentation

Syntax

```
CSRmeshResult TrackerReport (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8 ttl,
CSRMESH_TRACKER_REPORT_T * p_params)
```

Description

This function packs the given parameters into a CSRmesh message and sends it over the network.

Parameters

Parameter	Description
`nw_id`	Network identifier over which the message has to be sent.
`dest_id`	16-bit identifier of the destination device/group
`ttl`	TTL value with which the message needs to be sent.
`p_params`	Pointer to the message parameters of type CSRMESH_TRACKER_REPORT_T

Returns

CSRmeshResult. Refer to CSRmeshResult.

50 Time Model

Detailed Description

Time Model API.

The Time Model is used to distribute coordinated universal time, and time zone information, within a mesh network.

The time model has the following states:

- CurrentTime
- TimeInterval
- TimeZone

The current time state is the number of milliseconds that have passed since midnight on January 1st, 1970 at the Zulu time zone. The current time is represented as a 48-bit unsigned integer; it is not possible to represent any time before 1970.

The time interval state is the number of seconds between time broadcasts. This is represented in seconds as a 16-bit unsigned integer.

The time zone state is the number of 15-minute increments from UTC. This is an 8-bit signed integer.

If the TimeInterval state value is set to a non-zero value, then the server transmits a TIME_BROADCAST message approximately every TimeInterval seconds. The TTL is set to 0. The message is repeated, but each repeat uses the current value of the clock. The MasterFlag is set to 1 if this message is sent by the clock master, or set to 0 if this message is being relayed by another node.

50.1 Modules of Time Model

Client

Server

50.2 Data Structures of Time Model

`struct CSRMESH_TIME_SET_STATE_T`

CSRmesh Time Model message types.

`struct CSRMESH_TIME_GET_STATE_T`

Getting Time Broadcast Interval: Upon receiving a TIME_GET_STATE message, the device responds with a TIME_STATE message with the current state information.

`struct CSRMESH_TIME_STATE_T`

Set time broadcast interval.

`struct CSRMESH_TIME_BROADCAST_T`

Synchronise wall clock time from client device: This message is always sent with TTL=0. This message is sent at intervals by the clock master. It is always sent with TTL=0. It is repeated, but the time is updated before each repeat is sent. The clock master repeats the message 5 times, relaying stations repeat it 3 times. When a node receives a clock broadcast its behaviour depends on the current clock state: n MASTER: Ignore broadcasts.n INIT: Start the clock; relay this message. Set state to NO_RELAY if MasterFlag set, otherwise RELAY_MASTER. Start relay timer.n RELAY: Correct clock if required. Relay this message. Set state to NO_RELAY if MasterFlag set, otherwise RELAY_MASTER. Start relay timer.n NO_RELAY: Ignore. State will be reset to RELAY when the relay timer goes off.n RELAY_MASTER: Relay message only if it is from the clock master and set state to NO_RELAY.n The relay timer is by default 1/4 of the clock broadcast interval (15 seconds if the interval is 60 seconds). This means that each node will relay a message only once, and will give priority to messages from the clock master (which always causes the clock to be corrected). Messages from other nodes will only cause clock correction if they exceed the max clock skew (250ms).

51 Client

Detailed Description

51.1 Functions of Client

```
CSRmeshResult TimeModelClientInit (CSRMESH_MODEL_CALLBACK_T app_callback)
```

Initialises Time Model Client functionality.

```
CSRmeshResult TimeSetState (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8 ttl,
CSRMESH_TIME_SET_STATE_T *p_params)
```

Setting Time Broadcast Interval: Upon receiving a TIME_SET_STATE message, the device saves the TimeInterval field into the appropriate state value. It then responds with a TIME_STATE message with the current state information.

```
CSRmeshResult TimeGetState (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8 ttl,
CSRMESH_TIME_GET_STATE_T *p_params)
```

Getting Time Broadcast Interval: Upon receiving a TIME_GET_STATE message, the device responds with a TIME_STATE message with the current state information.

51.2 TimeModelClientInit Function Documentation

Syntax

```
CSRmeshResult TimeModelClientInit (CSRMESH_MODEL_CALLBACK_T app_callback)
```

Description

Parameters

Parameter	Description
app_callback	Pointer to the application callback function that will be called when the model client receives a message.

Returns

CSRmeshResult. Refer to CSRmeshResult.

51.3 TimeSetState Function Documentation

Syntax

```
CSRmeshResult TimeSetState (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8 ttl,
CSRMESH_TIME_SET_STATE_T * p_params)
```

Description

This function packs the given parameters into a CSRmesh message and sends it over the network. When a response is received the application callback function is called to notify the CSRMESH_TIME_STATE

Parameters

Parameter	Description
nw_id	Network identifier over which the message has to be sent.
dest_id	16-bit identifier of the destination device/group
ttl	TTL value with which the message needs to be sent.
p_params	Pointer to the message parameters of type CSRMESH_TIME_SET_STATE_T

Returns

CSRmeshResult. Refer to CSRmeshResult.

51.4 TimeGetState Function Documentation

Syntax

```
CSRmeshResult TimeGetState (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8 ttl,
CSRMESH_TIME_GET_STATE_T * p_params)
```

Description

This function packs the given parameters into a CSRmesh message and sends it over the network. When a response is received the application callback function is called to notify the CSRMESH_TIME_STATE

Parameters

Parameter	Description
`nw_id`	Network identifier over which the message has to be sent.
`dest_id`	16-bit identifier of the destination device/group
`ttl`	TTL value with which the message needs to be sent.
`p_params`	Pointer to the message parameters of type CSRMESH_TIME_GET_STATE_T

Returns

CSRmeshResult. Refer to CSRmeshResult.

52 Server

Detailed Description

52.1 Functions of Server

```
CSRmeshResult TimeModelInit (CsrUint8 nw_id, CsrUint16 *group_id_list,
CsrUint16 num_groups, CSRMESH_MODEL_CALLBACK_T app_callback)
```

Model Initialisation.

```
CSRmeshResult TimeState (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8 ttl,
CSRMESH_TIME_STATE_T *p_params)
```

Set time broadcast interval.

```
CSRmeshResult TimeBroadcast (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8 ttl,
CSRMESH_TIME_BROADCAST_T *p_params)
```

Synchronise wall clock time from client device: This message is always sent with TTL=0. This message is sent at intervals by the clock master. It is always sent with TTL=0. It is repeated, but the time is updated before each repeat is sent. The clock master repeats the message 5 times, relaying stations repeat it 3 times. When a node receives a clock broadcast its behaviour depends on the current clock state: n MASTER: Ignore broadcasts.n INIT: Start the clock; relay this message. Set state to NO_RELAY if MasterFlag set, otherwise RELAY_MASTER. Start relay timer.n RELAY: Correct clock if required. Relay this message. Set state to NO_RELAY if MasterFlag set, otherwise RELAY_MASTER. Start relay timer.n NO_RELAY: Ignore. State will be reset to RELAY when the relay timer goes off.n RELAY_MASTER: Relay message only if it is from the clock master and set state to NO_RELAY.n The relay timer is by default 1/4 of the clock broadcast interval (15 seconds if the interval is 60 seconds). This means that each node will relay a message only once, and will give priority to messages from the clock master (which always causes the clock to be corrected). Messages from other nodes will only cause clock correction if they exceed the max clock skew (250ms).

52.2 TimeModelInit Function Documentation

Syntax

```
CSRmeshResult TimeModelInit (CsrUint8 nw_id, CsrUint16 * group_id_list,
CsrUint16 num_groups, CSRMESH_MODEL_CALLBACK_T app_callback)
```

Description

Registers the model handler with the CSRmesh. Sets the CSRmesh to report num_groups as the maximum number of groups supported for the model

Parameters

Parameter	Description
nw_id	Identifier of the network to which the model has to be registered.
group_id_list	Pointer to a uint16 array to hold assigned group_ids. This must be NULL if no groups are supported
num_groups	Size of the group_id_list. This must be 0 if no groups are supported.
app_callback	Pointer to the application callback function. This function will be called to notify all model specific messages

Returns

CSRmeshResult. Refer to CSRmeshResult.

52.3 TimeState Function Documentation

Syntax

```
CSRmeshResult TimeState (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8 ttl,
CSRMESH_TIME_STATE_T * p_params)
```

Description

This function packs the given parameters into a CSRmesh message and sends it over the network.

Parameters

Parameter	Description
nw_id	Network identifier over which the message has to be sent.
dest_id	16-bit identifier of the destination device/group
ttl	TTL value with which the message needs to be sent.
p_params	Pointer to the message parameters of type CSRMESH_TIME_STATE_T

Returns

CSRmeshResult. Refer to CSRmeshResult.

52.4 TimeBroadcast Function Documentation

Syntax

```
CSRmeshResult TimeBroadcast (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8 ttl,
CSRMESH_TIME_BROADCAST_T * p_params)
```

Description

This function packs the given parameters into a CSRmesh message and sends it over the network.

Parameters

Parameter	Description
nw_id	Network identifier over which the message has to be sent.
dest_id	16-bit identifier of the destination device/group
ttl	TTL value with which the message needs to be sent.
p_params	Pointer to the message parameters of type CSRMESH_TIME_BROADCAST_T

Returns

CSRmeshResult. Refer to CSRmeshResult.

53 Switch Model

Detailed Description

Switch Model API.

The switch model does not define any messages. The switch model is configured using the group model. Switch Client Behaviour: If the user presses the on switch with a given switch_number, then the client sends a POWER_SET_STATE message to the appropriate GroupID as identified by the Instance value from the group model. If the user rotates a dimmer switch, then the client may send a LIGHT_LEVEL_NO_ACK message to the appropriate GroupID as configured in the group model. If the user finishes rotating a dimmer switch, then the client may send a LIGHT_LEVEL, and await an acknowledgement from the lights that this dimmer controls.

53.1 Modules of Switch Model

Client

Server

54 Client

Detailed Description

54.1 Functions of Client

```
CSRmeshResult SwitchModelClientInit (CSRMESH_MODEL_CALLBACK_T app_callback)
```

Initialises Switch Model Client functionality.

54.2 SwitchModelClientInit Function Documentation

Syntax

```
CSRmeshResult SwitchModelClientInit (CSRMESH_MODEL_CALLBACK_T app_callback)
```

Description

Parameters

Parameter	Description
app_callback	Pointer to the application callback function that will be called when the model client receives a message.

Returns

CSRmeshResult. Refer to CSRmeshResult.

55 Server

Detailed Description

55.1 Functions of Server

```
CSRmeshResult SwitchModelInit (CsrUint8 nw_id, CsrUint16 *group_id_list,
CsrUint16 num_groups, CSRMESH_MODEL_CALLBACK_T app_callback)
```

Model Initialisation.

55.2 SwitchModelInit Function Documentation

Syntax

```
CSRmeshResult SwitchModelInit (CsrUint8 nw_id, CsrUint16 * group_id_list,
CsrUint16 num_groups, CSRMESH_MODEL_CALLBACK_T app_callback)
```

Description

Registers the model handler with the CSRmesh. Sets the CSRmesh to report num_groups as the maximum number of groups supported for the model

Parameters

Parameter	Description
`nw_id`	Identifier of the network to which the model has to be registered.
`group_id_list`	Pointer to a uint16 array to hold assigned group_ids. This must be NULL if no groups are supported
`num_groups`	Size of the group_id_list. This must be 0 if no groups are supported.
`app_callback`	Pointer to the application callback function. This function will be called to notify all model specific messages

Returns

CSRmeshResult. Refer to CSRmeshResult.

56 Tuning Model

Detailed Description

Tuning Model API.

The Tuning Model is designed to allow for the local management of duty cycles through the determination of the local network density. The purpose of the Tuning model is to allow a device to detect that there are other similar devices nearby, and construct a metric for mesh density in the neighbourhood. The mesh density has to be a proxy for the local ability of the mesh to relay messages. The only information available is from the messages measured from the device itself. A reasonable approach it to measure how many messages are received, and adjust the listening duty cycle up or down to make that metric match a goal either fixed for this deployment or commanded by a higher-level monitoring program (SET_GOAL). Such a metric must not depend on normal product-oriented mesh traffic, because it is sporadic, unpredictable and time-dependent. In order to know what is measured, dedicated messages must be used with the following properties:

- Messages are not forwarded through the mesh, thus measuring only immediately neighbours.
- Messages do not occur in real traffic, thus making the metric independent of other time varying activities.
- Messages arrive at a fixed rate, so measuring the same information before and after an adjustment.
- Messages are transmitted from all stations in the same manner, providing uniformity throughout the mesh.
- Messages identify distinct sources, so that we can count neighbours individually.

The TUNING_PROBE message is used to facilitate the metric computation, when sent with TTL = 0.

56.1 Modules of Tuning Model

Client

Server

56.2 Data Structures of Tuning Model

`struct CSRMESH_TUNING_PROBE_T`

CSRmesh Tuning Model message types.

`struct CSRMESH_TUNING_GET_STATS_T`

Getting Tuning Stats: These messages are aimed at collecting statistics from specific nodes. This message allows for the request of all information or for some of its parts. Responses are multi-parts, each identified with an index (combining a continuation flag - top bit). MissingReplyParts for the required field serves at determining the specific responses one would like to collect. If instead all the information is requested, setting this field to zero will inform the destination device to send all messages. Importantly, response (STATS_RESPONSE) messages will not necessarily come back in order, or all reach the requestor. It is essential to handle these cases in the treatment of the collected responses.

`struct CSRMESH_TUNING_STATS_T`

Current Asset State: Response to the request. The PartNumber indicates the current index of the message, the top bit of this field is used to indicate that more messages are available after this part. For example, a response made up of three messages will have part numbers 0x80, 0x81 and 0x02. Each message has a maximum of two neighbours. The combination of these responses and PROBE (TTL>0) are a means to establish an overall perspective of the entire Mesh Network.

`struct CSRMESH_TUNING_ACK_CONFIG_T`

Current tuning config for a device: This message comes as a response to a SET_CONFIG. Encoding its various fields follow the same convention as the ones exposed in SET_CONFIG.

`struct CSRMESH_TUNING_SET_CONFIG_T`

Setting (or reading) tuning config: Omitted or zero fields mean do not change. This message enforces the state of the recipient. The state is defined by two goals, normal and high traffic, and their associated number of neighbour to decide which cases to follow. Goals are expressed with unit-less values ranging from 0 to 255. These goals relate to metrics calculated on the basis of density computed at the node and across the network. The expectation is for these goals to be maintained through modification of the receive duty cycle. The average of number of neighbours for high and normal traffic is expressed as a ratio, both numbers sharing the same denominator and each representing their respective numerators. The duty cycle encoding follows the same rules as per duty cycle encoding encountered in PROBE message. This message comes in two formats. A fully truncated form containing only the OpCode (thus of length 2) is used to indicate a request for information. This message should be answered by the appropriate ACK_CONFIG. Further interpretations of this message are: 1. Missing ACK field implies that a request for ACK_CONFIG is made. Thus, this is a special case of the fully truncated mode. However, the provided fields are meant to be used in the setting of goals. 2. Individual fields with zero value are meant NOT to be changed in the received element. Same as for missing fields in truncated messages. Furthermore, in order to improve testing, a combination of values for main and high goals are conventionally expected to be used for defining two behaviours: 1. Suspended: Tuning Probe messages (TTL=0) should be sent and statistics maintained, but the duty cycle should not be changed - thus goals will never be achieved. The encoding are: Main Goal = 0x00 and High Goal = 0xFE. 2. Disable: No Tuning Probe message should be sent and statistics should not be gathered - averaged values should decay. The encoding are: Main Goal = 0x00 and High Goal = 0xFF.

57 Client

Detailed Description

57.1 Functions of Client

```
CSRmeshResult TuningModelClientInit (CSRMESH_MODEL_CALLBACK_T app_callback)
```

Initialises Tuning Model Client functionality.

```
CSRmeshResult TuningProbe (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8 ttl,
CSRMESH_TUNING_PROBE_T *p_params)
```

Tuning Probe: The Tuning Probe message is sent to discover neighbours. This messages is issued by devices wanting to determine their density metrics.n The message is sent in two forms. A short form omitting both ScanDutyCycle and BatteryState with a TTL=0. This allows immediate neighbours to perform various calculations and in turn provide their own PROBE messages. The long version is only provided with TTL>0. This cannot be used for immediate neighbour density determination, but can be used to determine the overall network density. The ability to identify if a node is a potential pinch-point in the Mesh network can be achieved through the comparison of immediate and average number of neighbours within the network. The usage of the PROBE message with TTL=0 or TTL>0 is a way to perform these computations. It is worth noting that the periodicity of these two types of messages are different; messages with TTL>0 is much more infrequent than messages with TTL=0. Furthermore, it is wise not to use messages for TTL>0 and embedded values in the determination of the average values. The AverageNumNeighbour field is fixed point 6.2 format encoded. The ScanDutyCycle is expressing percentage for numbers from 1 to 100 and (x-100)/10 percentage for numbers from 101 to 255.

```
CSRmeshResult TuningGetStats (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8 ttl,
CSRMESH_TUNING_GET_STATS_T *p_params)
```

Getting Tuning Stats: These messages are aimed at collecting statistics from specific nodes. This message allows for the request of all information or for some of its parts. Responses are multi-parts, each identified with an index (combining a continuation flag - top bit). MissingReplyParts for the required field serves at determining the specific responses one would like to collect. If instead all the information is requested, setting this field to zero will inform the destination device to send all messages. Importantly, response (STATS_RESPONSE) messages will not necessarily come back in order, or all reach the requestor. It is essential to handle these cases in the treatment of the collected responses.

```
CSRmeshResult TuningSetConfig (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8 ttl,
CSRMESH_TUNING_SET_CONFIG_T *p_params)
```

Setting (or reading) tuning config: Omitted or zero fields mean do not change. This message enforces the state of the recipient. The state is defined by two goals, normal and high traffic, and their associated number of neighbour to decide which cases to follow. Goals are expressed with unit-less values ranging from 0 to 255. These goals relate to metrics calculated on the basis of density computed at the node and across the network. The expectation is for these goals to be maintained through modification of the receive duty cycle. The average of number of neighbours for high and normal traffic is expressed as a ratio, both numbers sharing the same denominator and each representing their respective numerators. The duty cycle encoding follows the same rules as per duty cycle encoding encountered in PROBE message. This message comes in two formats. A fully truncated form containing only the OpCode (thus of length 2) is used to indicate a request for information. This message should be answered by the appropriate ACK_CONFIG. Further interpretations of this message are: 1. Missing ACK field implies that a request for ACK_CONFIG is made. Thus, this is a special case of the fully truncated mode. However, the provided fields are meant to be used in the setting of goals. 2. Individual fields with zero value are meant NOT to be changed in the received element. Same as for missing fields in truncated messages. Furthermore, in order to improve testing, a combination of values for main and high goals are conventionally expected to be used for defining two behaviours: 1. Suspended: Tuning Probe messages (TTL=0) should be sent and statistics maintained, but the duty cycle should not be changed - thus goals will never be achieved. The encoding are: Main Goal = 0x00 and High Goal = 0xFE. 2. Disable: No Tuning Probe message should be sent and statistics should not be gathered - averaged values should decay. The encoding are: Main Goal = 0x00 and High Goal = 0xFF.

57.2 TuningModelClientInit Function Documentation

Syntax

```
CSRmeshResult TuningModelClientInit (CSRMESH_MODEL_CALLBACK_T app_callback)
```

Description

Parameters

Parameter	Description
app_callback	Pointer to the application callback function that will be called when the model client receives a message.

Returns

CSRmeshResult. Refer to CSRmeshResult.

57.3 TuningProbe Function Documentation

Syntax

```
CSRmeshResult TuningProbe (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8 ttl,
CSRMESH_TUNING_PROBE_T * p_params)
```

Description

This function packs the given parameters into a CSRmesh message and sends it over the network.

Parameters

Parameter	Description
nw_id	Network identifier over which the message has to be sent.
dest_id	16-bit identifier of the destination device/group
ttl	TTL value with which the message needs to be sent.
p_params	Pointer to the message parameters of type CSRMESH_TUNING_PROBE_T

Returns

CSRmeshResult. Refer to CSRmeshResult.

57.4 TuningGetStats Function Documentation

Syntax

```
CSRmeshResult TuningGetStats (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8 ttl,
CSRMESH_TUNING_GET_STATS_T * p_params)
```

Description

This function packs the given parameters into a CSRmesh message and sends it over the network. When a response is received the application callback function is called to notify the CSRMESH_TUNING_STATS

Parameters

Parameter	Description
nw_id	Network identifier over which the message has to be sent.
dest_id	16-bit identifier of the destination device/group
ttl	TTL value with which the message needs to be sent.
p_params	Pointer to the message parameters of type CSRMESH_TUNING_GET_STATS_T

Returns

CSRmeshResult. Refer to CSRmeshResult.

57.5 TuningSetConfig Function Documentation

Syntax

```
CSRmeshResult TuningSetConfig (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8 ttl,
CSRMESH_TUNING_SET_CONFIG_T * p_params)
```

Description

This function packs the given parameters into a CSRmesh message and sends it over the network. When a response is received the application callback function is called to notify the CSRMESH_TUNING_ACK_CONFIG

Parameters

Parameter	Description
nw_id	Network identifier over which the message has to be sent.
dest_id	16-bit identifier of the destination device/group
ttl	TTL value with which the message needs to be sent.
p_params	Pointer to the message parameters of type CSRMESH_TUNING_SET_CONFIG_T

Returns

CSRmeshResult. Refer to CSRmeshResult.

58 Server

Detailed Description

58.1 Data Structures of Server

```
struct  tuningStats_t
```

Tuning statistics of neighbouring devices.

58.2 Functions of Server

```
CSRmeshResult TuningModelInit (CsrUint8 nw_id, CsrUint16 *group_id_list,
CsrUint16 num_groups, CSRMESH_MODEL_CALLBACK_T app_callback)
```

Model Initialisation.

```
void TuningModelStart (uint16 probePeriod, uint16 reportPeriod)
```

Starts sending TUNING_PROBE messages and adjusts the scan duty cycle based on the analysis of the received messages.

```
void TuningModelStop (void)
```

Stops auto-tuning of rx duty cycle.

```
uint16 TuningReadStats (tuningStats_t **stats)
```

Returns the tuning statistics of the neighbouring devices.

58.3 TuningModelInit Function Documentation

Syntax

```
CSRmeshResult TuningModelInit (CsrUint8 nw_id, CsrUint16 * group_id_list,
CsrUint16 num_groups, CSRMESH_MODEL_CALLBACK_T app_callback)
```

Description

Registers the model handler with the CSRmesh. Sets the CSRmesh to report num_groups as the maximum number of groups supported for the model

Parameters

Parameter	Description
nw_id	Identifier of the network to which the model has to be registered.
group_id_list	Pointer to a uint16 array to hold assigned group_ids. This must be NULL if no groups are supported
num_groups	Size of the group_id_list. This must be 0 if no groups are supported.
app_callback	Pointer to the application callback function. This function will be called to notify all model specific messages

Returns

CSRmeshResult. Refer to CSRmeshResult.

58.4 TuningModelStart Function Documentation

Syntax

```
void TuningModelStart (uint16 probePeriod, uint16 reportPeriod)
```

Description

Returns

Nothing

58.5 TuningModelStop Function Documentation

Syntax

```
void TuningModelStop (void )
```

Description

This function stops the auto-tuning of the scan duty cycle. The scan duty cycle will not be restored to the original value set by the application. The application has to explicitly set the required duty cycle by calling the CsrMeshSetAdvScanParam when the tuning model is stopped.

Returns

Nothing

58.6 TuningReadStats Function Documentation

Syntax

```
uint16 TuningReadStats (tuningStats_t ** stats)
```

Description

Returns the tuning statistics of the neighbouring devices

Parameters

Parameter	Description
stats	Pointer to receive the tuning statistics buffer pointer.

Returns

Number of devices for which the statistics are collected.

59 Extension Model

Detailed Description

Extension Model API.

The Extension Model allows dynamic allocation of OpCode within a Mesh Network. This, permits extension of existing models without the risk of encoding clashes as per Data Model usage. A node wanting to introduce a new Model/Message or a new Message to an existing model, has to select an OpCode unique for that purpose. Given that there is no overall authority able to allocate a unique OpCode, this model proposes a collaborative method, within the existing Mesh Network, to obtain this unique information. Two messages are proposed, one performing a request, another informing of a conflict. As per the nature of Mesh Network, there is never an absolute guarantee of a solution, it is therefore necessary for the request to be repeated periodically. If anything, the periodic issue of this message will allow for new nodes to have the ability to collect the information. The request message has two purposes, indication of the intended OpCode and the mapping of this OpCode with the extension it is aimed at supplying. Identification of the function cover by the OpCode is done through an identity supplied by the manufacturer. In order to avoid the obvious denial of service through blocking of requests from specific manufacturers, the unique identity is calculated from the hashing of a sentence provided by the manufacturer. The one-way nature of the Hash makes it hard to reverse. For nodes intending to support these new messages, matching of the identity with the request, allows them to determine the OpCode to be used. Requests are relayed from node to node. Only when a node determine that the proposed Opcodes are conflicting with some internally used ones, the message will not be relayed, instead a conflict will be issued. Upon reception of a request, a node required to implement this function, takes note of the OpCode and starts processing messages with such OpCodes. Conflict messages carry the conflicting OpCodes and the reason for issuing of the conflict. Conflict messages are processed by all, thus allowing non-conflicting nodes to make of note of the fact that the OpCode is invalid and thus prevent their processing. Each node is expected to keep track of all the OpCodes it is currently using. These are either static or dynamic. A node wishing to introduce a new (model, message) pair can do so through a proposal to the Mesh Networks it belongs to, of the new range of Opcodes it intends to use.

59.1 Modules of Extension Model

Client

Server

59.2 Data Structures of Extension Model

`struct CSRMESH_EXTENSION_REQUEST_T`

CSRmesh Extension Model message types.

`struct CSRMESH_EXTENSION_CONFLICT_T`

Response to a REQUEST - only issued if a conflict is noticed. This message indicates that the proposed OpCode is already in use within the node processing the request message. Nodes receiving conflict extension will process this message and remove the conflicting OpCode from the list of OpCodes to handle. All conflict messages are relayed, processed or not. If a node receiving a REQUEST is able to match the hash of the provider previously assigned to an existing OpCode, but different to the proposed one, it responds with a CONFLICT with a reason combining the top bit with the previously associated range (0x80 | <old range>=""">). In such cases, the previously used OpCode (start of range) will be placed in the ProposedOpCode. Nodes receiving this conflict message with the top bit raised, will discard the initially proposed OpCode and replace it with the proposed code supplied in the conflict message.

60 Client

Detailed Description

60.1 Functions of Client

```
CSRmeshResult ExtensionModelClientInit (CSRMESH_MODEL_CALLBACK_T app_callback)
```

Initialises Extension Model Client functionality.

```
CSRmeshResult ExtensionRequest (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8
ttl, CSRMESH_EXTENSION_REQUEST_T *p_params)
```

Request for Extension OpCode to be approved by the whole Mesh. A device wanting to use an OpCode, makes a request to the entire Mesh Network. This message is issued to target identity 0. The device waits some time, proportional to the size of the Mesh network and only after this period, messages using these proposed OpCode are used. Device receiving this message and wanting to oppose the usage of such code will respond to the source node with a CONFLICT. In case no conflict is known and the OpCode is for a message the node is interested in implementing (through comparison with hash value), a record of the OpCode and its mapping is kept. Request messages are relayed in cases of absence of conflict. The hash function is SHA-256, padded as per SHA-256 specifications2, for which the least significant 6 bytes will be used in the message. The range parameter indicates the maximum number of OpCode reserved from the based provided in the Proposed OpCode field. The last OpCode reserved is determined through the sum of the Proposed OpCode with the range value. This range parameter varies from 0 to 127, leaving the top bit free.

```
CSRmeshResult ExtensionClientSetupOpcodeList (uint16 *opcode_list, CsrUint8
max_opcode_ranges)
```

Set Up Opcode List to be accessible by the extension model.

```
CSRmeshResult ExtensionSendMessage (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8
ttl, CsrUint8 *msg, CsrUint8 len)
```

Sends CSRmesh message with an extension opcode.

60.2 ExtensionModelClientInit Function Documentation

Syntax

```
CSRmeshResult ExtensionModelClientInit (CSRMESH_MODEL_CALLBACK_T app_callback)
```

Description

Parameters

Parameter	Description
app_callback	Pointer to the application callback function that will be called when the model client receives a message.

Returns

CSRmeshResult. Refer to CSRmeshResult.

60.3 ExtensionRequest Function Documentation

Syntax

```
CSRmeshResult ExtensionRequest (CsrUint8 nw_id, CsrUint16 dest_id,
CsrUint8 ttl, CSRMESH_EXTENSION_REQUEST_T * p_params)
```

Description

This function packs the given parameters into a CSRmesh message and sends it over the network. When a response is received the application callback function is called to notify the CSRMESH_EXTENSION_CONFLICT

Parameters

Parameter	Description
nw_id	Network identifier over which the message has to be sent.
dest_id	16-bit identifier of the destination device/group
ttl	TTL value with which the message needs to be sent.
p_params	Pointer to the message parameters of type CSRMESH_EXTENSION_REQUEST_T

Returns

CSRmeshResult. Refer to CSRmeshResult.

60.4 ExtensionClientSetupOpcodeList Function Documentation

Syntax

```
CSRmeshResult ExtensionClientSetupOpcodeList (uint16 * opcode_list,
CsrUint8 max_opcode_ranges)
```

Description

The function sets up a list to manage added opcodes through extension model. The function tells the extension model about the list of opcodes that is allocated for use with the extension model. The list must be a global variable and device and must accessible any time when the device is active.

Parameters

Parameter	Description
opcode_list	The List of the opcode, range and the hash string stored.
max_opcode_ranges	The maximum opcode ranges supported by application

Returns

CSRmeshResult. Refer to CSRmeshResult.

60.5 ExtensionSendMessage Function Documentation

Syntax

```
CSRmeshResult ExtensionSendMessage (CsrUint8 nw_id, CsrUint16 dest_id,
CsrUint8 ttl, CsrUint8 * msg, CsrUint8 len)
```

Description

This function sends a CSRmesh message to a destination device with a proposed extension opcode.

Parameters

Parameter	Description
nw_id	Network identifier over which the message has to be sent.
dest_id	16-bit identifier of the destination device/group

Parameter	Description
`ttl`	The time to live value to be set for the message
`msg`	Pointer to the message parameters
`len`	Length of the message

Returns

CSRmeshResult. Refer to CSRmeshResult.

61 Server

Detailed Description

61.1 Functions of Server

```
CSRmeshResult ExtensionModelInit (CsrUint8 nw_id, CsrUint16 *group_id_list,
CsrUint16 num_groups, CSRMESH_MODEL_CALLBACK_T app_callback)
```

Model Initialisation.

```
CSRmeshResult ExtensionConflict (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8
ttl, CSRMESH_EXTENSION_CONFLICT_T *p_params)
```

Response to a REQUEST - only issued if a conflict is noticed. This message indicates that the proposed OpCode is already in use within the node processing the request message. Nodes receiving conflict extension will process this message and remove the conflicting OpCode from the list of OpCodes to handle. All conflict messages are relayed, processed or not. If a node receiving a REQUEST is able to match the hash of the provider previously assigned to an existing OpCode, but different to the proposed one, it responds with a CONFLICT with a reason combining the top bit with the previously associated range (0x80 | <old range>=""). In such cases, the previously used OpCode (start of range) will be placed in the ProposedOpCode. Nodes receiving this conflict message with the top bit raised, will discard the initially proposed OpCode and replace it with the proposed code supplied in the conflict message.

```
bool ExtensionVerifyOpcodeConflictWithMesh (CsrUint16 opcode_start, CsrUint16
opcode_end)
```

Verify the opcodes passed Conflict with Mesh Opcodes.

```
CSRmeshResult ExtensionServerSetupOpcodeList (uint16 *opcode_list, CsrUint8
max_opcode_ranges)
```

Set Up Opcode List to be accessible by the extension model.

61.2 ExtensionModelInit Function Documentation

Syntax

```
CSRmeshResult ExtensionModelInit (CsrUint8 nw_id, CsrUint16 * group_id_list,
CsrUint16 num_groups, CSRMESH_MODEL_CALLBACK_T app_callback)
```

Description

Registers the model handler with the CSRmesh. Sets the CSRmesh to report num_groups as the maximum number of groups supported for the model

Parameters

Parameter	Description
nw_id	Identifier of the network to which the model has to be registered.
group_id_list	Pointer to a uint16 array to hold assigned group_ids. This must be NULL if no groups are supported
num_groups	Size of the group_id_list. This must be 0 if no groups are supported.
app_callback	Pointer to the application callback function. This function will be called to notify all model specific messages

Returns

CSRmeshResult. Refer to CSRmeshResult.

61.3 ExtensionConflict Function Documentation

Syntax

```
CSRmeshResult ExtensionConflict (CsrUint8 nw_id, CsrUint16 dest_id,
CsrUint8 ttl, CSRMESH_EXTENSION_CONFLICT_T * p_params)
```

Description

This function packs the given parameters into a CSRmesh message and sends it over the network.

Parameters

Parameter	Description
nw_id	Network identifier over which the message has to be sent.
dest_id	16-bit identifier of the destination device/group
ttl	TTL value with which the message needs to be sent.
p_params	Pointer to the message parameters of type CSRMESH_EXTENSION_CONFLICT_T

Returns

CSRmeshResult. Refer to CSRmeshResult.

61.4 ExtensionVerifyOpcodeConflictWithMesh Function Documentation

Syntax

```
bool ExtensionVerifyOpcodeConflictWithMesh (CsrUint16 opcode_start,
CsrUint16 opcode_end)
```

Description

The function checks whether the proposed opcodes conflict with the ones supported by the Mesh or not.

Parameters

Parameter	Description
opcode_start	Start of the opcode range that needs to be checked.
opcode_end	End of the opcode range that needs to be checked

Returns

TRUE if the opcodes conflict otherwise returns FALSE.

61.5 ExtensionServerSetupOpcodeList Function Documentation

Syntax

```
CSRmeshResult ExtensionServerSetupOpcodeList (uint16 * opcode_list,
CsrUint8 max_opcode_ranges)
```

Description

The function sets up a list to manage added opcodes through extension model. The function tells the extension model about the list of opcodes that is allocated for use with the extension model. The list must be a global variable and device and must accessible any time when the device is active.

Parameters

Parameter	Description
`opcode_list`	The List of the opcode, range and the hash string stored.
`max_opcode_ranges`	The maximum opcode ranges supported by application

Returns

CSRmeshResult. Refer to CSRmeshResult.

62 LargeObjectTransfer Model

Detailed Description

LargeObjectTransfer Model API.

Ability to exchange information between nodes such as to allow a peer-to-peer data transfer to be established after a negotiation phase.

62.1 Modules of LargeObjectTransfer Model

Client

Server

62.2 Data Structures of LargeObjectTransfer Model

`struct CSRMESH_LARGEOBJECTTRANSFER_ANNOUNCE_T`

CSRmesh LargeObjectTransfer Model message types.

`struct CSRMESH_LARGEOBJECTTRANSFER_INTEREST_T`

In case a node is ready to receive the proposed object, it responds with this message. The intended behaviour of the Large Object Transfer is to allow a Peer-to-Peer connection between the consumer and the producer. The consumer uses a ServiceID, part of which is randomly selected. The top 64 bits are 0x1122334455667788, the least significant 63 bits are randomly selected by the consumer node. The most significant bit of the least significant 64 bits is an encoding of the intent to relay the received data. Once this message has been issued, the consumer node starts advertising a connectable service with the 128-bits service composed through the concatenation of the fixed 64 bits and the randomly selected 63 bits. The duration of the advertisement is an implementation decision.

63 Client

Detailed Description

63.1 Functions of Client

```
CSRmeshResult LargeObjectTransferModelClientInit (CSRMESH_MODEL_CALLBACK_T
app_callback)
```

Initialises LargeObjectTransfer Model Client functionality.

```
CSRmeshResult LargeObjectTransferAnnounce (CsrUint8 nw_id, CsrUint16 dest_id,
CsrUint8 ttl, CSRMESH_LARGEOBJECTTRANSFER_ANNOUNCE_T *p_params)
```

A node wanting to provide a large object to neighbouring Mesh Nodes issues an ANNOUNCE with the associated content type. This message will have TTL=0, thus will only be answered by its immediate neighbours. The ANNOUNCE has the total size of the packet to be issued. The format and encoding of the large object is subject to the provided type and is out of scope of this document. The destination ID can either be 0, a group or a specific Device ID. In case the destination ID is not zero, only members of the group (associated with the LOT model) or the device with the specified Device ID responds with the intent to download the object for their own consumption. Every other node either ignores or accepts the offer for the purpose of relaying the packet.

63.2 LargeObjectTransferModelClientInit Function Documentation

Syntax

```
CSRmeshResult LargeObjectTransferModelClientInit
(CSRMESH_MODEL_CALLBACK_T app_callback)
```

Description

Parameters

Parameter	Description
app_callback	Pointer to the application callback function that will be called when the model client receives a message.

Returns

CSRmeshResult. Refer to CSRmeshResult.

63.3 LargeObjectTransferAnnounce Function Documentation

Syntax

```
CSRmeshResult LargeObjectTransferAnnounce (CsrUint8 nw_id, CsrUint16 dest_id,
CsrUint8 ttl, CSRMESH_LARGEOBJECTTRANSFER_ANNOUNCE_T * p_params)
```

Description

This function packs the given parameters into a CSRmesh message and sends it over the network. When a response is received the application callback function is called to notify the CSRMESH_LARGEOBJECTTRANSFER_INTEREST

Parameters

Parameter	Description
nw_id	Network identifier over which the message has to be sent.
dest_id	16-bit identifier of the destination device/group
ttl	TTL value with which the message needs to be sent.
p_params	Pointer to the message parameters of type CSRMESH_LARGEOBJECTTRANSFER_ANNOUNCE_T

Returns

CSRmeshResult. Refer to CSRmeshResult.

64 Server

Detailed Description

64.1 Functions of Server

```
CSRmeshResult LargeObjectTransferModelInit (CsrUint8 nw_id, CsrUint16
*group_id_list, CsrUint16 num_groups, CSRMESH_MODEL_CALLBACK_T app_callback)
```

Model Initialisation.

```
CSRmeshResult LargeObjectTransferInterest (CsrUint8 nw_id, CsrUint16 dest_id,
CsrUint8 ttl, CSRMESH_LARGEOBJECTTRANSFER_INTEREST_T *p_params)
```

In case a node is ready to receive the proposed object, it responds with this message. The intended behaviour of the Large Object Transfer is to allow a Peer-to-Peer connection between the consumer and the producer. The consumer uses a ServiceID, part of which is randomly selected. The top 64 bits are 0x1122334455667788, the least significant 63 bits are randomly selected by the consumer node. The most significant bit of the least significant 64 bits is an encoding of the intent to relay the received data. Once this message has been issued, the consumer node starts advertising a connectable service with the 128-bits service composed through the concatenation of the fixed 64 bits and the randomly selected 63 bits. The duration of the advertisement is an implementation decision.

64.2 LargeObjectTransferModelInit Function Documentation

Syntax

```
CSRmeshResult LargeObjectTransferModelInit (CsrUint8 nw_id, CsrUint16
* group_id_list, CsrUint16 num_groups, CSRMESH_MODEL_CALLBACK_T app_callback)
```

Description

Registers the model handler with the CSRmesh. Sets the CSRmesh to report num_groups as the maximum number of groups supported for the model

Parameters

Parameter	Description
nw_id	Identifier of the network to which the model has to be registered.
group_id_list	Pointer to a uint16 array to hold assigned group_ids. This must be NULL if no groups are supported
num_groups	Size of the group_id_list. This must be 0 if no groups are supported.
app_callback	Pointer to the application callback function. This function will be called to notify all model specific messages

Returns

CSRmeshResult. Refer to CSRmeshResult.

64.3 LargeObjectTransferInterest Function Documentation

Syntax

```
CSRmeshResult LargeObjectTransferInterest (CsrUint8 nw_id, CsrUint16 dest_id,
CsrUint8 ttl, CSRMESH_LARGEOBJECTTRANSFER_INTEREST_T * p_params)
```

Description

This function packs the given parameters into a CSRmesh message and sends it over the network.

Parameters

Parameter	Description
nw_id	Network identifier over which the message has to be sent.
dest_id	16-bit identifier of the destination device/group
ttl	TTL value with which the message needs to be sent.
p_params	Pointer to the message parameters of type CSRMESH_LARGEOBJECTTRANSFER_INTEREST_T

Returns

CSRmeshResult. Refer to CSRmeshResult.

65 Firmware Model

Detailed Description

Firmware Model API.

The Firmware Model allows a device to expose it's current firmware version revision and to accept new firmware downloads. There is no update method defined by this specification, this is left as an implementation detail for the firmware developers.

65.1 Modules of Firmware Model

Client

Server

65.2 Data Structures of Firmware Model

`struct CSRMESH_FIRMWARE_GET_VERSION_T`

CSRmesh Firmware Model message types.

`struct CSRMESH_FIRMWARE_VERSION_T`

Firmware version information.

`struct CSRMESH_FIRMWARE_UPDATE_REQUIRED_T`

Requesting a firmware update. Upon receiving this message, the device moves to a state where it is ready for receiving a firmware update.

`struct CSRMESH_FIRMWARE_UPDATE_ACKNOWLEDGED_T`

Acknowledgement message to the firmware update request.

66 Client

Detailed Description

66.1 Functions of Client

```
CSRmeshResult FirmwareModelClientInit (CSRMESH_MODEL_CALLBACK_T app_callback)
```

Initialises Firmware Model Client functionality.

```
CSRmeshResult FirmwareGetVersion (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8
ttl, CSRMESH_FIRMWARE_GET_VERSION_T *p_params)
```

Get firmwre verison: Upon receiving a FIRMWARE_GET_VERSION the device reponds with a FIRMWARE_VERSION message containing the current firmware version.

```
CSRmeshResult FirmwareUpdateRequired (CsrUint8 nw_id, CsrUint16 dest_id,
CsrUint8 ttl, CSRMESH_FIRMWARE_UPDATE_REQUIRED_T *p_params)
```

Requesting a firmware update. Upon receiving this message, the device moves to a state where it is ready for receiving a firmware update.

66.2 FirmwareModelClientInit Function Documentation

Syntax

```
CSRmeshResult FirmwareModelClientInit (CSRMESH_MODEL_CALLBACK_T app_callback)
```

Description

Parameters

Parameter	Description
app_callback	Pointer to the application callback function that will be called when the model client receives a message.

Returns

CSRmeshResult. Refer to CSRmeshResult.

66.3 FirmwareGetVersion Function Documentation

Syntax

```
CSRmeshResult FirmwareGetVersion (CsrUint8 nw_id, CsrUint16 dest_id,
CsrUint8 ttl, CSRMESH_FIRMWARE_GET_VERSION_T * p_params)
```

Description

This function packs the given parameters into a CSRmesh message and sends it over the network. When a response is received the application callback function is called to notify the CSRMESH_FIRMWARE_VERSION

Parameters

Parameter	Description
nw_id	Network identifier over which the message has to be sent.
dest_id	16-bit identifier of the destination device/group
ttl	TTL value with which the message needs to be sent.
p_params	Pointer to the message parameters of type CSRMESH_FIRMWARE_GET_VERSION_T

Returns

CSRmeshResult. Refer to CSRmeshResult.

66.4 FirmwareUpdateRequired Function Documentation

Syntax

```
CSRmeshResult FirmwareUpdateRequired (CsrUint8 nw_id, CsrUint16 dest_id,
CsrUint8 ttl, CSRMESH_FIRMWARE_UPDATE_REQUIRED_T * p_params)
```

Description

This function packs the given parameters into a CSRmesh message and sends it over the network. When a response is received the application callback function is called to notify the CSRMESH_FIRMWARE_UPDATE_ACKNOWLEDGED

Parameters

Parameter	Description
`nw_id`	Network identifier over which the message has to be sent.
`dest_id`	16-bit identifier of the destination device/group
`ttl`	TTL value with which the message needs to be sent.
`p_params`	Pointer to the message parameters of type CSRMESH_FIRMWARE_UPDATE_REQUIRED_T

Returns

CSRmeshResult. Refer to CSRmeshResult.

67 Server

Detailed Description

67.1 Functions of Server

```
CSRmeshResult FirmwareModelInit (CsrUint8 nw_id, CsrUint16 *group_id_list,
CsrUint16 num_groups, CSRMESH_MODEL_CALLBACK_T app_callback)
```

Model Initialisation.

```
CSRmeshResult FirmwareVersion (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8 ttl,
CSRMESH_FIRMWARE_VERSION_T *p_params)
```

Firmware version information.

```
CSRmeshResult FirmwareUpdateAcknowledged (CsrUint8 nw_id, CsrUint16 dest_id,
CsrUint8 ttl, CSRMESH_FIRMWARE_UPDATE_ACKNOWLEDGED_T *p_params)
```

Acknowledgement message to the firmware update request.

67.2 FirmwareModelInit Function Documentation

Syntax

```
CSRmeshResult FirmwareModelInit (CsrUint8 nw_id, CsrUint16 * group_id_list,
CsrUint16 num_groups, CSRMESH_MODEL_CALLBACK_T app_callback)
```

Description

Registers the model handler with the CSRmesh. Sets the CSRmesh to report num_groups as the maximum number of groups supported for the model

Parameters

Parameter	Description
`nw_id`	Identifier of the network to which the model has to be registered.
`group_id_list`	Pointer to a uint16 array to hold assigned group_ids. This must be NULL if no groups are supported

Parameter	Description
num_groups	Size of the group_id_list. This must be 0 if no groups are supported.
app_callback	Pointer to the application callback function. This function will be called to notify all model specific messages

Returns

CSRmeshResult. Refer to CSRmeshResult.

67.3 FirmwareVersion Function Documentation

Syntax

```
CSRmeshResult FirmwareVersion (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8 ttl,
CSRMESH_FIRMWARE_VERSION_T * p_params)
```

Description

This function packs the given parameters into a CSRmesh message and sends it over the network.

Parameters

Parameter	Description
nw_id	Network identifier over which the message has to be sent.
dest_id	16-bit identifier of the destination device/group
ttl	TTL value with which the message needs to be sent.
p_params	Pointer to the message parameters of type CSRMESH_FIRMWARE_VERSION_T

Returns

CSRmeshResult. Refer to CSRmeshResult.

67.4 FirmwareUpdateAcknowledged Function Documentation

Syntax

```
CSRmeshResult FirmwareUpdateAcknowledged (CsrUint8 nw_id, CsrUint16 dest_id,
CsrUint8 ttl, CSRMESH_FIRMWARE_UPDATE_ACKNOWLEDGED_T * p_params)
```

Description

This function packs the given parameters into a CSRmesh message and sends it over the network.

Parameters

Parameter	Description
`nw_id`	Network identifier over which the message has to be sent.
`dest_id`	16-bit identifier of the destination device/group
`ttl`	TTL value with which the message needs to be sent.
`p_params`	Pointer to the message parameters of type CSRMESH_FIRMWARE_UPDATE_ACKNOWLEDGED_T

Returns

CSRmeshResult. Refer to CSRmeshResult.

68 Diagnostic Model

Detailed Description

Diagnostic Model API.

This model permits the collection of local diagnostics for the purpose of accumulation of this information for further global data analysis. The diagnostic model is only about reporting information, how this information is gathered is beyond its scope. This model is limited to handling one single Network Key.

68.1 Modules of Diagnostic Model

Client

Server

68.2 Data Structures of Diagnostic Model

`struct CSRMESH_DIAGNOSTIC_STATE_T`

CSRmesh Diagnostic Model message types.

69 Client

Detailed Description

69.1 Functions of Client

```
CSRmeshResult DiagnosticModelClientInit (CSRMESH_MODEL_CALLBACK_T app_callback)
```

Initialises Diagnostic Model Client functionality.

```
CSRmeshResult DiagnosticState (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8 ttl,
CSRMESH_DIAGNOSTIC_STATE_T *p_params)
```

When received this message is interpreted as to reconfigure the set of information collected. Statistics gathering can be turned on/off ? in the off mode no measurement of messages count and RSSI measurements will be made. RSSI binning can be stored, such that collection ALL messages? RSSI (MASP/MCP, irrespective of encoding) are split between a given number of bin, each of equal dimensions. Masking of individual broadcast channel can be specified, resulting in the collection of information specifically on the selected channels. A REST bit is also available. When present all the accumulated data will be cleared and all counters restarted. Note that it is possible to change various configurations without the RESET flag, this will result in the continuation of accumulation and therefore incoherent statistics.

```
CSRmeshResult DiagnosticGetStats (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8
ttl, CSRMESH_DIAGNOSTIC_GET_STATS_T *p_params)
```

This function packs the given parameters into a CSRmesh message and sends it over the network. When a response is received the application callback function is called to notify the CSRMESH_DIAGNOSTIC_STATS.

69.2 DiagnosticModelClientInit Function Documentation

Syntax

```
CSRmeshResult DiagnosticModelClientInit (CSRMESH_MODEL_CALLBACK_T app_callback)
```

Description

Parameters

Parameter	Description
app_callback	Pointer to the application callback function that will be called when the model client receives a message.

Returns

CSRmeshResult. Refer to CSRmeshResult.

69.3 DiagnosticState Function Documentation

Syntax

```
CSRmeshResult DiagnosticState (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8 ttl,
CSRMESH_DIAGNOSTIC_STATE_T * p_params)
```

Description

This function packs the given parameters into a CSRmesh message and sends it over the network.

Parameters

Parameter	Description
nw_id	Network identifier over which the message has to be sent.
dest_id	16-bit identifier of the destination device/group
ttl	TTL value with which the message needs to be sent.
p_params	Pointer to the message parameters of type CSRMESH_DIAGNOSTIC_STATE_T

Returns

CSRmeshResult. Refer to CSRmeshResult.

69.4 DiagnosticGetStats Function Documentation

Syntax

```
CSRmeshResult DiagnosticGetStats (CsrUint8 nw_id, CsrUint16 dest_id,
CsrUint8 ttl, CSRMESH_DIAGNOSTIC_GET_STATS_T * p_params)
```

Description

Parameters

Parameter	Description
`nw_id`	Network identifier over which the message has to be sent.
`dest_id`	16-bit identifier of the destination device/group
`ttl`	TTL value with which the message needs to be sent.
`p_params`	Pointer to the message parameters of type CSRMESH_DIAGNOSTIC_GET_STATS_T

Returns

CSRmeshResult. Refer to CSRmeshResult.

70 Server

Detailed Description

70.1 Functions of Server

```
CSRmeshResult DiagnosticModelInit (CsrUint8 nw_id, CsrUint16 *group_id_list,
CsrUint16 num_groups, CSRMESH_MODEL_CALLBACK_T app_callback)
```

Model Initialisation.

```
CSRmeshResult DiagnosticStats (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8 ttl,
CSRMESH_DIAGNOSTIC_STATS_T *p_params)
```

This function packs the given parameters into a CSRmesh message and sends it over the network.

70.2 DiagnosticModelInit Function Documentation

Syntax

```
CSRmeshResult DiagnosticModelInit (CsrUint8 nw_id, CsrUint16 * group_id_list,
CsrUint16 num_groups, CSRMESH_MODEL_CALLBACK_T app_callback)
```

Description

Registers the model handler with the CSRmesh. Sets the CSRmesh to report num_groups as the maximum number of groups supported for the model

Parameters

Parameter	Description
`nw_id`	Identifier of the network to which the model has to be registered.
`group_id_list`	Pointer to a uint16 array to hold assigned group_ids. This must be NULL if no groups are supported

Parameter	Description
num_groups	Size of the group_id_list. This must be 0 if no groups are supported.
app_callback	Pointer to the application callback function. This function will be called to notify all model specific messages

Returns

CSRmeshResult. Refer to CSRmeshResult.

70.3 DiagnosticStats Function Documentation

Syntax

```
CSRmeshResult DiagnosticStats (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8 ttl,
CSRMESH_DIAGNOSTIC_STATS_T * p_params)
```

Description

Parameters

Parameter	Description
nw_id	Network identifier over which the message has to be sent.
dest_id	16-bit identifier of the destination device/group
ttl	TTL value with which the message needs to be sent.
p_params	Pointer to the message parameters of type CSRMESH_DIAGNOSTIC_STATS_T

Returns

CSRmeshResult. Refer to CSRmeshResult.

71 Action Model

Detailed Description

Action Model API.

71.1 Modules of Action Model

Client

Server

71.2 Data Structures of Action Model

`struct CSRMESH_ACTION_SET_ACTION_INFO_T`

CSRmesh Action Model message types.

`struct CSRMESH_ACTION_SET_ACTION_INFO_EXT_T`

This message defines the action to be sent when GET_ACTION is received. This is a multiple parts message, each part will be acknowledged through an ACTION_SET_ACK (0x51).This also contains the buffer to send the action message through it.

`struct CSRMESH_ACTION_SET_ACTION_T`

This message defines the action which needs to be taken. This is a multiple parts message, each part will be acknowledged through an ACTION_SET_ACK (0x51).

`struct CSRMESH_ACTION_SET_ACTION_ACK_T`

Every parts received is acknowledged, providing both Transaction ID and ActionID (which includes Part Number)

`struct CSRMESH_ACTION_GET_ACTION_STATUS_T`

Request towards a node about the various Actions currently handled. This will be answered through an ACTION_STATUS. A Transaction ID is provided to ensure identification of response with request.

`struct CSRMESH_ACTION_ACTION_STATUS_T`

This provides a bitmask identifying all the currently allocated ActionID. The Action Supported field provides an indication on how many actions this device has capacity for. Note that a device will never have more than 32 actions.

`struct CSRMESH_ACTION_DELETE_T`

This message allows a set of actions to be removed. Actions are identified through a bitmask reflecting the ActionID one wishes to delete. This message will be acknowledged through ACTION_DELETE_ACK.

`struct CSRMESH_ACTION_DELETE_ACK_T`

This message confirms the delete commands and report which actions have been removed. The provided transaction ID will match the one supplied by ACTION_DELETE.

`struct CSRMESH_ACTION_GET_T`

Using this message, it is possible to interrogate a node about the specific payload associated with an Action ID. This will be answered by and ACTION_SET_ACTION message, ACTION_SET_ACTION_ACK will need to be issued in order to collect all parts.

72 Client

Detailed Description

72.1 Functions of Client

```
CSRmeshResult ActionModelClientInit (CSRMESH_MODEL_CALLBACK_T app_callback)
```

Initialises Action Model Client functionality.

```
CSRmeshResult ActionClientPrepareAction (void)
```

To set an action on a node the following functions should be called in sequence. ActionClientPrepareAction() followed by the model API whose action need to be set followed by ActionClientSetAction(). ActionClientPrepareAction() function indicates the action model and the mesh stack that next model message being prepared is for action set message and would not be transmitted over the air.

```
CSRmeshResult ActionClientSetAction (CsrUint8 nw_id, CsrUint16 dest_id,
CsrUint8 ttl, CSRMESH_ACTION_SET_ACTION_INFO_T *p_params)
```

To set an action on a node the following functions should be called in sequence. ActionClientPrepareAction() followed by the model API whose action need to be set followed by ActionClientSetAction(). ActionClientSetAction() function forms the action set msgs with the action characteristics and the model message API previously called and sends it over the network onto the node with dest_id address. This is a multiple part message, where each part will be acknowledged through an ACTION_SET_ACK (0x51).

```
CSRmeshResult ActionGetActionStatus (CsrUint8 nw_id, CsrUint16 dest_id,
CsrUint8 ttl, CSRMESH_ACTION_GET_ACTION_STATUS_T *p_params)
```

Request towards a node about the various Actions currently handled. This will be answered through an ACTION_STATUS. A Transaction ID is provided to ensure identification of response with request.

```
CSRmeshResult ActionDelete (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8 ttl,
CSRMESH_ACTION_DELETE_T *p_params)
```

This message allows a set of actions to be removed. Actions are identified through a bitmask reflecting the ActionID one wishes to delete. This message will be acknowledged through ACTION_DELETE_ACK.

```
CSRmeshResult ActionGet (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8 ttl,
CSRMESH_ACTION_GET_T *p_params)
```

Using this message, it is possible to interrogate a node about the specific payload associated with an Action ID. This will be answered by and ACTION_SET_ACTION message, ACTION_SET_ACTION_ACK will need to be issued in order to collect all parts.

72.2 ActionModelClientInit Function Documentation

Syntax

```
CSRmeshResult ActionModelClientInit (CSRMESH_MODEL_CALLBACK_T app_callback)
```

Description

Parameters

Parameter	Description
app_callback	Pointer to the application callback function that will be called when the model client receives a message.

Returns

CSRmeshResult. Refer to CSRmeshResult.

72.3 ActionClientPrepareAction Function Documentation

Syntax

```
CSRmeshResult ActionClientPrepareAction (void )
```

Description

This function prepares the action model and the mesh stack such that the next call to the model API would be for setting the action and not to be sent over the network.

Returns

CSRmeshResult. Refer to CSRmeshResult.

72.4 ActionClientSetAction Function Documentation

Syntax

```
CSRmeshResult ActionClientSetAction (CsrUint8 nw_id, CsrUint16 dest_id,
CsrUint8 ttl, CSRMESH_ACTION_SET_ACTION_INFO_T * p_params)
```

Description

This function forms the action set msgs with the action characteristics and the model message API previously called and sends it over the network onto the node with dest_id address. This is a multiple part message, where each part will be acknowledged through an CSRMESH_ACTION_SET_ACTION_ACK (0x51).

When a response is received the application callback function is called to notify the CSRMESH_ACTION_SET_ACTION_ACK

Parameters

Parameter	Description
nw_id	Network identifier over which the message has to be sent.
dest_id	16-bit identifier of the destination device/group
ttl	TTL value with which the message needs to be sent.
p_params	Pointer to the message parameters of type CSRMESH_ACTION_SET_ACTION_INFO_T

Returns

CSRmeshResult. Refer to CSRmeshResult.

72.5 ActionGetActionStatus Function Documentation

Syntax

```
CSRmeshResult ActionGetActionStatus (CsrUint8 nw_id, CsrUint16 dest_id,
CsrUint8 ttl, CSRMESH_ACTION_GET_ACTION_STATUS_T * p_params)
```

Description

This function packs the given parameters into a CSRmesh message and sends it over the network. When a response is received the application callback function is called to notify the CSRMESH_ACTION_ACTION_STATUS

Parameters

Parameter	Description
nw_id	Network identifier over which the message has to be sent.
dest_id	16-bit identifier of the destination device/group

Parameter	Description
ttl	TTL value with which the message needs to be sent.
p_params	Pointer to the message parameters of type CSRMESH_ACTION_GET_ACTION_STATUS_T

Returns

CSRmeshResult. Refer to CSRmeshResult.

72.6 ActionDelete Function Documentation

Syntax

```
CSRmeshResult ActionDelete (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8 ttl,
CSRMESH_ACTION_DELETE_T * p_params)
```

Description

This function packs the given parameters into a CSRmesh message and sends it over the network. When a response is received the application callback function is called to notify the CSRMESH_ACTION_DELETE_ACK

Parameters

Parameter	Description
nw_id	Network identifier over which the message has to be sent.
dest_id	16-bit identifier of the destination device/group
ttl	TTL value with which the message needs to be sent.
p_params	Pointer to the message parameters of type CSRMESH_ACTION_DELETE_T

Returns

CSRmeshResult. Refer to CSRmeshResult.

72.7 ActionGet Function Documentation

Syntax

```
CSRmeshResult ActionGet (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8 ttl,
CSRMESH_ACTION_GET_T * p_params)
```

Description

This function packs the given parameters into a CSRmesh message and sends it over the network. When a response is received the application callback function is called to notify the CSRMESH_ACTION_SET_ACTION

Parameters

Parameter	Description
nw_id	Network identifier over which the message has to be sent.
dest_id	16-bit identifier of the destination device/group
ttl	TTL value with which the message needs to be sent.
p_params	Pointer to the message parameters of type CSRMESH_ACTION_GET_T

Returns

CSRmeshResult. Refer to CSRmeshResult.

73 Server

Detailed Description

73.1 Functions of Server

```
CSRmeshResult ActionModelInit (CsrUint8 nw_id, CsrUint16 *group_id_list,
CsrUint16 num_groups, CSRMESH_MODEL_CALLBACK_T app_callback)
```

Model Initialisation.

```
CSRmeshResult ActionPrepareAction (void)
```

To set an action on a node the following functions should be called in sequence. ActionPrepareAction() followed by the model API whose action need to be set followed by ActionSetAction(). ActionPrepareAction function indicates the action model and the mesh stack that next model message being prepared is for action set message and would not be transmitted over the air.

```
CSRmeshResult ActionSetAction (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8 ttl,
CSRMESH_ACTION_SET_ACTION_INFO_T *p_params)
```

To set an action on a node the following functions should be called in sequence. ActionPrepareAction() followed by the model API whose action need to be set followed by ActionSetAction(). ActionSetAction() function forms the action set msgs with the action characteristics and the model message API previously called and sends it over the network onto the node with dest_id address. This is a multiple part message, where each part will be acknowledged through an ACTION_SET_ACK (0x51).

```
CSRmeshResult ActionSetActionAck (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8
ttl, CSRMESH_ACTION_SET_ACTION_ACK_T *p_params)
```

Every parts received is acknowledged, providing both Transaction ID and ActionID (which includes Part Number)

```
CSRmeshResult ActionActionStatus (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8
ttl, CSRMESH_ACTION_ACTION_STATUS_T *p_params)
```

This provides a bitmask identifying all the currently allocated ActionID. The Action Supported field provides an indication on how many actions this device has capacity for. Note that a device will never have more than 32 actions.

```
CSRmeshResult ActionDeleteAck (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8 ttl,
CSRMESH_ACTION_DELETE_ACK_T *p_params)
```

This message confirms the delete commands and report which actions have been removed. The provided transaction ID will match the one supplied by ACTION_DELETE.

```
CSRmeshResult ActionSendMessage (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8
ttl, CsrUint8 *msg, CsrUint8 len)
```

Sends CSRmesh message that was received as an action.

73.2 ActionModelInit Function Documentation

Syntax

```
CSRmeshResult ActionModelInit (CsrUint8 nw_id, CsrUint16 * group_id_list,
CsrUint16 num_groups, CSRMESH_MODEL_CALLBACK_T app_callback)
```

Description

Registers the model handler with the CSRmesh. Sets the CSRmesh to report num_groups as the maximum number of groups supported for the model

Parameters

Parameter	Description
nw_id	Identifier of the network to which the model has to be registered.
group_id_list	Pointer to a uint16 array to hold assigned group_ids. This must be NULL if no groups are supported
num_groups	Size of the group_id_list. This must be 0 if no groups are supported.
app_callback	Pointer to the application callback function. This function will be called to notify all model specific messages

Returns

CSRmeshResult. Refer to CSRmeshResult.

73.3 ActionPrepareAction Function Documentation

Syntax

```
CSRmeshResult ActionPrepareAction (void )
```

Description

This function prepares the action model and the mesh stack such that the next call to the model API would be for setting the action and not to be sent over the network.

Returns

CSRmeshResult. Refer to CSRmeshResult.

73.4 ActionSetAction Function Documentation

Syntax

```
CSRmeshResult ActionSetAction (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8 ttl,
CSRMESH_ACTION_SET_ACTION_INFO_T * p_params)
```

Description

This function forms the action set msgs with the action characteristics and the model message API previously called and sends it over the network onto the node with dest_id address. This is a multiple part message, where each part will be acknowledged through an CSRMESH_ACTION_SET_ACTION_ACK (0x51).

When a response is received the application callback function is called to notify the CSRMESH_ACTION_SET_ACTION_ACK

Parameters

Parameter	Description
nw_id	Network identifier over which the message has to be sent.
dest_id	16-bit identifier of the destination device/group
ttl	TTL value with which the message needs to be sent.
p_params	Pointer to the message parameters of type CSRMESH_ACTION_SET_ACTION_INFO_T

Returns

CSRmeshResult. Refer to CSRmeshResult.

73.5 ActionSetActionAck Function Documentation

Syntax

```
CSRmeshResult ActionSetActionAck (CsrUint8 nw_id, CsrUint16 dest_id,
CsrUint8 ttl, CSRMESH_ACTION_SET_ACTION_ACK_T * p_params)
```

Description

This function packs the given parameters into a CSRmesh message and sends it over the network.

Parameters

Parameter	Description
nw_id	Network identifier over which the message has to be sent.
dest_id	16-bit identifier of the destination device/group
ttl	TTL value with which the message needs to be sent.
p_params	Pointer to the message parameters of type CSRMESH_ACTION_SET_ACTION_ACK_T

Returns

CSRmeshResult. Refer to CSRmeshResult.

73.6 ActionActionStatus Function Documentation

Syntax

```
CSRmeshResult ActionActionStatus (CsrUint8 nw_id, CsrUint16 dest_id,
CsrUint8 ttl, CSRMESH_ACTION_ACTION_STATUS_T * p_params)
```

Description

This function packs the given parameters into a CSRmesh message and sends it over the network.

Parameters

Parameter	Description
nw_id	Network identifier over which the message has to be sent.
dest_id	16-bit identifier of the destination device/group
ttl	TTL value with which the message needs to be sent.
p_params	Pointer to the message parameters of type CSRMESH_ACTION_ACTION_STATUS_T

Returns

CSRmeshResult. Refer to CSRmeshResult.

73.7 ActionDeleteAck Function Documentation

Syntax

```
CSRmeshResult ActionDeleteAck (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8 ttl,
CSRMESH_ACTION_DELETE_ACK_T * p_params)
```

Description

This function packs the given parameters into a CSRmesh message and sends it over the network.

Parameters

Parameter	Description
nw_id	Network identifier over which the message has to be sent.
dest_id	16-bit identifier of the destination device/group
ttl	TTL value with which the message needs to be sent.
p_params	Pointer to the message parameters of type CSRMESH_ACTION_DELETE_ACK_T

Returns

CSRmeshResult. Refer to CSRmeshResult.

73.8 ActionSendMessage Function Documentation

Syntax

```
CSRmeshResult ActionSendMessage (CsrUint8 nw_id, CsrUint16 dest_id,
CsrUint8 ttl, CsrUint8 * msg, CsrUint8 len)
```

Description

This function sends a CSRmesh message to a destination device as set by the CSRMESH_ACTION_SET message.

Parameters

Parameter	Description
nw_id	Network identifier over which the message has to be sent.
dest_id	16-bit identifier of the destination device/group
p_params	Pointer to the message parameters of type CSRMESH_ACTION_ACTION_STATUS_T

Returns

CSRmeshResult. Refer to CSRmeshResult.

74 Beacon Model

Detailed Description

Beacon Model API.

This model allows managing beacons of various types (CSR retail beacons, Apple iBeacons, Google URI or Eddystone beacons,LTE direct beacons and so on). Beacons will normally be operating as beacons only, but will periodically turn into mesh devices for a duration of time.

74.1 Modules of Beacon Model

Client

Server

74.2 Data Structures of Beacon Model

`struct CSRMESH_BEACON_SET_STATUS_T`

CSRmesh Beacon Model message types.

`struct CSRMESH_BEACON_GET_BEACON_STATUS_T`

Message to be sent to a beacon in order to recover its current status. This message fetch information pertinent to a particular type. The transaction ID will help matching the pertinent BEACON_STATUS message: a different transaction ID shall be used for a different type (although the BEACON_STATUS message has the type pertinent to the status and thus could be used in conjunction to the Transaction ID for disambiguation).

`struct CSRMESH_BEACON_BEACON_STATUS_T`

This message issue issued by a Beacon as response to BEACON_SET_STATUS or BEACON_GET_STATUS. Furthermore, a Beacon appearing on the mesh sporadically, will issue such message (with destination ID set to 0) as soon as it re-joins the mesh. In this case, one message per active beacon type should be issued. This message will provide the Payload ID currently in used for the associated type.

`struct CSRMESH_BEACON_GET_TYPES_T`

Message allowing a node to fetch all supported types and associated information by a given Beacon.

`struct CSRMESH_BEACON_TYPES_T`

Provides information on the set of beacon supported by the node. The battery level is reported as well as time since last message.

```
struct  CSRMESH_BEACON_SET_PAYLOAD_T
```

One or more of these packets may be sent to a beacon to set its payload. The content is either the raw advert data, or extra information (such as crypto cycle) which a beacon may require. The payload data is sent as a length and an offset, so the whole payload need not be sent if it has not changed. A beacon can support many beacon types - it can be sent as many different payloads as needed, one for each type. The first byte of the first payload packet contains the length and offset of the payload and the payload ID; this allows a beacon which already has the payload to send an immediate acknowledgement, saving traffic. This ID will be sent back in the 'Payload ACK' message if the beacon has received the whole payload. The payload may have to be split in several parts, in which case only the last part shall be acknowledged. Upon receiving a BEACON_SET_PAYLOAD, the beacon will update its corresponding beacon data: if data was previously available, it will be replaced by the provided payload, thus a beacon can only hold one payload per beacon type.

```
struct  CSRMESH_BEACON_PAYLOAD_ACK_T
```

Only one Acknowledgement message will occur for multiple BEACON_SET_PAYLOAD messages sharing the same transaction ID. Only when the last segment is received that such acknowledgement will be issued. Where missing payload messages exist, the list of their indices will be provided in the Ack field.

```
struct  CSRMESH_BEACON_GET_PAYLOAD_T
```

Message allowing a node to retrieve the current payload held on a given beacon. This message shall be answered by one or more BEACON_SET_PAYLOAD messages.

75 Client

Detailed Description

75.1 Functions of Client

```
CSRmeshResult BeaconModelClientInit (CSRMESH_MODEL_CALLBACK_T app_callback)
```

Initialises Beacon Model Client functionality.

```
CSRmeshResult BeaconSetStatus (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8 ttl,
CSRMESH_BEACON_SET_STATUS_T *p_params)
```

May be sent to a beacon to set its status. More than one such command can be sent to set the intervals for different beacon types, which need not be the same. This message also allows for the wakeup intervals to be set if the beacon elects to only be on the mesh sporadically. The time is always with respect to the beacon's internal clock and has no absolute meaning. This message will be answered through BEACON_STATUS.

```
CSRmeshResult BeaconGetBeaconStatus (CsrUint8 nw_id, CsrUint16 dest_id,
CsrUint8 ttl, CSRMESH_BEACON_GET_BEACON_STATUS_T *p_params)
```

Message to be sent to a beacon in order to recover its current status. This message fetch information pertinent to a particular type. The transaction ID will help matching the pertinent BEACON_STATUS message: a different transaction ID shall be used for a different type (although the BEACON_STATUS message has the type pertinent to the status and thus could be used in conjunction to the Transaction ID for disambiguation).

```
CSRmeshResult BeaconGetTypes (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8 ttl,
CSRMESH_BEACON_GET_TYPES_T *p_params)
```

Message allowing a node to fetch all supported types and associated information by a given Beacon.

```
CSRmeshResult BeaconClientSetPayload (CsrUint8 nw_id, CsrUint16 dest_id,
CsrUint8 ttl, CSRMESH_BEACON_SET_PAYLOAD_T *p_params)
```

One or more of these packets may be sent to a beacon to set its payload. The content is either the raw advert data, or extra information (such as crypto cycle) which a beacon may require. The payload data is sent as a length and an offset, so the whole payload need not be sent if it has not changed. A beacon can support many beacon types - it can be sent as many different payloads as needed, one for each type. The first byte of the first payload packet contains the length and offset of the payload and the payload ID; this allows a beacon which already has the payload to send an immediate acknowledgement, saving traffic. This ID will be sent back in the 'Payload ACK' message if the beacon has received the whole payload. The payload may have to be split in several parts, in which case only the last part shall be acknowledged. Upon receiving a BEACON_SET_PAYLOAD, the beacon will update its corresponding beacon data: if data was previously available, it will be replaced by the provided payload, thus a beacon can only hold one payload per beacon type.

```
CSRmeshResult BeaconClientPayloadAck (CsrUint8 nw_id, CsrUint16 dest_id,
CsrUint8 ttl, CSRMESH_BEACON_PAYLOAD_ACK_T *p_params)
```

Only one Acknowledgement message will occur for multiple BEACON_SET_PAYLOAD messages sharing the same transaction ID. Only when the last segment is received that such acknowledgement will be issued. Where missing payload messages exist, the list of their indices will be provided in the Ack field.

```
CSRmeshResult BeaconGetPayload (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8
ttl, CSRMESH_BEACON_GET_PAYLOAD_T *p_params)
```

Message allowing a node to retrieve the current payload held on a given beacon. This message shall be answered by one or more BEACON_SET_PAYLOAD messages.

75.2 BeaconModelClientInit Function Documentation

Syntax

```
CSRmeshResult BeaconModelClientInit (CSRMESH_MODEL_CALLBACK_T app_callback)
```

Description

Parameters

Parameter	Description
app_callback	Pointer to the application callback function that will be called when the model client receives a message.

Returns

CSRmeshResult. Refer to CSRmeshResult.

75.3 BeaconSetStatus Function Documentation

Syntax

```
CSRmeshResult BeaconSetStatus (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8 ttl,
CSRMESH_BEACON_SET_STATUS_T * p_params)
```

Description

This function packs the given parameters into a CSRmesh message and sends it over the network. When a response is received the application callback function is called to notify the CSRMESH_BEACON_BEACON_STATUS

Parameters

Parameter	Description
nw_id	Network identifier over which the message has to be sent.
dest_id	16-bit identifier of the destination device/group
ttl	TTL value with which the message needs to be sent.
p_params	Pointer to the message parameters of type CSRMESH_BEACON_SET_STATUS_T

Returns

CSRmeshResult. Refer to CSRmeshResult.

75.4 BeaconGetBeaconStatus Function Documentation

Syntax

```
CSRmeshResult BeaconGetBeaconStatus (CsrUint8 nw_id, CsrUint16 dest_id,
CsrUint8 ttl, CSRMESH_BEACON_GET_BEACON_STATUS_T * p_params)
```

Description

This function packs the given parameters into a CSRmesh message and sends it over the network. When a response is received the application callback function is called to notify the CSRMESH_BEACON_BEACON_STATUS

Parameters

Parameter	Description
nw_id	Network identifier over which the message has to be sent.
dest_id	16-bit identifier of the destination device/group
ttl	TTL value with which the message needs to be sent.
p_params	Pointer to the message parameters of type CSRMESH_BEACON_GET_BEACON_STATUS_T

Returns

CSRmeshResult. Refer to CSRmeshResult.

75.5 BeaconGetTypes Function Documentation

Syntax

```
CSRmeshResult BeaconGetTypes (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8 ttl,
CSRMESH_BEACON_GET_TYPES_T * p_params)
```

Description

This function packs the given parameters into a CSRmesh message and sends it over the network. When a response is received the application callback function is called to notify the CSRMESH_BEACON_TYPES

Parameters

Parameter	Description
nw_id	Network identifier over which the message has to be sent.
dest_id	16-bit identifier of the destination device/group
ttl	TTL value with which the message needs to be sent.
p_params	Pointer to the message parameters of type CSRMESH_BEACON_GET_TYPES_T

Returns

CSRmeshResult. Refer to CSRmeshResult.

75.6 BeaconClientSetPayload Function Documentation

Syntax

```
CSRmeshResult BeaconClientSetPayload (CsrUint8 nw_id, CsrUint16 dest_id,
CsrUint8 ttl, CSRMESH_BEACON_SET_PAYLOAD_T * p_params)
```

Description

This function packs the given parameters into a CSRmesh message and sends it over the network. When a response is received the application callback function is called to notify the CSRMESH_BEACON_PAYLOAD_ACK

Parameters

Parameter	Description
nw_id	Network identifier over which the message has to be sent.
dest_id	16-bit identifier of the destination device/group
ttl	TTL value with which the message needs to be sent.
p_params	Pointer to the message parameters of type CSRMESH_BEACON_SET_PAYLOAD_T

Returns

CSRmeshResult. Refer to CSRmeshResult.

75.7 BeaconClientPayloadAck Function Documentation

Syntax

```
CSRmeshResult BeaconClientPayloadAck (CsrUint8 nw_id, CsrUint16 dest_id,
CsrUint8 ttl, CSRMESH_BEACON_PAYLOAD_ACK_T * p_params)
```

Description

This function packs the given parameters into a CSRmesh message and sends it over the network.

Parameters

Parameter	Description
nw_id	Network identifier over which the message has to be sent.
dest_id	16-bit identifier of the destination device/group
ttl	TTL value with which the message needs to be sent.
p_params	Pointer to the message parameters of type CSRMESH_BEACON_PAYLOAD_ACK_T

Returns

CSRmeshResult. Refer to CSRmeshResult.

75.8 BeaconGetPayload Function Documentation

Syntax

```
CSRmeshResult BeaconGetPayload (CsrUint8 nw_id, CsrUint16 dest_id,
CsrUint8 ttl, CSRMESH_BEACON_GET_PAYLOAD_T * p_params)
```

Description

This function packs the given parameters into a CSRmesh message and sends it over the network. When a response is received the application callback function is called to notify the CSRMESH_BEACON_SET_PAYLOAD

Parameters

Parameter	Description
nw_id	Network identifier over which the message has to be sent.
dest_id	16-bit identifier of the destination device/group
ttl	TTL value with which the message needs to be sent.
p_params	Pointer to the message parameters of type CSRMESH_BEACON_GET_PAYLOAD_T

Returns

CSRmeshResult. Refer to CSRmeshResult.

76 Server

Detailed Description

76.1 Functions of Server

```
CSRmeshResult BeaconModelInit (CsrUint8 nw_id, CsrUint16 *group_id_list,
CsrUint16 num_groups, CSRMESH_MODEL_CALLBACK_T app_callback)
```

Model Initialisation.

```
CSRmeshResult BeaconBeaconStatus (CsrUint8 nw_id, CsrUint16 src_id, CsrUint16
dest_id, CsrUint8 ttl, CSRMESH_BEACON_BEACON_STATUS_T *p_params)
```

This message issue issued by a Beacon as response to BEACON_SET_STATUS or BEACON_GET_STATUS. Furthermore, a Beacon appearing on the mesh sporadically, will issue such message (with destination ID set to 0) as soon as it re-joins the mesh. In this case, one message per active beacon type should be issued. This message will provide the Payload ID currently in used for the associated type.

```
CSRmeshResult BeaconTypes (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8 ttl,
CSRMESH_BEACON_TYPES_T *p_params)
```

Provides information on the set of beacon supported by the node. The battery level is reported as well as time since last message.

```
CSRmeshResult BeaconSetPayload (CsrUint8 nw_id, CsrUint16 src_id, CsrUint16
dest_id, CsrUint8 ttl, CSRMESH_BEACON_SET_PAYLOAD_T *p_params)
```

One or more of these packets may be sent to a beacon to set its payload. The content is either the raw advert data, or extra information (such as crypto cycle) which a beacon may require. The payload data is sent as a length and an offset, so the whole payload need not be sent if it has not changed. A beacon can support many beacon types - it can be sent as many different payloads as needed, one for each type. The first byte of the first payload packet contains the length and offset of the payload and the payload ID; this allows a beacon which already has the payload to send an immediate acknowledgement, saving traffic. This ID will be sent back in the 'Payload ACK' message if the beacon has received the whole payload. The payload may have to be split in several parts, in which case only the last part shall be acknowledged. Upon receiving a BEACON_SET_PAYLOAD, the beacon will update its corresponding beacon data: if data was previously available, it will be replaced by the provided payload, thus a beacon can only hold one payload per beacon type.

```
CSRmeshResult BeaconPayloadAck (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8
ttl, CSRMESH_BEACON_PAYLOAD_ACK_T *p_params)
```

Only one Acknowledgement message will occur for multiple BEACON_SET_PAYLOAD messages sharing the same transaction ID. Only when the last segment is received that such acknowledgement will be issued. Where missing payload messages exist, the list of their indices will be provided in the Ack field.

76.2 BeaconModelInit Function Documentation

Syntax

```
CSRmeshResult BeaconModelInit (CsrUint8 nw_id, CsrUint16 * group_id_list,
CsrUint16 num_groups, CSRMESH_MODEL_CALLBACK_T app_callback)
```

Description

Registers the model handler with the CSRmesh. Sets the CSRmesh to report num_groups as the maximum number of groups supported for the model

Parameters

Parameter	Description
nw_id	Identifier of the network to which the model has to be registered.
group_id_list	Pointer to a uint16 array to hold assigned group_ids. This must be NULL if no groups are supported
num_groups	Size of the group_id_list. This must be 0 if no groups are supported.
app_callback	Pointer to the application callback function. This function will be called to notify all model specific messages

Returns

CSRmeshResult. Refer to CSRmeshResult.

76.3 BeaconBeaconStatus Function Documentation

Syntax

```
CSRmeshResult BeaconBeaconStatus (CsrUint8 nw_id, CsrUint16 src_id,
CsrUint16 dest_id, CsrUint8 ttl, CSRMESH_BEACON_BEACON_STATUS_T * p_params)
```

Description

This function packs the given parameters into a CSRmesh message and sends it over the network.

Parameters

Parameter	Description
nw_id	Network identifier over which the message has to be sent.
dest_id	16-bit identifier of the destination device/group
ttl	TTL value with which the message needs to be sent.
p_params	Pointer to the message parameters of type CSRMESH_BEACON_BEACON_STATUS_T

Returns

CSRmeshResult. Refer to CSRmeshResult.

76.4 BeaconTypes Function Documentation

Syntax

```
CSRmeshResult BeaconTypes (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8 ttl,
CSRMESH_BEACON_TYPES_T * p_params)
```

Description

This function packs the given parameters into a CSRmesh message and sends it over the network.

Parameters

Parameter	Description
nw_id	Network identifier over which the message has to be sent.
dest_id	16-bit identifier of the destination device/group
ttl	TTL value with which the message needs to be sent.
p_params	Pointer to the message parameters of type CSRMESH_BEACON_TYPES_T

Returns

CSRmeshResult. Refer to CSRmeshResult.

76.5 BeaconSetPayload Function Documentation

Syntax

```
CSRmeshResult BeaconSetPayload (CsrUint8 nw_id, CsrUint16 src_id,
CsrUint16 dest_id, CsrUint8 ttl, CSRMESH_BEACON_SET_PAYLOAD_T * p_params)
```

Description

This function packs the given parameters into a CSRmesh message and sends it over the network. When a response is received the application callback function is called to notify the CSRMESH_BEACON_PAYLOAD_ACK

Parameters

Parameter	Description
nw_id	Network identifier over which the message has to be sent.
dest_id	16-bit identifier of the destination device/group
ttl	TTL value with which the message needs to be sent.
p_params	Pointer to the message parameters of type CSRMESH_BEACON_SET_PAYLOAD_T

Returns

CSRmeshResult. Refer to CSRmeshResult.

76.6 BeaconPayloadAck Function Documentation

Syntax

```
CSRmeshResult BeaconPayloadAck (CsrUint8 nw_id, CsrUint16 dest_id,
CsrUint8 ttl, CSRMESH_BEACON_PAYLOAD_ACK_T * p_params)
```

Description

This function packs the given parameters into a CSRmesh message and sends it over the network.

Parameters

Parameter	Description
`nw_id`	Network identifier over which the message has to be sent.
`dest_id`	16-bit identifier of the destination device/group
`ttl`	TTL value with which the message needs to be sent.
`p_params`	Pointer to the message parameters of type CSRMESH_BEACON_PAYLOAD_ACK_T

Returns

CSRmeshResult. Refer to CSRmeshResult.

77 BeaconProxy Model

Detailed Description

BeaconProxy Model API.

The messages in this model are used to manage a beacon proxy, controlling which nodes it is a proxy for, and allowing its currently stored messages to be interrogated and managed .A proxy can control groups of beacons, and can have group address assigned so that groups of beacons can be updated with one message. Groups can also be used in the conventional way, but in that case we'll lose the ability to know whether everyone has received the message.

77.1 Modules of BeaconProxy Model

Client

Server

77.2 Data Structures of BeaconProxy Model

`struct CSRMESH_BEACONPROXY_ADD_T`

CSRmesh BeaconProxy Model message types.

`struct CSRMESH_BEACONPROXY_REMOVE_T`

May be sent to a proxy to remove from the list of devices it manages. Any message queued for those devices will be cleared. This request will be acknowledged using a Proxy Command Status Devices. Groups can be used, implying that all its members will be removed from the proxy management. Specifying 0 in the Device Addresses will be interpreted as stopping all Proxy activities on the targeted proxy.

`struct CSRMESH_BEACONPROXY_COMMAND_STATUS_DEVICES_T`

Generic acknowledgement - the transaction ID permits reconciliation with sender.

`struct CSRMESH_BEACONPROXY_PROXY_STATUS_T`

Provides generic information on the internal states of a Proxy. Including number of managed nodes, groups, states of the various queues.

```
struct  CSRMESH_BEACONPROXY_GET_DEVICES_T
```

Fetch the current set of devices ID managed by a given proxy. This will be answered by one or more BEACON_PROXY_DEVICES messages.

```
struct  CSRMESH_BEACONPROXY_DEVICES_T
```

Provide the list of Devices and Groups currently managed by a given Proxy. This will be a multiple parts message.

78 Client

Detailed Description

78.1 Functions of Client

```
CSRmeshResult BeaconProxyModelClientInit (CSRMESH_MODEL_CALLBACK_T app_callback)
```

Initialises BeaconProxy Model Client functionality.

```
CSRmeshResult BeaconProxyAdd (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8 ttl,
CSRMESH_BEACONPROXY_ADD_T *p_params)
```

Message can be sent to a beacon-proxy to add to the list of devices it manages. This request will be acknowledged using a Proxy Command Status Devices. Up to four device ID can be specified in this message: Group can be defined. This message also permits the flushing of pending messages for managed IDs.

```
CSRmeshResult BeaconProxyRemove (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8
ttl, CSRMESH_BEACONPROXY_REMOVE_T *p_params)
```

May be sent to a proxy to remove from the list of devices it manages. Any message queued for those devices will be cleared. This request will be acknowledged using a Proxy Command Status Devices. Groups can be used, implying that all its members will be removed from the proxy management. Specifying 0 in the Device Addresses will be interpreted as stopping all Proxy activities on the targeted proxy.

```
CSRmeshResult BeaconProxyGetStatus (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8
ttl)
```

Fetch the current status - this will be answered by a BEACON_PROXY_STATUS.

```
CSRmeshResult BeaconProxyGetDevices (CsrUint8 nw_id, CsrUint16 dest_id,
CsrUint8 ttl, CSRMESH_BEACONPROXY_GET_DEVICES_T *p_params)
```

Fetch the current set of devices ID managed by a given proxy. This will be answered by one or more BEACON_PROXY_DEVICES messages.

78.2 BeaconProxyModelClientInit Function Documentation

Syntax

```
CSRmeshResult BeaconProxyModelClientInit (CSRMESH_MODEL_CALLBACK_T app_callback)
```

Description

Parameters

Parameter	Description
app_callback	Pointer to the application callback function that will be called when the model client receives a message.

Returns

CSRmeshResult. Refer to CSRmeshResult.

78.3 BeaconProxyAdd Function Documentation

Syntax

```
CSRmeshResult BeaconProxyAdd (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8 ttl,
CSRMESH_BEACONPROXY_ADD_T * p_params)
```

Description

This function packs the given parameters into a CSRmesh message and sends it over the network. When a response is received the application callback function is called to notify the CSRMESH_BEACONPROXY_COMMAND_STATUS_DEVICES

Parameters

Parameter	Description
nw_id	Network identifier over which the message has to be sent.
dest_id	16-bit identifier of the destination device/group
ttl	TTL value with which the message needs to be sent.
p_params	Pointer to the message parameters of type CSRMESH_BEACONPROXY_ADD_T

Returns

CSRmeshResult. Refer to CSRmeshResult.

78.4 BeaconProxyRemove Function Documentation

Syntax

```
CSRmeshResult BeaconProxyRemove (CsrUint8 nw_id, CsrUint16 dest_id,
CsrUint8 ttl, CSRMESH_BEACONPROXY_REMOVE_T * p_params)
```

Description

This function packs the given parameters into a CSRmesh message and sends it over the network. When a response is received the application callback function is called to notify the CSRMESH_BEACONPROXY_COMMAND_STATUS_DEVICES

Parameters

Parameter	Description
nw_id	Network identifier over which the message has to be sent.
dest_id	16-bit identifier of the destination device/group
ttl	TTL value with which the message needs to be sent.
p_params	Pointer to the message parameters of type CSRMESH_BEACONPROXY_REMOVE_T

Returns

CSRmeshResult. Refer to CSRmeshResult.

78.5 BeaconProxyGetStatus Function Documentation

Syntax

```
CSRmeshResult BeaconProxyGetStatus (CsrUint8 nw_id, CsrUint16 dest_id,
CsrUint8 ttl)
```

Description

This function packs the given parameters into a CSRmesh message and sends it over the network. When a response is received the application callback function is called to notify the CSRMESH_BEACONPROXY_PROXY_STATUS

Parameters

Parameter	Description
nw_id	Network identifier over which the message has to be sent.
dest_id	16-bit identifier of the destination device/group
ttl	TTL value with which the message needs to be sent.

Returns

CSRmeshResult. Refer to CSRmeshResult.

78.6 BeaconProxyGetDevices Function Documentation

Syntax

```
CSRmeshResult BeaconProxyGetDevices (CsrUint8 nw_id, CsrUint16 dest_id,
CsrUint8 ttl, CSRMESH_BEACONPROXY_GET_DEVICES_T * p_params)
```

Description

This function packs the given parameters into a CSRmesh message and sends it over the network. When a response is received the application callback function is called to notify the CSRMESH_BEACONPROXY_DEVICES

Parameters

Parameter	Description
nw_id	Network identifier over which the message has to be sent.
dest_id	16-bit identifier of the destination device/group
ttl	TTL value with which the message needs to be sent.
p_params	Pointer to the message parameters of type CSRMESH_BEACONPROXY_GET_DEVICES_T

Returns

CSRmeshResult. Refer to CSRmeshResult.

79 Server

Detailed Description

79.1 Functions of Server

```
CSRmeshResult BeaconProxyModelInit (CsrUint8 nw_id, CsrUint16 *group_id_list,
CsrUint16 num_groups, CSRMESH_MODEL_CALLBACK_T app_callback)
```

Model Initialisation.

```
CSRmeshResult BeaconProxySetupBeaconList (CsrUint16 *beacon_list, CsrUint8
max_groups, CsrUint8 max_devices)
```

Sets up a list to manage added beacons.

```
CSRmeshResult BeaconProxyCommandStatusDevices (CsrUint8 nw_id, CsrUint16
dest_id, CsrUint8 ttl, CSRMESH_BEACONPROXY_COMMAND_STATUS_DEVICES_T *p_params)
```

Generic acknowledgement - the transaction ID permits reconciliation with sender.

```
CSRmeshResult BeaconProxyProxyStatus (CsrUint8 nw_id, CsrUint16 dest_id,
CsrUint8 ttl, CSRMESH_BEACONPROXY_PROXY_STATUS_T *p_params)
```

Provides generic information on the internal states of a Proxy. Including number of managed nodes, groups, states of the various queues.

```
CSRmeshResult BeaconProxyDevices (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8
ttl, CSRMESH_BEACONPROXY_DEVICES_T *p_params)
```

Provide the list of Devices and Groups currently managed by a given Proxy. This will be a multiple parts message.

79.2 BeaconProxyModelInit Function Documentation

Syntax

```
CSRmeshResult BeaconProxyModelInit (CsrUint8 nw_id, CsrUint16 * group_id_list,
CsrUint16 num_groups, CSRMESH_MODEL_CALLBACK_T app_callback)
```

Description

Registers the model handler with the CSRmesh. Sets the CSRmesh to report num_groups as the maximum number of groups supported for the model

Parameters

Parameter	Description
nw_id	Identifier of the network to which the model has to be registered.
group_id_list	Pointer to a uint16 array to hold assigned group_ids. This must be NULL if no groups are supported
num_groups	Size of the group_id_list. This must be 0 if no groups are supported.
app_callback	Pointer to the application callback function. This function will be called to notify all model specific messages

Returns

CSRmeshResult. Refer to CSRmeshResult.

79.3 BeaconProxySetupBeaconList Function Documentation

Syntax

```
CSRmeshResult BeaconProxySetupBeaconList (CsrUint16 * beacon_list,
CsrUint8 max_groups, CsrUint8 max_devices)
```

Description

Parameters

Parameter	Description
beacon_list	Pointer to buffer to hold the added beacon device IDs and group IDs. The buffer should be large enough to store max_devices + max_groups
max_devices	Maximum number of devices that the proxy can manage
max_groups	Maximum groups of beacons the proxy can manage

Returns

CSRmeshResult. Refer to CSRmeshResult.

79.4 BeaconProxyCommandStatusDevices Function Documentation

Syntax

```
CSRmeshResult BeaconProxyCommandStatusDevices (CsrUint8 nw_id,
CsrUint16 dest_id, CsrUint8 ttl, CSRMESH_BEACONPROXY_COMMAND_STATUS_DEVICES_T
* p_params)
```

Description

This function packs the given parameters into a CSRmesh message and sends it over the network.

Parameters

Parameter	Description
nw_id	Network identifier over which the message has to be sent.
dest_id	16-bit identifier of the destination device/group
ttl	TTL value with which the message needs to be sent.
p_params	Pointer to the message parameters of type CSRMESH_BEACONPROXY_COMMAND_STATUS_DEVICES_T

Returns

CSRmeshResult. Refer to CSRmeshResult.

79.5 BeaconProxyProxyStatus Function Documentation

Syntax

```
CSRmeshResult BeaconProxyProxyStatus (CsrUint8 nw_id, CsrUint16 dest_id,
CsrUint8 ttl, CSRMESH_BEACONPROXY_PROXY_STATUS_T * p_params)
```

Description

This function packs the given parameters into a CSRmesh message and sends it over the network.

Parameters

Parameter	Description
nw_id	Network identifier over which the message has to be sent.
dest_id	16-bit identifier of the destination device/group
ttl	TTL value with which the message needs to be sent.
p_params	Pointer to the message parameters of type CSRMESH_BEACONPROXY_PROXY_STATUS_T

Returns

CSRmeshResult. Refer to CSRmeshResult.

79.6 BeaconProxyDevices Function Documentation

Syntax

```
CSRmeshResult BeaconProxyDevices (CsrUint8 nw_id, CsrUint16 dest_id,
CsrUint8 ttl, CSRMESH_BEACONPROXY_DEVICES_T * p_params)
```

Description

This function packs the given parameters into a CSRmesh message and sends it over the network.

Parameters

Parameter	Description
nw_id	Network identifier over which the message has to be sent.
dest_id	16-bit identifier of the destination device/group
ttl	TTL value with which the message needs to be sent.
p_params	Pointer to the message parameters of type CSRMESH_BEACONPROXY_DEVICES_T

Returns

CSRmeshResult. Refer to CSRmeshResult.

80 NVM_Access

Detailed Description

80.1 Data Structures of NVM_Access

`struct nvm_cs_header_t`

NVM key descriptor.

`struct nvm_versioned_header_t`

Versioned header information.

80.2 Macros of NVM_Access

```
#define NVM_PAD_ROUND_UP_TO_16(x)    ((sizeof(nvm_versioned_header_t) + x + 15)
& ~0x0f)
```

Macros to round a number up to the next 16 or 32 after including the Versioned Header size.

80.3 Typedefs of NVM_Access

```
typedef bool(* migrate_handler_t) (uint16 version_in_nvm, uint16
nvm_app_data_offset, uint16 nvm_data_length)
```

Function to convert data in the NVM to a newer version.

80.4 Enumerations of NVM_Access

`enum nvm_data_id_t`

NVM key identifiers.

80.5 Functions of NVM_Access

```
bool Nvm_ValidateVersionedHeader (uint16 *nvm_offset, nvm_data_id_t key_id,
uint16 data_len, uint16 version, migrate_handler_t migrate_handler)
```

Verifies that the data block in NVM matches the expected identifier.

```
void Nvm_WriteVersionedHeader (uint16 *nvm_offset, nvm_data_id_t key_id, uint16
data_len, uint16 version)
```

Writes a versioned header to NVM.

```
void Nvm_Init (store_id_t id, uint16 nvm_sanity, uint16 *nvm_offset)
```

Initialises the NVM.

```
void AppNvmReady (bool nvm_fresh, uint16 nvm_offset)
```

Called when the NVM initialisation is done.

```
void Nvm_Read (uint16 *buffer, uint16 length, uint16 offset)
```

Read words from the NVM store after preparing the NVM to be readable.

```
void Nvm_Write (uint16 *buffer, uint16 length, uint16 offset)
```

Write words to the NVM store after preparing the NVM to be writable.

```
void NvmProcessEvent (msg_t *msg)
```

Handles the user store messages.

```
sys_status Nvm_SetMemType (memory_type_t type)
```

Sets the memory device type for the user store.

80.6 bool(* migrate_handler_t) (uint16 version_in_nvm, uint16 nvm_app_data_offset, uint16 nvm_data_length) Typedef Documentation

Definition

```
typedef bool(* migrate_handler_t) (uint16 version_in_nvm, uint16
nvm_app_data_offset, uint16 nvm_data_length)
```

Description

Function that can be called during the NVM version validation process, initiated by calling Nvm_ValidateVersionedHeader(). The function should return FALSE if the NVM data is totally incompatible with the new function. If it is possible to migrate the data between the versions then the application should use a Nvm_Read() and Nvm_Write() to reformat the data to meet the new version. The application should not write more than nvm_data_length words into NVM from nvm_app_data_offset.

80.7 Nvm_ValidateVersionedHeader Function Documentation

Syntax

```
bool Nvm_ValidateVersionedHeader (uint16 * nvm_offset, nvm_data_id_t key_id,
uint16 data_len, uint16 version, migrate_handler_t migrate_handler)
```

Description

This function validates the Versioned NVM header in NVM against a set of parameters. If the versioned header in NVM matches the ID and length passed, but the version number does not match, then the migrate_handler is called. The migrate_handler should attempt to update the data in NVM to the new version.

Parameters

Parameter	Description
[in,out]nvm_offset	Pointer to NVM offset
[in]key_id	System specific ID to check for
[in]data_len	Length of data expected
[in]version	Application specific version number for the data
[in]migrate_handler	If not NULL, a function which is called to modify the NVM data a new application specific format.

Returns

True if the requested versioned header was found; nvm_offset will be updated to the next NVM offset

80.8 Nvm_WriteVersionedHeader Function Documentation

Syntax

```
void Nvm_WriteVersionedHeader (uint16 * nvm_offset, nvm_data_id_t key_id,
uint16 data_len, uint16 version)
```

Description

This function writes meta data to the NVM. The data contains an ID which should be used by the application to identify its data block.

Parameters

Parameter	Description
[in,out]nvm_offset	Offset into the NVM
[in]key_id	Identifier for the block of data
[in]data_len	Size of the block of data
[in]version	Application specific version information about the block of data

Returns

Nothing

80.9 Nvm_Init Function Documentation

Syntax

```
void Nvm_Init (store_id_t id, uint16 nvm_sanity, uint16 * nvm_offset)
```

Description

This function initialises the NVM

Parameters

Parameter	Description
[in]nvm_sanity	NVM sanity word
[in]nvm_offset	Pointer to NVM offset

Returns

Nothing

80.10 AppNvmReady Function Documentation

Syntax

```
void AppNvmReady (bool nvm_fresh, uint16 nvm_offset)
```

Description

This function is called when the NVM is initialised

Parameters

Parameter	Description
[in]nvm_fresh	Boolean variable to check if NVM has been not been written before
[out]nvm_offset	Current NVM offset

Returns

Nothing

80.11 Nvm_Read Function Documentation

Syntax

```
void Nvm_Read (uint16 * buffer, uint16 length, uint16 offset)
```

Description

This function reads words from the NVM store after preparing the NVM to be readable

Parameters

Parameter	Description
[out]buffer	The buffer to read words into
[in]length	The number of words to read
[in]offset	The word offset within the NVM Store to read from

Returns

Nothing

80.12 Nvm_Write Function Documentation

Syntax

```
void Nvm_Write (uint16 * buffer, uint16 length, uint16 offset)
```

Description

This function writes words to the NVM store after preparing the NVM to be writable

Parameters

Parameter	Description
[in]buffer	The buffer to write
[in]length	The number of words to write
[in]offset	The word offset within the NVM Store to write to

Returns

Nothing

80.13 NvmProcessEvent Function Documentation

Syntax

```
void NvmProcessEvent (msg_t * msg)
```

Description

This function handles the user store messages

Parameters

Parameter	Description
[in]msg	Pointer to incoming message event

Returns

Nothing

80.14 Nvm_SetMemType Function Documentation

Syntax

```
sys_status Nvm_SetMemType (memory_type_t type)
```

Description

This function sets the memory device type for the user store

Parameters

Parameter	Description
[in]memory_type_t	Memory device type to set

Returns

Status

81 Data Structures

Here are the data structures with brief descriptions:

Name	Description
CSR_MESH_APP_EVENT_DATA_T	CSR Mesh App Event Data - to provide event, status * app data to app
CSR_MESH_APPEARANCE_T	CSRmesh Device Appearance. The Appearance is a 24-bit value that is composed of an "organization" and an "organization appearance"
CSR_MESH_ASSOCIATION_ATTENTION_DATA_T	Association attention request data type
CSR_MESH_AUTH_CODE_T	64 bit Authorisation Code type
CSR_MESH_BEARER_STATE_DATA_T	CSR Mesh Bearer state Type
CSR_MESH_CC_INFO_T	Client config info
CSR_MESH_CONFIG_BEARER_PARAM_T	CSRmesh Scan and Advertising Parameters
CSR_MESH_CONFIG_MSG_PARAMS_T	CSRmesh Scan and Advertising Parameters
CSR_MESH_CONFORMANCE_SIGNATURE_T	CSRmesh conformance signature Information
CSR_MESH_DEVICE_APPEARANCE_T	Device Appearance data type
CSR_MESH_DEVICE_ID_UPDATE_T	CSR Mesh device id update data Type
CSR_MESH_GROUP_ID_RELATED_DATA_T	CSR Mesh Group Id Related Data - To provide to app while handling Group model data in core mesh
CSR_MESH_MASP_DEVICE_DATA_T	CSRmesh Device Identifier Data and Signature
CSR_MESH_MASP_DEVICE_IND_T	Device indication data type
CSR_MESH_MASP_DEVICE_SIGN_LIST_NODE_T	CSRmesh Device Signature Data Linked List node
CSR_MESH_MESH_ID_DATA_T	CSR Mesh mesh_id data Type

Name	Description
CSR_MESH_NETID_LIST_T	CSR Mesh Network Id List
CSR_MESH_NW_RELAY_T	CSR Mesh transmit state
CSR_MESH_SEQ_CACHE_T	CSR mesh <<SRC,SEQ>> Cache Structure
CSR_MESH_SQN_LOOKUP_TABLE_T	CSR Mesh <<SRC,SEQ>> Lookup Table
CSR_MESH_STACK_VERSION_T	CSRmesh stack version Information
CSR_MESH_TRANSMIT_STATE_T	CSR Mesh transmit state
CSR_MESH_TTL_T	CSR Mesh TTL
CSR_MESH_UUID_T	128-bit UUID type
CSR_MESH_VID_PID_VERSION_T	CSRmesh Product ID, Vendor ID and Version Information
CSR_SCHED_ADV_DATA_T	CSR Mesh Scheduling LE Advertising Data
CSR_SCHED_ADV_PARAMS_T	CSR Mesh Scheduling LE Advertisment Parameter Type
CSR_SCHED_GATT_EVENT_DATA_T	CSR Mesh Scheduling GATT Event Type
CSR_SCHED_GENERIC_LE_PARAM_T	CSR Mesh Scheduling Generic LE Parameter Type
CSR_SCHED_LE_PARAMS_T	CSR Mesh Scheduling LE Parameter Type
CSR_SCHED_MESH_LE_PARAM_T	CSR Mesh Scheduling Mesh-LE Parameter Type
CSR_SCHED_MESH_TX_BUFFER_T	CSR Mesh Sched TX queue buffer Type
CSR_SCHED_MESH_TX_PARAM_T	CSR Mesh Scheduling Tx Parameter Type
CSR_SCHED_SCAN_DUTY_PARAM_T	CSR Mesh Scheduling Scan Duty Parameter Type
CSR_SCHED_SCAN_WINDOW_PARAM_T	CSR Mesh Scheduling Scan Window Parameter Type
CSRMESH_ACTION_ACTION_STATUS_T	This provides a bitmask identifying all the currently allocated ActionID. The Action Supported field provides an indication on how many actions this device has capacity for. Note that a device will never have more than 32 actions

Name	Description
CSRMESH_ACTION_DELETE_ACK_T	This message confirms the delete commands and report which actions have been removed. The provided transaction ID will match the one supplied by ACTION_DELETE
CSRMESH_ACTION_DELETE_T	This message allows a set of actions to be removed. Actions are identified through a bitmask reflecting the ActionID one wishes to delete. This message will be acknowledged through ACTION_DELETE_ACK
CSRMESH_ACTION_GET_ACTION_STATUS_T	Request towards a node about the various Actions currently handled. This will be answered through an ACTION_STATUS. A Transaction ID is provided to ensure identification of response with request
CSRMESH_ACTION_GET_T	Using this message, it is possible to interrogate a node about the specific payload associated with an Action ID. This will be answered by and ACTION_SET_ACTION message, ACTION_SET_ACTION_ACK will need to be issued in order to collect all parts
CSRMESH_ACTION_SET_ACTION_ACK_T	Every parts received is acknowledged, providing both Transaction ID and ActionID (which includes Part Number)
CSRMESH_ACTION_SET_ACTION_INFO_EXT_T	This message defines the action to be sent when GET_ACTION is received. This is a multiple parts message, each part will be acknowledged through an ACTION_SET_ACK (0x51).This also contains the buffer to send the action message through it
CSRMESH_ACTION_SET_ACTION_INFO_T	CSRmesh Action Model message types
CSRMESH_ACTION_SET_ACTION_T	This message defines the action which needs to be taken. This is a multiple parts message, each part will be acknowledged through an ACTION_SET_ACK (0x51)
CSRMESH_ACTUATOR_GET_TYPES_T	CSRmesh Actuator Model message types
CSRMESH_ACTUATOR_GET_VALUE_ACK_T	Get Current sensor state
CSRMESH_ACTUATOR_SET_VALUE_T	Get sensor state. Upon receiving an ACTUATOR_SET_VALUE_NO_ACK message, where the destination address is the device ID of this device and the Type field is a supported actuator type, the device shall immediately use the Value field for the given Type field. The meaning of this actuator value is not defined in this specification. Upon receiving an ACTUATOR_SET_VALUE_NO_ACK message, where the destination address is is the device ID of this device but the Type field is not a supported actuator type, the device shall ignore the message
CSRMESH_ACTUATOR_TYPES_T	Actuator types
CSRMESH_ACTUATOR_VALUE_ACK_T	Current sensor state

Name	Description
CSRMESH_ASSET_ANNOUNCE_T	Asset announcement
CSRMESH_ASSET_GET_STATE_T	Getting Asset State: Upon receiving an ASSET_GET_STATE message, the device responds with an ASSET_STATE message with the current state information
CSRMESH_ASSET_SET_STATE_T	CSRmesh Asset Model message types
CSRMESH_ASSET_STATE_T	Current asset state
CSRMESH_ATTENTION_SET_STATE_T	CSRmesh Attention Model message types
CSRMESH_ATTENTION_STATE_T	Current battery state
CSRMESH_BATTERY_GET_STATE_T	CSRmesh Battery Model message types
CSRMESH_BATTERY_STATE_T	Current battery state
CSRMESH_BEACON_BEACON_STATUS_T	This message issue issued by a Beacon as response to BEACON_SET_STATUS or BEACON_GET_STATUS. Furthermore, a Beacon appearing on the mesh sporadically, will issue such message (with destination ID set to 0) as soon as it re-joins the mesh. In this case, one message per active beacon type should be issued. This message will provide the Payload ID currently in used for the associated type
CSRMESH_BEACON_GET_BEACON_STATUS_T	Message to be sent to a beacon in order to recover its current status. This message fetch information pertinent to a particular type. The transaction ID will help matching the pertinent BEACON_STATUS message: a different transaction ID shall be used for a different type (although the BEACON_STATUS message has the type pertinent to the status and thus could be used in conjunction to the Transaction ID for disambiguation)
CSRMESH_BEACON_GET_PAYLOAD_T	Message allowing a node to retrieve the current payload held on a given beacon. This message shall be answered by one or more BEACON_SET_PAYLOAD messages
CSRMESH_BEACON_GET_TYPES_T	Message allowing a node to fetch all supported types and associated information by a given Beacon
CSRMESH_BEACON_PAYLOAD_ACK_T	Only one Acknowledgement message will occur for multiple BEACON_SET_PAYLOAD messages sharing the same transaction ID. Only when the last segment is received that such acknowledgement will be issued. Where missing payload messages exist, the list of their indices will be provided in the Ack field

Name	Description
CSRMESH_BEACON_SET_PAYLOAD_T	One or more of these packets may be sent to a beacon to set its payload. The content is either the raw advert data, or extra information (such as crypto cycle) which a beacon may require. The payload data is sent as a length and an offset, so the whole payload need not be sent if it has not changed. A beacon can support many beacon types - it can be sent as many different payloads as needed, one for each type. The first byte of the first payload packet contains the length and offset of the payload and the payload ID; this allows a beacon which already has the payload to send an immediate acknowledgement, saving traffic. This ID will be sent back in the 'Payload ACK' message if the beacon has received the whole payload. The payload may have to be split in several parts, in which case only the last part shall be acknowledged. Upon receiving a BEACON_SET_PAYLOAD, the beacon will update its corresponding beacon data: if data was previously available, it will be replaced by the provided payload, thus a beacon can only hold one payload per beacon type
CSRMESH_BEACON_SET_STATUS_T	CSRmesh Beacon Model message types
CSRMESH_BEACON_TYPES_T	Provides information on the set of beacon supported by the node. The battery level is reported as well as time since last message
CSRMESH_BEACONPROXY_ADD_T	CSRmesh BeaconProxy Model message types
CSRMESH_BEACONPROXY_COMMAND_STATUS_DEVICES_T	Generic acknowledgement - the transaction ID permits reconciliation with sender
CSRMESH_BEACONPROXY_DEVICES_T	Provide the list of Devices and Groups currently managed by a given Proxy. This will be a multiple parts message
CSRMESH_BEACONPROXY_GET_DEVICES_T	Fetch the current set of devices ID managed by a given proxy. This will be answered by one or more BEACON_PROXY_DEVICES messages
CSRMESH_BEACONPROXY_PROXY_STATUS_T	Provides generic information on the internal states of a Proxy. Including number of managed nodes, groups, states of the various queues
CSRMESH_BEACONPROXY_REMOVE_T	May be sent to a proxy to remove from the list of devices it manages. Any message queued for those devices will be cleared. This request will be acknowledged using a Proxy Command Status Devices. Groups can be used, implying that all its members will be removed from the proxy management. Specifying 0 in the Device Addresses will be interpreted as stopping all Proxy activities on the targeted proxy
CSRMESH_BEARER_GET_STATE_T	Getting Bearer State: Upon receiving a BEARER_GET_STATE message, where the destination address is the device ID of this device, the device responds with a BEARER_STATE message with the current state information

Name	Description
CSRMESH_BEARER_SET_STATE_T	CSRmesh Bearer Model message types
CSRMESH_BEARER_STATE_T	Set bearer state
CSRMESH_CONFIG_DEVICE_IDENTIFIER_T	Device identifier
CSRMESH_CONFIG_DISCOVER_DEVICE_T	Upon receiving a CONFIG_DISCOVER_DEVICE message directed at the 0x0000 group identifier or to DeviceID of this device, the device responds with a CONFIG_DEVICE_IDENTIFIER message
CSRMESH_CONFIG_GET_INFO_T	Upon receiving a CONFIG_GET_INFO message, directed at the DeviceID of this device, the device responds with a CONFIG_INFO message. The Info field of the CONFIG_GET_INFO message determines the information to be included in the CONFIG_INFO message. The following information values are defined: DeviceUUIDLow (0x00) contains the least significant eight octets of the DeviceUUID state value. DeviceUUIDHigh (0x01) contains the most significant eight octets of the DeviceUUID state value. ModelsLow (0x02) contains the least significant eight octets of the ModelsSupported state value. ModelsHigh (0x03) contains the most significant eight octets of the ModelsSupported state value
CSRMESH_CONFIG_GET_PARAMETERS_T	Upon receiving a CONFIG_GET_PARAMETERS message, where the destination address is the DeviceID of this device, the device will respond with a CONFIG_PARAMETERS message with the current config model state information
CSRMESH_CONFIG_INFO_T	Current device information
CSRMESH_CONFIG_LAST_SEQUENCE_NUMBER_T	CSRmesh Config Model message types
CSRMESH_CONFIG_PARAMETERS_T	Configuration parameters
CSRMESH_CONFIG_RESET_DEVICE_T	Upon receiving a CONFIG_RESET_DEVICE message from a trusted device directed at only this device, the local device sets the DeviceID to zero, and forgets all network keys, associated NetworkIVs and other configuration information. The device may act as if it is not associated and use MASP to re-associate with a network. Note: If the CONFIG_RESET_DEVICE message is received on any other destination address than the DeviceID of the local device, it is ignored. This is typically used when selling a device, to remove the device from the network of the seller so that the purchaser can associate the device with their network

Name	Description
CSRMESH_CONFIG_SET_DEVICE_IDENTIFIER_T	When the device with a DeviceID of 0x0000 receives a CONFIG_SET_DEVICE_IDENTIFIER message and the DeviceHash of the message matches the DeviceHash of this device, the DeviceID of this device is set to the DeviceID field of this message. Then the device responds with the DEVICE_CONFIG_IDENTIFIER message using the new DeviceID as the source address. Note: This function is not necessary in normal operation of a CSRmesh network as DeviceID is distributed as part of the MASP protocol in the MASP_ID_DISTRIBUTION message
CSRMESH_CONFIG_SET_PARAMETERS_T	Upon receiving a CONFIG_SET_PARAMETERS message, where the destination address is the DeviceID of this device, the device saves the TxInterval, TxDuration, RxDutyCycle, TxPower and TTL fields into the TransmitInterval, TransmitDuration, ReceiverDutyCycle, TransmitPower and DefaultTimeToLive state respectively. Then the device responds with a CONFIG_PARAMETERS message with the current configuration model state information
CSRMESH_DATA_BLOCK_SEND_T	A block of data, no acknowledgement. Upon receiving a DATA_BLOCK_SEND message, the device passes the DatagramOctets field up to the application for processing
CSRMESH_DATA_STREAM_FLUSH_T	CSRmesh Data Model message types
CSRMESH_DATA_STREAM_RECEIVED_T	Acknowledgement of data received
CSRMESH_DATA_STREAM_SEND_T	Sending Data: Upon receiving a DATA_STREAM_SEND message, the device first checks if the StreamSN field is the same as the StreamSequenceNumber model state. If these values are the same, the device passes the StreamOctets field up to the application for processing, and increments StreamSequenceNumber by the length of the StreamOctets field. It then responds with a DATA_STREAM_RECEIVED message with the current value of the StreamSequenceNumber. Note: The DATA_STREAM_RECEIVED message is sent even if the StreamSN received is different from the StreamSequenceNumber state. This allows missing packets to be detected and retransmitted by the sending device
CSRMESH_DIAGNOSTIC_STATE_T	CSRmesh Diagnostic Model message types
CSRMESH_EVENT_DATA_T	CSRmesh Event Data

Name	Description
CSRMESH_EXTENSION_CONFLICT_T	Response to a REQUEST - only issued if a conflict is noticed. This message indicates that the proposed OpCode is already in use within the node processing the request message. Nodes receiving conflict extension will process this message and remove the conflicting OpCode from the list of OpCodes to handle. All conflict messages are relayed, processed or not. If a node receiving a REQUEST is able to match the hash of the provider previously assigned to an existing OpCode, but different to the proposed one, it responds with a CONFLICT with a reason combining the top bit with the previously associated range (0x80 | <old range>=""">). In such cases, the previously used OpCode (start of range) will be placed in the ProposedOpCode. Nodes receiving this conflict message with the top bit raised, will discard the initially proposed OpCode and replace it with the proposed code supplied in the conflict message
CSRMESH_EXTENSION_REQUEST_T	CSRmesh Extension Model message types
CSRMESH_FIRMWARE_GET_VERSION_T	CSRmesh Firmware Model message types
CSRMESH_FIRMWARE_UPDATE_ACKNOWLEDGED_T	Acknowledgement message to the firmware update request
CSRMESH_FIRMWARE_UPDATE_REQUIRED_T	Requesting a firmware update. Upon receiving this message, the device moves to a state where it is ready for receiving a firmware update
CSRMESH_FIRMWARE_VERSION_T	Firmware version information
CSRMESH_GROUP_GET_MODEL_GROUPID_T	Getting Model Group ID: Upon receiving a GROUP_GET_MODEL_GROUPID message, where the destination address is the DeviceID of this device, the device responds with a GROUP_MODEL_GROUPID message with the current state information held for the given Model and GroupIndex values
CSRMESH_GROUP_GET_NUMBER_OF_MODEL_GROUPIDS_T	CSRmesh Group Model message types
CSRMESH_GROUP_MODEL_GROUPID_T	GroupID of a model
CSRMESH_GROUP_NUMBER_OF_MODEL_GROUPIDS_T	Get number of groups supported by the model
CSRMESH_GROUP_SET_MODEL_GROUPID_T	Setting Model Group ID: Upon receiving a GROUP_SET_MODEL_GROUPID message, where the destination address is the DeviceID of this device, the device saves the Instance and GroupID fields into the appropriate state value determined by the Model and GroupIndex fields. It then responds with a GROUP_MODEL_GROUPID message with the current state information held for the given model and the GroupIndex values

Name	Description
CSRMESH_LARGEOBJECTTRANSFER_ANNOUNCE_T	CSRmesh LargeObjectTransfer Model message types
CSRMESH_LARGEOBJECTTRANSFER_INTEREST_T	In case a node is ready to receive the proposed object, it responds with this message. The intended behaviour of the Large Object Transfer is to allow a Peer-to-Peer connection between the consumer and the producer. The consumer uses a ServiceID, part of which is randomly selected. The top 64 bits are 0x1122334455667788, the least significant 63 bits are randomly selected by the consumer node. The most significant bit of the least significant 64 bits is an encoding of the intent to relay the received data. Once this message has been issued, the consumer node starts advertising a connectable service with the 128-bits service composed through the concatenation of the fixed 64 bits and the randomly selected 63 bits. The duration of the advertisement is an implementation decision
CSRMESH_LIGHT_GET_STATE_T	Getting Light State: Upon receiving a LIGHT_GET_STATE message, the device responds with a LIGHT_STATE message
CSRMESH_LIGHT_GET_WHITE_T	Setting Light White level
CSRMESH_LIGHT_SET_COLOR_TEMP_T	Setting Light Colour Temperature: Upon receiving a LIGHT_SET_COLOR_TEMP message, the device saves the ColorTemperature field into the TargetColorTemperature state variable. If the TempDuration field is zero, the CurrentColorTemperature variable is set to TargetColorTemperature and DeltaColorTemperature is set to zero. If the TempDuration field is greater than zero, then the device calculates the difference between TargetColorTemperature and CurrentColorTemperature, over the TempDuration field and store this into a DeltaColorTemperature state variable, so that over TempDuration seconds, CurrentColorTemperature changes smoothly to TargetColorTemperature. The device then responds with a LIGHT_STATE message
CSRMESH_LIGHT_SET_LEVEL_T	CSRmesh Light Model message types
CSRMESH_LIGHT_SET_POWER_LEVEL_T	Setting Light Power and Light Level: Upon receiving a LIGHT_SET_POWER_LEVEL_NO_ACK message, the device sets the current PowerState to the Power field, the TargetLevel variable to the Level field, the DeltaLevel to the difference between TargetLevel and CurrentLevel divided by the LevelDuration field, saves the Sustain and Decay fields into the LevelSustain and LevelDecay variables, and sets LevelSDState to the Attacking state. If ACK is requested, the device should respond with a LIGHT_STATE message

Name	Description
CSRMESH_LIGHT_SET_RGB_T	Setting Light Colour: Upon receiving a LIGHT_SET_RGB_NO_ACK message, the device saves the Level, Red, Green, and Blue fields into the TargetLevel, TargetRed, TargetGreen, and TargetBlue variables respectively. LevelSDState should be set to Attacking. If the Duration field is zero, then the device saves the Level, Red, Green, and Blue fields into the CurrentLevel, CurrentRed, CurrentGreen and CurrentBlue variables, and sets the DeltaLevel, DeltaRed, DeltaGreen, and DeltaBlue variables to zero. If the Duration field is greater than zero, then the device calculates the DeltaLevel, DeltaRed, DeltaGreen, and DeltaBlue levels from the differences between the Current values and the Target values divided by the Duration field, so that over Duration seconds, the CurrentLevel, CurrentRed, CurrentGreen, and CurrentBlue variables are changed smoothly to the TargetLevel, TargetRed, TargetGreen and TargetBlue values. If ACK is requested, the device responds with a LIGHT_STATE message
CSRMESH_LIGHT_SET_WHITE_T	Setting Light White level
CSRMESH_LIGHT_STATE_T	Current light state
CSRMESH_LIGHT_WHITE_T	Setting Light White level
CSRMESH_PING_REQUEST_T	CSRmesh Ping Model message types
CSRMESH_PING_RESPONSE_T	Ping response
CSRMESH_POWER_GET_STATE_T	Getting Power State: Upon receiving a POWER_GET_STATE message, the device responds with a POWER_STATE message with the current state information
CSRMESH_POWER_SET_STATE_T	CSRmesh Power Model message types
CSRMESH_POWER_STATE_T	Current power state
CSRMESH_POWER_TOGGLE_STATE_T	Toggling Power State: Upon receiving a POWER_Toggle_STATE_NO_ACK message, the device sets the PowerState state value as defined: 1.If the current PowerState is 0x00, Off, then PowerState should be set to 0x01, On. 2.If the current PowerState is 0x01, On, then PowerState should be set to 0x00, Off. 3.If the current PowerState is 0x02, Standby, then PowerState should be set to 0x03, OnFromStandby. 4.If the current PowerState is 0x03, OnFromStandby, then PowerState should be set to 0x02, Standby. Then the device responds with a POWER_STATE message with the current state information

Name	Description
CSRMESH_SENSOR_GET_STATE_T	Getting Sensor State: Upon receiving a SENSOR_GET_STATE message, where the destination address is the deviceID of this device and the Type field is a supported sensor type, the device shall respond with a SENSOR_STATE message with the current state information of the sensor type. Upon receiving a SENSOR_GET_STATE message, where the destination address is the device ID of this device but the Type field is not a supported sensor type, the device shall ignore the message
CSRMESH_SENSOR_GET_TYPES_T	CSRmesh Sensor Model message types
CSRMESH_SENSOR_MISSING_T	Sensor data is missing. Proxy Behaviour: Upon receiving a SENSOR_MISSING message, the proxy determines if it has remembered this type and value and then writes that type and value to the device that sent the message
CSRMESH_SENSOR_READ_VALUE_T	Getting Sensor Value: Upon receiving a SENSOR_READ_VALUE message, where the Type field is a supported sensor type, the device responds with a SENSOR_VALUE message with the value of the sensor type. Proxy Behaviour: Upon receiving a SENSOR_GET_STATE where the destination of the message and the sensor type correspond to a previously received SENSOR_BROADCAST_VALUE or SENSOR_BROADCAST_NEW message, the device responds with a SENSOR_VALUE message with the remembered values
CSRMESH_SENSOR_SET_STATE_T	Setting Sensor State: Upon receiving a SENSOR_SET_STATE message, where the destination address is the device ID of this device and the Type field is a supported sensor type, the device saves the RxDutyCycle field and responds with a SENSOR_STATE message with the current state information of the sensor type. If the Type field is not a supported sensor type, the device ignores the message
CSRMESH_SENSOR_STATE_T	Current sensor state
CSRMESH_SENSOR_TYPES_T	Sensor types
CSRMESH_SENSOR_VALUE_T	Current sensor value
CSRMESH_SENSOR_WRITE_VALUE_T	Writing Sensor Value: Upon receiving a SENSOR_WRITE_VALUE message, where the Type field is a supported sensor type, the device saves the value into the current value of the sensor type on this device and responds with a SENSOR_VALUE message with the current value of this sensor type

Name	Description
CSRMESH_TIME_BROADCAST_T	Synchronise wall clock time from client device: This message is always sent with TTL=0. This message is sent at intervals by the clock master. It is always sent with TTL=0. It is repeated, but the time is updated before each repeat is sent. The clock master repeats the message 5 times, relaying stations repeat it 3 times. When a node receives a clock broadcast its behaviour depends on the current clock state: n MASTER: Ignore broadcasts.n INIT: Start the clock; relay this message. Set state to NO_RELAY if MasterFlag set, otherwise RELAY_MASTER. Start relay timer.n RELAY: Correct clock if required. Relay this message. Set state to NO_RELAY if MasterFlag set, otherwise RELAY_MASTER. Start relay timer.n NO_RELAY: Ignore. State will be reset to RELAY when the relay timer goes off.n RELAY_MASTER: Relay message only if it is from the clock master and set state to NO_RELAY.n The relay timer is by default 1/4 of the clock broadcast interval (15 seconds if the interval is 60 seconds). This means that each node will relay a message only once, and will give priority to messages from the clock master (which always causes the clock to be corrected). Messages from other nodes will only cause clock correction if they exceed the max clock skew (250ms)
CSRMESH_TIME_GET_STATE_T	Getting Time Broadcast Interval: Upon receiving a TIME_GET_STATE message, the device responds with a TIME_STATE message with the current state information
CSRMESH_TIME_SET_STATE_T	CSRmesh Time Model message types
CSRMESH_TIME_STATE_T	Set time broadcast interval
CSRMESH_TRACKER_FIND_T	CSRmesh Tracker Model message types
CSRMESH_TRACKER_FOUND_T	Asset found
CSRMESH_TRACKER_REPORT_T	Asset report
CSRMESH_TRACKER_SET_PROXIMITY_CONFIG_T	Set tracker proximity config
CSRMESH_TUNING_ACK_CONFIG_T	Current tuning config for a device: This message comes as a response to a SET_CONFIG. Encoding its various fields follow the same convention as the ones exposed in SET_CONFIG

Name	Description
CSRMESH_TUNING_GET_STATS_T	Getting Tuning Stats: These messages are aimed at collecting statistics from specific nodes. This message allows for the request of all information or for some of its parts. Responses are multi-parts, each identified with an index (combining a continuation flag - top bit). MissingReplyParts for the required field serves at determining the specific responses one would like to collect. If instead all the information is requested, setting this field to zero will inform the destination device to send all messages. Importantly, response (STATS_RESPONSE) messages will not necessarily come back in order, or all reach the requestor. It is essential to handle these cases in the treatment of the collected responses
CSRMESH_TUNING_PROBE_T	CSRmesh Tuning Model message types
CSRMESH_TUNING_SET_CONFIG_T	Setting (or reading) tuning config: Omitted or zero fields mean do not change. This message enforces the state of the recipient. The state is defined by two goals, normal and high traffic, and their associated number of neighbour to decide which cases to follow. Goals are expressed with unit-less values ranging from 0 to 255. These goals relate to metrics calculated on the basis of density computed at the node and across the network. The expectation is for these goals to be maintained through modification of the receive duty cycle. The average of number of neighbours for high and normal traffic is expressed as a ratio, both numbers sharing the same denominator and each representing their respective numerators. The duty cycle encoding follows the same rules as per duty cycle encoding encountered in PROBE message. This message comes in two formats. A fully truncated form containing only the OpCode (thus of length 2) is used to indicate a request for information. This message should be answered by the appropriate ACK_CONFIG. Further interpretations of this message are: 1. Missing ACK field implies that a request for ACK_CONFIG is made. Thus, this is a special case of the fully truncated mode. However, the provided fields are meant to be used in the setting of goals. 2. Individual fields with zero value are meant NOT to be changed in the received element. Same as for missing fields in truncated messages. Furthermore, in order to improve testing, a combination of values for main and high goals are conventionally expected to be used for defining two behaviours: 1. Suspended: Tuning Probe messages (TTL=0) should be sent and statistics maintained, but the duty cycle should not be changed - thus goals will never be achieved. The encoding are: Main Goal = 0x00 and High Goal = 0xFE. 2. Disable: No Tuning Probe message should be sent and statistics should not be gathered - averaged values should decay. The encoding are: Main Goal = 0x00 and High Goal = 0xFF

Name	Description
CSRMESH_TUNING_STATS_T	Current Asset State: Response to the request. The PartNumber indicates the current index of the message, the top bit of this field is used to indicate that more messages are available after this part. For example, a response made up of three messages will have part numbers 0x80, 0x81 and 0x02. Each message has a maximum of two neighbours. The combination of these responses and PROBE (TTL>0) are a means to establish an overall perspective of the entire Mesh Network
CSRMESH_WATCHDOG_INTERVAL_T	Watchdog interval state
CSRMESH_WATCHDOG_MESSAGE_T	CSRmesh Watchdog Model message types
CSRMESH_WATCHDOG_SET_INTERVAL_T	Upon receiving a WATCHDOG_SET_INTERVAL message, the device shall save the Interval and ActiveAfterTime fields into the Interval and ActiveAfterTime variables and respond with a WATCHDOG_INTERVAL message with the current variable values
nvm_cs_header_t	NVM key descriptor
nvm_versioned_header_t	Versioned header information
tuningStats_t	Tuning statistics of neighbouring devices

82 CSR_MESH_APP_EVENT_DATA_T Struct Reference

Data Fields

Data	Field
CSR_MESH_EVENT_T	event CSR Mesh Event.
CSR_MESH_OPERATION_STATUS_T	status CSR Mesh Operation Status.
void *	appCallbackDataPtr App Data.

83 CSR_MESH_APPEARANCE_T Struct Reference

Data Fields

Data	Field
CsrUint8	organization Identifies the organization which assigns device appearance values.
CsrUint16	value Appearance value.
CsrUint32	deviceHash deviceHash of remote device

84 CSR_MESH_ASSOCIATION_ATTENTION_DATA_T Struct Reference

Data Fields

Data	Field
CsrUint8	attract_attention Enable or disable attracting attention 0 - Do not attract attention 1 - Attract attention Refer to Handling 8 bit data types on XAP.
CsrUint16	duration Duration for which the attention is requested.

85 CSR_MESH_AUTH_CODE_T Struct Reference

Data Fields

Data	Field
CsrUint8	auth_code [CSR_MESH_AUTH_CODE_SIZE_IN_BYTES] CSRmesh 64 bit Authorisation Code.

86 CSR_MESH_BEARER_STATE_DATA_T Struct Reference

Data Fields

Data	Field
CsrUint16	bearerRelayActive Mask of bearer Relay state.
CsrUint16	bearerEnabled Mask of bearer Active state.
CsrUint16	bearerPromiscuous Mask of promiscuous mode.

87 CSR_MESH_CC_INFO_T Struct Reference

Data Fields

Data	Field
`CsrBool`	auth_code_considered_for_cc flag to indicate if Confirmation code is computed using auth code or not. This is only used if type is CSR_MESH_CONFIRMATION_CODE

88 CSR_MESH_CONFIG_BEARER_PARAM_T Struct Reference

Data Fields

Data	Field
CsrUint16	scan_duty_cycle CSRmesh scan duty cycle (0 - 100 percent)
CsrUint16	advertising_interval CSRmesh advertising interval in milliseconds.
CsrUint16	advertising_time CSRmesh advertising time in milliseconds.
CsrInt8	tx_power CSRmesh tx_power.
CsrUint8	default_ttl CSRmesh default ttl.

89 CSR_MESH_CONFIG_MSG_PARAMS_T Struct Reference

Data Fields

Data	Field
CsrUint8	tx_queue_size CSRmesh Max number of messages in transmit queue.
CsrUint8	relay_repeat_count CSRmesh Number of times to transmit a relayed message.
CsrUint8	device_repeat_count CsRmesh Number of times to transmit a message from this device.

90 CSR_MESH_CONFORMANCE_SIGNATURE_T Struct Reference

Data Fields

Data	Field
`CsrUint16`	conformance_sig [CSR_MESH_CONFORMANCE_SIGNATURE_LEN/2] Conformance signature.

91 CSR_MESH_DEVICE_APPEARANCE_T Struct Reference

Data Fields

Data	Field
CsrUint8	shortName [CSR_MESH_SHORT_NAME_LENGTH] short name of the device
CSR_MESH_APPEARANCE_T	appearance deviceApperance of the device

92 CSR_MESH_DEVICE_ID_UPDATE_T Struct Reference

CSR Mesh device id update data Type.

93 CSR_MESH_GROUP_ID_RELATED_DATA_T Struct Reference

Data Fields

Data	Field
CsrUint8	netId network Id of the nwk for which the group is set
CsrUint8	model MCP Model No.
CsrUint8	gpIdx Group Idx.
CsrUint8	instance Group Instance.
CsrUint16	gpId Group Id.

94 CSR_MESH_MASP_DEVICE_DATA_T Struct Reference

Data Fields

Data	Field
`CsrUint8`	netId Network ID.
`CsrUint16`	deviceId Device ID.
`CsrUint16 *`	deviceSign Pointer to Device Signature. Size of device signature is CSR_MESH_DEVICE_SIGN_SIZE_IN_WORDS. This signature is used to securely update network parameters.

95 CSR_MESH_MASP_DEVICE_IND_T Struct Reference

Data Fields

Data	Field
CsrUint32	deviceHash deviceHash of remote device
CSR_MESH_UUID_T	uuid CSRmesh 128-bit UUID of remote device.

96 CSR_MESH_MASP_DEVICE_SIGN_LIST_NODE_T Struct Reference

Data Fields

Data	Field
CsrUint16	deviceId Device ID.
CsrUint16	deviceSign [CSR_MESH_DEVICE_SIGN_SIZE_IN_WORDS] Device Signature. Size of device signature is CSR_MESH_DEVICE_SIGN_SIZE_IN_WORDS. This signature is used to securely update network parameters.
struct CSR_MESH_MASP_DEVICE_SIGN_LIST_NODE *	next Pointer to the next element of the linked list.

97 CSR_MESH_MESH_ID_DATA_T Struct Reference

Data Fields

Data	Field
CsrUint8	meshId [CSR_MESH_MESH_ID_SIZE_BYTES] CSR Mesh mesh_id.

98 CSR_MESH_NETID_LIST_T Struct Reference

Data Fields

Data	Field
CsrUint8	length List of existing mesh network IDs. Each octet value identifies one distinguished mesh network. This is defined as a flexible array and a field of this type will be part of union CSR_MESH_APP_CB_DATA_T. The array netIdList will have maximum number of array elements i.e. allowed by the maximum size of the union. An invalid network id is defined as 0xFF.
CsrUint8	netIDList [CSR_MESH_NO_OF_NWKS] list of Net-ids

99 CSR_MESH_NW_RELAY_T Struct Reference

Data Fields

Data	Field
CsrUint8	netId Nwk id for which realy is enable.
CsrBool	enable realy information

100 CSR_MESH_SEQ_CACHE_T Struct Reference

Data Fields

Data	Field
CsrUint8	seq_deviation Maximum Allowed threshold for Older Sequence Numbers.
CsrUint8	cached_dev_count Maximum Number of stored devices in the Look Up Table.
CSR_MESH_SQN_LOOKUP_TABLE_T *	seq_lookup_table Pointer to the memory allocated for Look Up Table. This should match to fit the cached_dev_count number of devices.

101 CSR_MESH_SQN_LOOKUP_TABLE_T Struct Reference

Data Fields

Data	Field
CsrUint8	counter Counter to keep track of oldest cache entry.
CsrUint16	src Source Id.
CsrUint32	sqn Sequence Number.

102 CSR_MESH_STACK_VERSION_T Struct Reference

Data Fields

Data	Field
CsrUint8	major [CSR_MESH_STACK_MAJOR_LEN] stack version major.
CsrUint8	minor [CSR_MESH_STACK_MINOR_LEN] stack version minor.
CsrUint8	variant [CSR_MESH_STACK_VARIANT_LEN] stack version variant.

103 CSR_MESH_TRANSMIT_STATE_T Struct Reference

Data Fields

Data	Field
CsrUint16	bearerRelayActive Bearer realy information.
CsrUint16	bearerEnabled Bearer Enable information.
CsrUint16	bearerPromiscuous Bearer Promiscuous information.
CSR_MESH_NW_RELAY_T	relay Mcp realy information.
CsrBool	maspRelayEnable Masp realy information.

104 CSR_MESH_TTL_T Struct Reference

Data Fields

Data	Field
CSR_MESH_MESSAGE_T	msgType Msg type - Whether MASP or MCP.
CsrUint8	ttl TTL Value.

105 CSR_MESH_UUID_T Struct Reference

Data Fields

Data	Field
`CsrUint8`	uuid [16] CSRmesh 128-bit UUID.

106 CSR_MESH_VID_PID_VERSION_T Struct Reference

Data Fields

Data	Field
CsrUint16	vendor_id Vendor Identifier.
CsrUint16	product_id Product Identifier.
CsrUint32	version Version Number.

107 CSR_SCHED_ADV_DATA_T Struct Reference

Data Fields

Data	Field
CSR_SCHED_ADV_PARAMS_T	adv_params
CsrUint8	ad_data [MAX_USER_ADV_DATA_LEN]
CsrUint8	ad_data_length
CsrUint8	scan_rsp_data [MAX_USER_ADV_DATA_LEN] User Scan response data.
CsrUint8	scan_rsp_data_length User Scan response data length.

108 CSR_SCHED_ADV_PARAMS_T Struct Reference

Data Fields

Data	Field
TYPED_BD_ADDR_T	bd_addr CSRmesh BD-ADDR.
ls_advert_type	adv_type CSRmesh Advertisment Type.
gap_role	role CSRmesh Gap Role.
gap_mode_connect	connect_mode CSRmesh Gap connect mode.
gap_mode_discover	discover_mode CSRmesh Gap discover mode.
gap_mode_bond	bond CSRmesh Gap bond mode.
gap_mode_security	security_mode CSRmesh Gap security mode.

109 CSR_SCHED_GATT_EVENT_DATA_T Struct Reference

Data Fields

Data	Field
CsrUint16	cid Channel id for which the GATT event is received.
CsrBool	is_gatt_bearer_ready Flag to identify the Gatt connection state.
CsrUint16	conn_interval Connection interval value updated during GATT connection.
CsrBool	is_notification_enabled flag to identify the cccd state

110 CSR_SCHED_GENERIC_LE_PARAM_T Struct Reference

Data Fields

Data	Field
CSR_SCHED_SCAN_TYPE_T	scan_param_type CSRmesh Scan Parameter type.
CSR_SCHED_SCAN_PARAM_T	scan_param CSRmesh Scan Parameter.
CsrUint32	advertising_interval CSRmesh advertising interval in milliseconds.
CsrUint32	advertising_time CSRmesh advertising time in milliseconds.

111 CSR_SCHED_LE_PARAMS_T Struct Reference

Data Fields

Data	Field
CSR_SCHED_MESH_LE_PARAM_T	mesh_le_param CSRmesh MESH-LE parameter.
CSR_SCHED_GENERIC_LE_PARAM_T	generic_le_param CSRmesh Generic LE parameter.

112 CSR_SCHED_MESH_LE_PARAM_T Struct Reference

Data Fields

Data	Field
CsrBool	is_le_bearer_ready Flag to check whether bearer is ready.
CSR_SCHED_MESH_TX_PARAM_T	tx_param CSRmesh transmit parameter.

113 CSR_SCHED_MESH_TX_BUFFER_T Struct Reference

CSR Mesh Sched TX queue buffer Type.

114 CSR_SCHED_MESH_TX_PARAM_T Struct Reference

Data Fields

Data	Field
CsrUint8	tx_queue_size CSRmesh Transmit Queue Size.
CsrUint8	relay_repeat_count CSRmesh Relay message repeat count.
CsrUint8	device_repeat_count CSRmesh Device message repeat count.

115 CSR_SCHED_SCAN_DUTY_PARAM_T Struct Reference

Data Fields

Data	Field
CsrUint16	scan_duty_cycle scan duty cycle 0-100: duty cycle percentage in 1 percent steps for value 101-255 duty cycle = (value - 100)/10 percentage in 0.1 percent steps 0.1 to 15.5%
CsrUint16	min_scan_slot scan slot value ranges from 0x0004 to 0x4000

116 CSR_SCHED_SCAN_WINDOW_PARAM_T Struct Reference

Data Fields

Data	Field
CsrUint32	scan_window scan window param in milliseconds value ranges from 2.5 msecs to 10240 msecs
CsrUint32	scan_interval scan interval param in milliseconds value ranges 2.5 msecs to 10240 msecs

117 CSRMESH_ACTION_ACTION_STATUS_T Struct Reference

Data Fields

Data	Field
CsrUint32	actionids Bit mask indicating which Actions are stored. The bit position (for clarity 00000000 00000000 00000000 00000001 = actionID #1 only) determines the Action ID (from 1 to 31)
CsrUint8	maxactionssupported Max number of actions supported by a Node device.
CsrUint8	tid Transaction ID. The TID or transaction identifier field is an 8-bit integer that identifies a given message with a known transaction from a source device.

118 CSRMESH_ACTION_DELETE_ACK_T Struct Reference

Data Fields

Data	Field
CsrUint32	actionids bit mask indicating which Action which have been deleted. The bit position (LSB = actionID #1) determines the Action ID (from 1 to 31).
CsrUint8	tid Transaction ID. The TID or transaction identifier field is an 8-bit integer that identifies a given message with a known transaction from a source device.

119 CSRMESH_ACTION_DELETE_T Struct Reference

Data Fields

Data	Field
CsrUint32	actionids Bit mask indicating which Action to be deleted. The bit position (for clarity 00000000 00000000 00000000 00000001 = actionID #1 only) determines the Action ID (from 1 to 31)
CsrUint8	tid Transaction ID. The TID or transaction identifier field is an 8-bit integer that identifies a given message with a known transaction from a source device.

120 CSRMESH_ACTION_GET_ACTION_STATUS_T Struct Reference

Data Fields

Data	Field
CsrUint8	tid Transaction ID. The TID or transaction identifier field is an 8-bit integer that identifies a given message with a known transaction from a source device.

121 CSRMESH_ACTION_GET_T Struct Reference

Data Fields

Data	Field
`CsrUint8`	tid Transaction ID. The TID or transaction identifier field is an 8-bit integer that identifies a given message with a known transaction from a source device.

122 CSRMESH_ACTION_SET_ACTION_ACK_T Struct Reference

Data Fields

Data	Field
CsrUint8	tid Transaction ID. The TID or transaction identifier field is an 8-bit integer that identifies a given message with a known transaction from a source device.

123 CSRMESH_ACTION_SET_ACTION_INFO_EXT_T Struct Reference

Data Fields

Data	Field
CSRMESH_ACTION_SET_ACTION_INFO_T	set_action Set Action strcture containing the action information to be filled.
uint8	buf [11] The buffer to store the action.
uint8	buf_size The size of the action buffer stored.

124 CSRMESH_ACTION_SET_ACTION_INFO_T Struct Reference

Detailed Description

This message defines the action which needs to be taken. This is a multiple parts message, each part will be acknowledged through an ACTION_SET_ACK (0x51).

125 CSRMESH_ACTION_SET_ACTION_T Struct Reference

Data Fields

Data	Field
CsrUint8	actionid Binary: b'PPCAAAAA' : PP part number (max of 4), C is 1 if this is the last part, AAAAA 5bits of Action Identifier.
CsrUint8	payload [8/size_in_bytes(CsrUint8)] Structured payload.
CsrUint8	payload_len Length of the payload field.
CsrUint8	tid Transaction ID. The TID or transaction identifier field is an 8-bit integer that identifies a given message with a known transaction from a source device.

126 CSRMESH_ACTUATOR_GET_TYPES_T Struct Reference

Detailed Description

Upon receiving an ACTUATOR_GET_TYPES message, the device responds with an ACTUATOR_TYPES message with a list of supported types greater than or equal to the FirstType field. If the device does not support any types greater than or equal to FirstType, it sends an ACTUATOR_TYPES message with a zero length Types field.

127 CSRMESH_ACTUATOR_GET_VALUE_ACK_T Struct Reference

Data Fields

Data	Field
`sensor_type_t`	type Actuator type. The Type field is a 16-bit value that determines the actuator type.
`CsrUint8`	tid Transaction identifier. The TID, or transaction identifier field is an 8-bit integer that identifies a given message with a known transaction from a source device.

128 CSRMESH_ACTUATOR_SET_VALUE_T Struct Reference

Data Fields

Data	Field
`sensor_type_t`	type Actuator type. The Type field is the actuator type value being set.
`CsrUint8`	value [4/size_in_bytes(CsrUint8)] Actuator value. The Value field is the value that the actuator type.
`CsrUint8`	value_len Length of the value field.
`CsrUint8`	tid Transaction identifier. The TID, or transaction identifier field is an 8-bit integer that identifies a given message with a known transaction from a source device.

129 CSRMESH_ACTUATOR_TYPES_T Struct Reference

Data Fields

Data	Field
sensor_type_t	types [8/size_in_bytes(sensor_type_t)] Array of sensor types supported (16 bits per type). The Types field is a variable length sequence of sensor types that the device sending this message supports. The sensor types included in this message is ordered by sensor type number and is always higher than the FirstType in the corresponding ACTUATOR_GET_TYPES message. Note: This field can be zero octets in length if a device does not support any sensor types greater than or equal to the FirstType in the corresponding ACTUATOR_GET_TYPES message.
CsrUint8	types_len Length of the types field.
CsrUint8	tid Transaction identifier. The TID, or transaction identifier field is an 8-bit integer that identifies a given message with a known transaction from a source device.

130 CSRMESH_ACTUATOR_VALUE_ACK_T Struct Reference

Data Fields

Data	Field
`sensor_type_t`	type Actuator type. The Type field is a 16-bit value that determines the actuator type.
`CsrUint8`	value [4/size_in_bytes(CsrUint8)] Actuator value associated to the provided type.
`CsrUint8`	value_len Length of the value field.
`CsrUint8`	tid Transaction identifier. The TID, or transaction identifier field is an 8-bit integer that identifies a given message with a known transaction from a source device.

131 CSRMESH_ASSET_ANNOUNCE_T Struct Reference

Data Fields

Data	Field
CsrInt8	txpower Asset TxPower.
CsrUint16	interval Interval between asset broadcasts.
CsrUint16	sideeffects Side effects for this asset.

132 CSRMESH_ASSET_GET_STATE_T Struct Reference

Data Fields

Data	Field
CsrUint8	tid Transaction identifier. The TID, or transaction identifier field is an 8-bit integer that identifies a given message with a known transaction from a source device.

133 CSRMESH_ASSET_SET_STATE_T Struct Reference

Detailed Description

Setting Asset State: Upon receiving an ASSET_SET_STATE message, the device saves the Interval, SideEffects, ToDestination, TxPower, Number of Announcements and AnnounceInterval fields into the appropriate state values. It then responds with an ASSET_STATE message with the current state information.

134 CSRMESH_ASSET_STATE_T Struct Reference

Data Fields

Data	Field
CsrUint16	interval Interval between asset broadcasts.
CsrUint16	sideeffects Side effects for this asset.
CsrUint16	todestinationid Asset announce destination.
CsrInt8	txpower Asset TxPower.
CsrUint8	numannounces Number of times to send announce.
CsrUint8	announceinterval Time between announce repeats.
CsrUint8	tid Transaction identifier. The TID, or transaction identifier field is an 8-bit integer that identifies a given message with a known transaction from a source device.

135 CSRMESH_ATTENTION_SET_STATE_T Struct Reference

Detailed Description

Setting Flashing State: Upon receiving an ATTENTION_SET_STATE message, the device saves the AttractAttention and AttentionDuration fields into the appropriate state value. It then responds with an ATTENTION_STATE message with the current state information. If the AttractAttention field is set to 0x01 and the AttentionDuration is not 0xFFFF, then any existing attention timer is cancelled and a new attention timer is started that will expire after AttentionDuration milliseconds. If the AttractAttention field is set to 0x01 and the AttentionDuration field is 0xFFFF, then the attention timer is ignored. If the AttractAttention field is set to 0x00, then the attention timer is cancelled if it is already running.

136 CSRMESH_ATTENTION_STATE_T Struct Reference

Data Fields

Data	Field
`CsrBool`	attractattention Enable or disable attracting of attention.
`CsrUint16`	duration Duration for attracting attention.
`CsrUint8`	tid Transaction identifier. The TID, or transaction identifier field is an 8-bit integer that identifies a given message with a known transaction from a source device.

137 CSRMESH_BATTERY_GET_STATE_T Struct Reference

Detailed Description

Getting Battery State: Upon receiving a BATTERY_GET_STATE message, the device responds with a BATTERY_STATE message with the current state information.

138 CSRMESH_BATTERY_STATE_T Struct Reference

Data Fields

Data	Field
CsrUint8	batterylevel Battery level.
CsrUint8	batterystate State of the battery.
CsrUint8	tid Transaction identifier. The TID, or transaction identifier field is an 8-bit integer that identifies a given message with a known transaction from a source device.

139 CSRMESH_BEACON_BEACON_STATUS_T Struct Reference

Data Fields

Data	Field
CsrUint8	beacontype 0x00=CSR, 0x01=iBeacon, 0x02=Eddystone URL, 0x03=Eddystone UID, 0x04=LTE direct, 0x05=Lumicast.
CsrUint8	beaconinterval Time between beacon transmissions in 100ths of seconds (0..100). Setting this to 0 stops that beacon type transmitting.
CsrUint16	meshinterval How many minutes between mesh wakeups (1..N).
CsrUint8	meshtime How long node stays as a mesh node (10s of seconds).
CsrUint8	txpower -127..+127 dbM.
CsrUint8	batterylevel 0..100 (percent).
CsrUint16	payloadid ID of latest received payload for this beacon type, as returned by beacon payload ack.
CsrUint8	tid Transaction ID. The TID or transaction identifier field is an 8-bit integer that identifies a given message with a known transaction from a source device.

140 CSRMESH_BEACON_GET_BEACON_STATUS_T Struct Reference

Data Fields

Data	Field
CsrUint8	beacontype 0x00=CSR, 0x01=iBeacon, 0x02=Eddystone URL, 0x03=Eddystone UID, 0x04=LTE direct, 0x05=Lumicast.
CsrUint8	tid Transaction ID. The TID or transaction identifier field is an 8-bit integer that identifies a given message with a known transaction from a source device.

141 CSRMESH_BEACON_GET_PAYLOAD_T Struct Reference

Data Fields

Data	Field
CsrUint8	payloadtype 0x00=CSR, 0x01=iBeacon, 0x02=Eddystone URL, 0x03=Eddystone UID, 0x4=LTE direct, 0x5=Lumicast.

142 CSRMESH_BEACON_GET_TYPES_T Struct Reference

Data Fields

Data	Field
CsrUint8	tid Transaction ID. The TID or transaction identifier field is an 8-bit integer that identifies a given message with a known transaction from a source device.

143 CSRMESH_BEACON_PAYLOAD_ACK_T Struct Reference

Data Fields

Data	Field
CsrUint8	payloadtype 0x00=CSR, 0x01=iBeacon, 0x02=Eddystone URL, 0x03=Eddystone UID, 0x04=LTE direct, 0x05=Lumicast.
CsrUint16	ackornack 0 means success, set bits indicate missing packet numbers 1..16.
CsrUint16	payloadid Payload ID which was received, or 0xFFFF if no final packet was received.

144 CSRMESH_BEACON_SET_PAYLOAD_T Struct Reference

Data Fields

Data	Field
CsrUint8	packetframing Bits 0..3 defines packet number 1..16, Bits 4..7 gives total number of packets 1..16.
CsrUint16	payloadid ID of this payload (so beacon can know whether it needs to update).
CsrUint8	beacontype 0x00=CSR, 0x01=iBeacon, 0x02=Eddystone URL, 0x03=Eddystone UID, 0x04=LTE direct, 0x05=Lumicast.
CsrUint8	payloadlength Total length of payload data being sent.
CsrUint8	payloadoffset Offset into Payload. If this number is 0..0xBF then it is an offset into the raw advert data. If this number is 0xC0.0xFF then it is an offset into non-advertising data (e.g. crypto information).
CsrUint8	payload [MAX_BEACON_PAYLOAD_LENGTH] Payload data - up to 4 bytes total (segmented)
CsrUint8	payload_len Length of the payload field.

145 CSRMESH_BEACON_SET_STATUS_T Struct Reference

Detailed Description

May be sent to a beacon to set its status. More than one such command can be sent to set the intervals for different beacon types, which need not be the same. This message also allows for the wakeup intervals to be set if the beacon elects to only be on the mesh sporadically. The time is always with respect to the beacon's internal clock and has no absolute meaning. This message will be answered through BEACON_STATUS.

146 CSRMESH_BEACON_TYPES_T Struct Reference

Data Fields

Data	Field
CsrUint16	beacontype 0x01=CSR, 0x02=iBeacon, 0x04=Eddystone URL, 0x08=Eddystone UID, 0x10=LTE direct, 0x20=Lumicast.
CsrUint8	batterylevel 0..100 (percent)
CsrUint16	timesincelastmessage Time in 10s of seconds since last message received. 0 means message received less than 10 seconds ago, 1 means 10..19 seconds and so on. The special value 0xFFFF means no messages have been received from this beacon since the proxy was last started. If this message is received direct from a beacon rather than from a proxy, this field will always be 0.
CsrUint8	tid Transaction ID. The TID or transaction identifier field is an 8-bit integer that identifies a given message with a known transaction from a source device.

147 CSRMESH_BEACONPROXY_ADD_T Struct Reference

Detailed Description

Message can be sent to a beacon-proxy to add to the list of devices it manages. This request will be acknowledged using a Proxy Command Status Devices. Up to four device ID can be specified in this message: Group can be defined. This message also permits the flushing of pending messages for managed IDs.

148 CSRMESH_BEACONPROXY_COMMAND_STATUS_DEVICES_T Struct Reference

Data Fields

Data	Field
CsrUint8	tid Transaction ID. The TID or transaction identifier field is an 8-bit integer that identifies a given message with a known transaction from a source device.

149 CSRMESH_BEACONPROXY_DEVICES_T Struct Reference

Data Fields

Data	Field
`CsrUint8`	packetnumber Packet number. Bit 7 will be set if this is the final packet. Following fields form the payload, and will be spread across multiple packets as required.
`CsrUint8`	tid Transaction ID. The TID or transaction identifier field is an 8-bit integer that identifies a given message with a known transaction from a source device.
`CsrUint8`	numberofgroups Number of groups being managed.
`CsrUint16`	numberofdevices Number of device being managed.
`CsrUint16 *`	groups Group IDs of managed groups.
`CsrUint16 *`	devices Device IDs of managed groups.

150 CSRMESH_BEACONPROXY_GET_DEVICES_T Struct Reference

Data Fields

Data	Field
CsrUint8	tid Transaction ID. The TID or transaction identifier field is an 8-bit integer that identifies a given message with a known transaction from a source device.

151 CSRMESH_BEACONPROXY_PROXY_STATUS_T Struct Reference

Data Fields

Data	Field
CsrUint16	nummanagednodes How many nodes this proxy is currently being a proxy for. If the proxy has been told to be a proxy for device address 0, this will be 0xFFFF.
CsrUint8	nummanagedgroups How many groups this proxy is managing.
CsrUint16	numqueuedtxmsgs How many messages are currently queued at this proxy for transmission to beacons.
CsrUint16	numqueuedrxmsgs How many messages have been received from beacons at this proxy since its status was last queried.
CsrUint16	maxqueuesize Current maximum queue size.

152 CSRMESH_BEACONPROXY_REMOVE_T Struct Reference

Data Fields

Data	Field
`CsrUint8`	numdevices Bits 0..2 How many device are in this message. Queued messages for these devices will be cleared.
`CsrUint8`	deviceaddresses [8/size_in_bytes(CsrUint8)] Device addresses to remove. These can be group addresses. Also, the special address '0' can be used to cease making this a proxy for all devices.
`CsrUint8`	deviceaddresses_len Length of the deviceaddresses field.
`CsrUint8`	tid Transaction ID. The TID or transaction identifier field is an 8-bit integer that identifies a given message with a known transaction from a source device.

153 CSRMESH_BEARER_GET_STATE_T Struct Reference

Data Fields

Data	Field
CsrUint8	tid Transaction identifier.

154 CSRMESH_BEARER_SET_STATE_T Struct Reference

Detailed Description

Setting Bearer State: Upon receiving a BEARER_SET_STATE message, where the destination address is the device ID of this device, the device saves the BearerRelayActive, BearerEnabled, and BearerPromiscuous fields into the appropriate state value. Then the device responds with a BEARER_STATE message with the current state information.

155 CSRMESH_BEARER_STATE_T Struct Reference

Data Fields

Data	Field
CsrUint16	bearerrelayactive Bitfield of active relay bearers.
CsrUint16	bearerenabled Bitfield of enabled bearers.
CsrUint16	bearerpromiscuous Relay all messages, regardless of MAC.
CsrUint8	tid Transaction identifier.

156 CSRMESH_CONFIG_DEVICE_IDENTIFIER_T Struct Reference

Data Fields

Data	Field
CsrUint8	devicehash [8/size_in_bytes(CsrUint8)] Hash of the DeviceUUID. The DeviceHash field is a 64-bit hash of the DeviceUUID that is used to identify this device.
CsrUint8	tid Transaction identifier. The TID, or transaction identifier field is an 8-bit integer that identifies a given message with a known transaction from a source device.

157 CSRMESH_CONFIG_DISCOVER_DEVICE_T Struct Reference

Data Fields

Data	Field
CsrUint8	tid Transaction identifier. The TID, or transaction identifier field is an 8-bit integer that identifies a given message with a known transaction from a source device.

158 CSRMESH_CONFIG_GET_INFO_T Struct Reference

Data Fields

Data	Field
CsrUint8	info The type of information being requested. The Info field is an 8-bit unsigned integer that enumerates the information being requested. The following values are defined: 0x00 = UUID Low, 0x01 UUID High, 0x02 = Model Low, 0x03 = Model High, 0x04 = VID_PID_Version, 0x05 = Appearance, 0x06 = LastETag, 0x07 = Conformance, 0x08 = Stack Version.
CsrUint8	tid Transaction identifier. The TID, or transaction identifier field is an 8-bit integer that identifies a given message with a known transaction from a source device.

159 CSRMESH_CONFIG_GET_PARAMETERS_T Struct Reference

Data Fields

Data	Field
CsrUint8	tid Transaction identifier. The TID, or transaction identifier field is an 8-bit integer that identifies a given message with a known transaction from a source device.

160 CSRMESH_CONFIG_INFO_T Struct Reference

Data Fields

Data	Field
CsrUint8	info The type of information. The Info field is an 8-bit unsigned integer that enumerates the information being requested. The following values are defined:0x00 = UUID Low, 0x01 = UUID High, 0x02 = Model Low, 0x03 = Model High.
CsrUint8	information [8/size_in_bytes(CsrUint8)] Information requested. The Information field is a 64-bit value that contains information determined by the Info field. If the Info field is 0x00, the Information field contains the least significant 64-bits of the DeviceUUID. If the Info field is 0x01, the Information field contains the most significant 64-bits of the DeviceUUID. If the Info field is 0x02, the Information field contains the least significant 64-bits of the ModelsSupported bit-field. If the Info field is 0x03, the Information field contains the most significant 64-bits of the ModelsSupported bit-field.
CsrUint8	information_len Length of the information field.
CsrUint8	tid Transaction identifier. The TID, or transaction identifier field is an 8-bit integer that identifies a given message with a known transaction from a source device.

161 CSRMESH_CONFIG_LAST_SEQUENCE_NUMBER_T Struct Reference

Detailed Description

Upon receiving a CONFIG_LAST_SEQUENCE_NUMBER message from a trusted device, the local device updates the SequenceNumber to at least one higher than the LastSequenceNumber in the message. Note: A trusted device means a device that is not only on the same CSRmesh network, having the same network key, but also interacted with in the past. This message is most useful to check if a device has been reset, for example when the batteries of the device are changed, but it does not remember its last sequence number in non-volatile memory.

162 CSRMESH_CONFIG_PARAMETERS_T Struct Reference

Data Fields

Data	Field
CsrUint16	txinterval The transmit interval for this device. The TxInterval field is a 16-bit unsigned integer in milliseconds that determines the interval between transmitting messages.
CsrUint16	txduration How long a single transmission should occur for. The TxDuration, or transmit duration field is a 16-bit unsigned integer in milliseconds that determines the duration of transmission for a single message.
CsrUint8	rxdutycycle How much time this device listens for messages. The RxDutyCycle, or receiver duty cycle field is an 8-bit unsigned integer in 1/255ths of a second that determines how often the device listens for CSRmesh messages. Note: The value 0x00 implies that the device does not listen for messages. The value 0xFF implies that the device continuously listens for messages.
CsrInt8	txpower How loud should this device transmit messages. The TxPower, or transmit power field is an 8-bit signed integer in decibels that determines the radio transmit power for a device.
CsrUint8	ttl The initial default time to live for messages. The TimeToLive field is an 8-bit unsigned integer that determines the default value for the TTL packet field in an MTL packet.
CsrUint8	tid Transaction identifier. The TID, or transaction identifier field is an 8-bit integer that identifies a given message with a known transaction from a source device.

163 CSRMESH_CONFIG_RESET_DEVICE_T Struct Reference

Data Fields

Data	Field
CsrUint8	signature [8/size_in_bytes(CsrUint8)] 8 lowest bytes of HMAC(DHM_KEY,DeviceHash).

164 CSRMESH_CONFIG_SET_DEVICE_IDENTIFIER_T Struct Reference

Data Fields

Data	Field
CsrUint8	devicehash [8/size_in_bytes(CsrUint8)] The DeviceHash for the device that will have a new DeviceID.
CsrUint16	deviceid The new DeviceID for the device identified by DeviceHash.

165 CSRMESH_CONFIG_SET_PARAMETERS_T Struct Reference

Data Fields

Data	Field
CsrUint16	txinterval The transmit interval for this device. The TxInterval field is a 16-bit unsigned integer in milliseconds that determines the interval between transmitting messages.
CsrUint16	txduration How long a single transmission should occur for. The TxDuration, or transmit duration field is a 16-bit unsigned integer in milliseconds that determines the duration of transmission for a single message.
CsrUint8	rxdutycycle How much time this device listens for messages. The RxDutyCycle, or receiver duty cycle field is an 8-bit unsigned integer in 1/255ths of a second that determines how often the device listens for CSRmesh messages. Note: The value 0x00 implies that the device does not listen for messages. The value 0xFF implies that the device continuously listens for messages.
CsrInt8	txpower How loud the device transmits messages. The TxPower, or transmit power field is an 8-bit signed integer in decibels that determines the radio transmit power for a device.
CsrUint8	timetolive The initial default time to live for messages. The TimeToLive field is an 8-bit unsigned integer that determines the default value for the TTL packet field in an MTL packet.
CsrUint8	tid Transaction identifier. The TID, or transaction identifier field is an 8-bit integer that identifies a given message with a known transaction from a source device.

166 CSRMESH_DATA_BLOCK_SEND_T Struct Reference

Data Fields

Data	Field
CsrUint8	datagramoctets [10/size_in_bytes(CsrUint8)] Datagram of octets.
CsrUint8	datagramoctets_len Length of the datagramoctets field.

167 CSRMESH_DATA_STREAM_FLUSH_T Struct Reference

Detailed Description

Flushing Data: Upon receiving a DATA_STREAM_FLUSH message, the device saves the StreamSN field into the StreamSequenceNumber model state and responds with DATA_STREAM_RECEIVED with the StreamNESN field set to the value of the StreamSequenceNumber model state. The device also flushes all partially received stream data from this peer device.

168 CSRMESH_DATA_STREAM_RECEIVED_T Struct Reference

Data Fields

Data	Field
CsrUint16	streamnesn Stream next expected sequence number.

169 CSRMESH_DATA_STREAM_SEND_T Struct Reference

Data Fields

Data	Field
CsrUint16	streamsn Sequence number of the first octet in StreamOctets.
CsrUint8	streamoctets [8/size_in_bytes(CsrUint8)] Arbitrary stream of data.
CsrUint8	streamoctets_len Length of the streamoctets field.

170 CSRMESH_DIAGNOSTIC_STATE_T Struct Reference

Detailed Description

When received this message is interpreted as to reconfigure the set of information collected. Statistics gathering can be turned on/off ? in the off mode no measurement of messages count and RSSI measurements will be made. RSSI binning can be stored, such that collection ALL messages? RSSI (MASP/MCP, irrespective of encoding) are split between a given number of bin, each of equal dimensions. Masking of individual broadcast channel can be specified, resulting in the collection of information specifically on the selected channels. A REST bit is also available. When present all the accumulated data will be cleared and all counters restarted. Note that it is possible to change various configurations without the RESET flag, this will result in the continuation of accumulation and therefore incoherent statistics.

171 CSRMESH_EVENT_DATA_T Struct Reference

CSRmesh Event Data.

172 CSRMESH_EXTENSION_CONFLICT_T Struct Reference

Data Fields

Data	Field
CsrUint8	extensionhash [6/size_in_bytes(CsrUint8)] 48bits Hash (SHA-256) of Text supplied by OEM for characterisation of intended usage
CsrUint16	proposedopcode Proposed OpCode - start of range requested.
CsrUint8	reason Code describing reason for rejection.

173 CSRMESH_EXTENSION_REQUEST_T Struct Reference

Detailed Description

Request for Extension OpCode to be approved by the whole Mesh. A device wanting to use an OpCode, makes a request to the entire Mesh Network. This message is issued to target identity 0. The device waits some time, proportional to the size of the Mesh network and only after this period, messages using these proposed OpCode are used. Device receiving this message and wanting to oppose the usage of such code will respond to the source node with a CONFLICT. In case no conflict is known and the OpCode is for a message the node is interested in implementing (through comparison with hash value), a record of the OpCode and its mapping is kept. Request messages are relayed in cases of absence of conflict. The hash function is SHA-256, padded as per SHA-256 specifications2, for which the least significant 6 bytes will be used in the message. The range parameter indicates the maximum number of OpCode reserved from the based provided in the Proposed OpCode field. The last OpCode reserved is determined through the sum of the Proposed OpCode with the range value. This range parameter varies from 0 to 127, leaving the top bit free.

174 CSRMESH_FIRMWARE_GET_VERSION_T Struct Reference

Detailed Description

Get firmwre verison: Upon receiving a FIRMWARE_GET_VERSION the device reponds with a FIRMWARE_VERSION message containing the current firmware version

175 CSRMESH_FIRMWARE_UPDATE_ACKNOWLEDGED_T Struct Reference

Data Fields

Data	Field
`CsrUint8`	tid Transaction ID. The TID or transaction identifier field is an 8-bit integer that identifies a given message with a known transaction from a source device.

176 CSRMESH_FIRMWARE_UPDATE_REQUIRED_T Struct Reference

Data Fields

Data	Field
CsrUint8	tid Transaction ID. The TID or transaction identifier field is an 8-bit integer that identifies a given message with a known transaction from a source device.

177 CSRMESH_FIRMWARE_VERSION_T Struct Reference

Data Fields

Data	Field
CsrUint16	majorversion Firmware Major Version.
CsrUint16	minorversion Firmware Minor Version.
CsrUint8	reserved [4/size_in_bytes(CsrUint8)] Not used.
CsrUint8	tid Transaction ID. The TID or transaction identifier field is an 8-bit integer that identifies a given message with a known transaction from a source device.

178 CSRMESH_GROUP_GET_MODEL_GROUPID_T Struct Reference

Data Fields

Data	Field
CsrUint8	model Model number.
CsrUint8	groupindex Index of the group.
CsrUint8	tid Transaction identifier. The TID, or transaction identifier field is an 8-bit integer that identifies a given message with a known transaction from a source device.

179 CSRMESH_GROUP_GET_NUMBER_OF_MODEL_GROUPIDS_T Struct Reference

Detailed Description

Getting Number of Group IDs: Upon receiving a GROUP_GET_NUMBER_OF_MODEL_GROUPS message, where the destination address is the DeviceID of this device, the device responds with a GROUP_NUMBER_OF_MODEL_GROUPS message with the number of Group IDs that the given model supports on this device.

180 CSRMESH_GROUP_MODEL_GROUPID_T Struct Reference

Data Fields

Data	Field
`CsrUint8`	model Model number.
`CsrUint8`	groupindex Index of the GroupID in this model.
`CsrUint8`	instance Instance of the group.
`CsrUint16`	groupid Group identifier.
`CsrUint8`	tid Transaction identifier. The TID, or transaction identifier field is an 8-bit integer that identifies a given message with a known transaction from a source device.

181 CSRMESH_GROUP_NUMBER_OF_MODEL_GROUPIDS_T Struct Reference

Data Fields

Data	Field
CsrUint8	model Model number.
CsrUint8	numgroups Number of model groups.
CsrUint8	tid Transaction identifier. The TID, or transaction identifier field is an 8-bit integer that identifies a given message with a known transaction from a source device.

182 CSRMESH_GROUP_SET_MODEL_GROUPID_T Struct Reference

Data Fields

Data	Field
CsrUint8	model Model number.
CsrUint8	groupindex Index of the group.
CsrUint8	instance Instance of the group.
CsrUint16	groupid Group identifier.
CsrUint8	tid Transaction identifier. The TID, or transaction identifier field is an 8-bit integer that identifies a given message with a known transaction from a source device.

183 CSRMESH_LARGEOBJECTTRANSFER_ANNOUNCE_T Struct Reference

Detailed Description

A node wanting to provide a large object to neighbouring Mesh Nodes issues an ANNOUNCE with the associated content type. This message will have TTL=0, thus will only be answered by its immediate neighbours. The ANNOUNCE has the total size of the packet to be issued. The format and encoding of the large object is subject to the provided type and is out of scope of this document. The destination ID can either be 0, a group or a specific Device ID. In case the destination ID is not zero, only members of the group (associated with the LOT model) or the device with the specified Device ID responds with the intent to download the object for their own consumption. Every other node either ignores or accepts the offer for the purpose of relaying the packet.

184 CSRMESH_LARGEOBJECTTRANSFER_INTEREST_T Struct Reference

Data Fields

Data	Field
CsrUint8	serviceid [8/size_in_bytes(CsrUint8)] Least Significant 63 bits of Service ID to be used for transfer - Most Significant Bit set to 1 if the source intends to use the packet.

185 CSRMESH_LIGHT_GET_STATE_T Struct Reference

Data Fields

Data	Field
CsrUint8	tid Transaction identifier. The TID, or transaction identifier field is an 8-bit integer that identifies a given message with a known transaction from a source device.

186 CSRMESH_LIGHT_GET_WHITE_T Struct Reference

Data Fields

Data	Field
CsrUint8	tid Transaction identifier. The TID, or transaction identifier field is an 8-bit integer that identifies a given message with a known transaction from a source device.

187 CSRMESH_LIGHT_SET_COLOR_TEMP_T Struct Reference

Data Fields

Data	Field
CsrUint16	colortemperature Colour temperature.
CsrUint16	tempduration Time over which colour temperature will change.
CsrUint8	tid Transaction identifier. The TID, or transaction identifier field is an 8-bit integer that identifies a given message with a known transaction from a source device.

188 CSRMESH_LIGHT_SET_LEVEL_T Struct Reference

Detailed Description

Setting Light Level: Upon receiving a LIGHT_SET_LEVEL_NO_ACK message, the device saves the Level field into the CurrentLevel model state. LevelSDState should be set to Idle. If ACK is requested, the device should respond with a LIGHT_STATE message.

189 CSRMESH_LIGHT_SET_POWER_LEVEL_T Struct Reference

Data Fields

Data	Field
CsrUint8	power Power state.
CsrUint8	level Light level.
CsrUint16	levelduration Time before light reaches desired level.
CsrUint16	sustain Time that light will stay at desired level.
CsrUint16	decay Time over which light will decay to zero light level.
CsrUint8	tid Transaction identifier. The TID, or transaction identifier field is an 8-bit integer that identifies a given message with a known transaction from a source device.

190 CSRMESH_LIGHT_SET_RGB_T Struct Reference

Data Fields

Data	Field
CsrUint8	level Light level.
CsrUint8	red Red light level.
CsrUint8	green Green light level.
CsrUint8	blue Blue light level.
CsrUint16	colorduration Time over which the colour will change.
CsrUint8	tid Transaction identifier. The TID, or transaction identifier field is an 8-bit integer that identifies a given message with a known transaction from a source device.

191 CSRMESH_LIGHT_SET_WHITE_T Struct Reference

Data Fields

Data	Field
CsrUint8	level White level.
CsrUint16	duration Time before light reaches desired level.
CsrUint8	tid Transaction identifier. The TID, or transaction identifier field is an 8-bit integer that identifies a given message with a known transaction from a source device.

192 CSRMESH_LIGHT_STATE_T Struct Reference

Data Fields

Data	Field
CsrUint8	power Power state.
CsrUint8	level Light level.
CsrUint8	red Red light level.
CsrUint8	green Green light level.
CsrUint8	blue Blue light level.
CsrUint16	colortemperature Colour temperature.
CsrUint8	supports Bit field of supported light behaviour.
CsrUint8	tid Transaction identifier. The TID, or transaction identifier field is an 8-bit integer that identifies a given message with a known transaction from a source device.

193 CSRMESH_LIGHT_WHITE_T Struct Reference

Data Fields

Data	Field
CsrUint8	level White level.
CsrUint8	tid Transaction identifier. The TID, or transaction identifier field is an 8-bit integer that identifies a given message with a known transaction from a source device.

194 CSRMESH_PING_REQUEST_T Struct Reference

Detailed Description

Ping Request: Upon receiving a PING_REQUEST message, the device responds with a PING_RESPONSE message with the TTLAtRx field set to the TTL value from the PING_REQUEST message, and the RSSIAtRx field set to the RSSI value of the PING_REQUEST message. If the bearer used to receive the PING_REQUEST message does not have an RSSI value, then the value 0x00 is used.

195 CSRMESH_PING_RESPONSE_T Struct Reference

Data Fields

Data	Field
CsrUint8	arbitrarydata [4/size_in_bytes(CsrUint8)] Arbitrary data.
CsrUint8	ttlatrx Time to live when received.
CsrUint8	rssiatrx Receiver signal strength when received.

196 CSRMESH_POWER_GET_STATE_T Struct Reference

Data Fields

Data	Field
CsrUint8	tid Transaction identifier. The TID, or transaction identifier field is an 8-bit integer that identifies a given message with a known transaction from a source device.

197 CSRMESH_POWER_SET_STATE_T Struct Reference

Detailed Description

Setting Power State: Upon receiving a POWER_SET_STATE_NO_ACK message, the device sets the PowerState state value to the PowerState field. It then responds with a POWER_STATE message with the current state information.

198 CSRMESH_POWER_STATE_T Struct Reference

Data Fields

Data	Field
CsrUint8	state Power state.
CsrUint8	tid Transaction identifier. The TID, or transaction identifier field is an 8-bit integer that identifies a given message with a known transaction from a source device.

199 CSRMESH_POWER_TOGGLE_STATE_T Struct Reference

Data Fields

Data	Field
CsrUint8	tid Transaction identifier. The TID, or transaction identifier field is an 8-bit integer that identifies a given message with a known transaction from a source device.

200 CSRMESH_SENSOR_GET_STATE_T Struct Reference

Data Fields

Data	Field
`sensor_type_t`	type Sensor type. The Type field is a 16-bit unsigned integer that determines the sensor types being configured.
`CsrUint8`	tid Transaction identifier. The TID, or transaction identifier field is an 8-bit integer that identifies a given message with a known transaction from a source device.

201 CSRMESH_SENSOR_GET_TYPES_T Struct Reference

Detailed Description

Upon receiving a SENSOR_GET_TYPES message, the device responds with a SENSOR_TYPES message with the list of supported types greater than or equal to the FirstType field. If the device does not support any types greater than or equal to the FirstType field, then it sends a SENSOR_TYPES message with zero-length Types field.

202 CSRMESH_SENSOR_MISSING_T Struct Reference

Data Fields

Data	Field
`sensor_type_t`	types [8/size_in_bytes(sensor_type_t)] A Types field is an array of one, two, three or four sensor types, with each value being a 16-bit unsigned integer that determines one of the sensor type that is missing.
`CsrUint8`	types_len Length of the types field.

203 CSRMESH_SENSOR_READ_VALUE_T Struct Reference

Data Fields

Data	Field
`sensor_type_t`	type Sensor type. The Type field is a 16-bit unsigned integer that determines the sensor type being read.
`sensor_type_t`	type2 Sensor type. The Type field is a 16-bit unsigned integer that determines the second sensor type being read. This field is optional.
`CsrUint8`	tid Transaction identifier. The TID, or transaction identifier field is an 8-bit integer that identifies a given message with a known transaction from a source device.

204 CSRMESH_SENSOR_SET_STATE_T Struct Reference

Data Fields

Data	Field
`sensor_type_t`	type Sensor type.The Type field is a 16-bit unsigned integer that determines the sensor type for the associated state.
`CsrUint8`	repeatinterval How often the sensor value is repeated. The RepeatInterval is an 8-bit unsigned integer representing the interval in seconds between repeated transmissions of sensor values by this device.
`CsrUint8`	tid Transaction identifier. The TID, or transaction identifier field is an 8-bit integer that identifies a given message with a known transaction from a source device.

205 CSRMESH_SENSOR_STATE_T Struct Reference

Data Fields

Data	Field
`sensor_type_t`	type Sensor type. The Type field is a 16-bit unsigned integer that determines the sensor type for the associated state.
`CsrUint8`	repeatinterval How often the sensor value is repeated.
`CsrUint8`	tid Transaction identifier. The TID, or transaction identifier field is an 8-bit integer that identifies a given message with a known transaction from a source device.

206 CSRMESH_SENSOR_TYPES_T Struct Reference

Data Fields

Data	Field
`sensor_type_t`	types [8/size_in_bytes(sensor_type_t)] Array of sensor types supported (16 bits per type). The Types field is a variable length sequence of sensor types that the device sending this message supports. The sensor types included in this message shall be ordered by sensor type number and shall always be higher than the FirstType in the corresponding SENSOR_GET_TYPES message. Note: This field can be zero octets in length if a device does not support any sensor types greater than or equal to the FirstType in the corresponding SENSOR_GET_TYPES message.
`CsrUint8`	types_len Length of the types field.
`CsrUint8`	tid Transaction identifier. The TID, or transaction identifier field is an 8-bit integer that identifies a given message with a known transaction from a source device.

207 CSRMESH_SENSOR_VALUE_T Struct Reference

Data Fields

Data	Field
`sensor_type_t`	type Sensor type. The Type field is a 16-bit unsigned integer that determines the sensor type of the value.
`CsrUint8`	value [4/size_in_bytes(CsrUint8)] Sensor value. The Value field is from 1 to 4 octets in length; the format of this field is determined by the Type field.
`CsrUint8`	value_len Length of the value field.
`sensor_type_t`	type2 Sensor type. The Type field is a 16-bit unsigned integer that determines the second sensor type of the value. This field is optional, and can be zero octets in length.
`CsrUint8`	value2 [4/size_in_bytes(CsrUint8)] Sensor value. The Value field is from 1 to 4 octets in length; the format of this field is determined by the Type2 field. This field shall only be included if the Type2 field exists.
`CsrUint8`	value2_len Length of the value2 field.
`CsrUint8`	tid Transaction identifier. The TID, or transaction identifier field is an 8-bit integer that identifies a given message with a known transaction from a source device.

208 CSRMESH_SENSOR_WRITE_VALUE_T Struct Reference

Data Fields

Data	Field
`sensor_type_t`	type Sensor type. The Type field is a 16-bit unsigned integer that determines the sensor type of the value.
`CsrUint8`	value [4/size_in_bytes(CsrUint8)] Sensor Value. The Value field is from 1 to 4 octets in length; the format of this field is determined by the Type field.
`CsrUint8`	value_len Length of the value field.
`sensor_type_t`	type2 Sensor type. The Type field is a 16-bit unsigned integer that determines the second sensor type of the value. This field is optional, and can be zero octets in length.
`CsrUint8`	value2 [4/size_in_bytes(CsrUint8)] Sensor Value. The Value field is from 1 to 4 octets in length; the format of this field is determined by the Type2 field. This field is optional, and can be zero octets in length.
`CsrUint8`	value2_len Length of the value2 field.
`CsrUint8`	tid Transaction identifier. The TID, or transaction identifier field is an 8-bit integer that identifies a given message with a known transaction from a source device.

209 CSRMESH_TIME_BROADCAST_T Struct Reference

Data Fields

Data	Field
CsrUint8	currenttime [6/size_in_bytes(CsrUint8)] Current local time in milliseconds from 1970-01-01 00:00Z.
CsrInt8	timezone Time zone offset of current time from UTC (in 15 minute increments)
CsrUint8	masterclock 1 if sent by clock master, 0 if not

210 CSRMESH_TIME_GET_STATE_T Struct Reference

Data Fields

Data	Field
CsrUint8	tid Transaction ID. The TID or transaction identifier field is an 8-bit integer that identifies a given message with a known transaction from a source device.

211 CSRMESH_TIME_SET_STATE_T Struct Reference

Detailed Description

Setting Time Broadcast Interval: Upon receiving a TIME_SET_STATE message, the device saves the TimeInterval field into the appropriate state value. It then responds with a TIME_STATE message with the current state information.

212 CSRMESH_TIME_STATE_T Struct Reference

Data Fields

Data	Field
CsrUint16	interval Interval between time broadcasts.
CsrUint8	tid Transaction ID. The TID or transaction identifier field is an 8-bit integer that identifies a given message with a known transaction from a source device.

213 CSRMESH_TRACKER_FIND_T Struct Reference

Detailed Description

Finding an Asset: Upon receiving a TRACKER_FIND message, the server checks its tracker cache to see if it has received an ASSET_ANNOUNCE message recently that has the same DeviceID. If it finds one, it will send a TRACKER_FOUND message with the cached information.

214 CSRMESH_TRACKER_FOUND_T Struct Reference

Data Fields

Data	Field
CsrUint16	assetdeviceid Device ID of this asset.
CsrInt8	rssi RSSI of ASSET_ANNOUNCE.
CsrUint8	zone Zone: 0=Immediate 1=Near 2=Distant.
CsrUint16	ageseconds How long since last seen.
CsrUint16	sideeffects Side effects for this asset.
CsrUint8	tid Transaction identifier. The TID, or transaction identifier field is an 8-bit integer that identifies a given message with a known transaction from a source device.

215 CSRMESH_TRACKER_REPORT_T Struct Reference

Data Fields

Data	Field
CsrUint16	assetdeviceid Device ID of this asset.
CsrInt8	rssi RSSI of ASSET_ANNOUNCE.
CsrUint8	zone Zone: 0=Immediate 1=Near 2=Distant.
CsrUint16	ageseconds How long since last seen.
CsrUint16	sideeffects Side effects for this asset.

216 CSRMESH_TRACKER_SET_PROXIMITY_CONFIG_T Struct Reference

Data Fields

Data	Field
CsrInt8	zone0rssithreshold Threshold in dBm for Zone 0 (signed). Default -60.
CsrInt8	zone1rssithreshold Threshold in dBm for Zone 1 (signed). Default -83.
CsrInt8	zone2rssithreshold Threshold in dBm for Zone 2 (signed). Default -100.
CsrUint16	cachedeleteinterval How long until cached entry is deleted (seconds). Default 600.
CsrUint8	delayoffset Offset value for RSSI to delay computation. Default 60.
CsrUint8	delayfactor Factor for RSSI to delay computation. Default 30.
CsrUint16	reportdest Destination ID to which asset reports will be sent. Default 0.

217 CSRMESH_TUNING_ACK_CONFIG_T Struct Reference

Data Fields

Data	Field
CsrUint8	numeratorhightrafficneighratio Default is 14. See tuning doc for explanation.
CsrUint8	numeratornormaltrafficneighratio Default is 12. See tuning doc for explanation.
CsrUint8	denominatortrafficneighratio Default is 10. See tuning doc for explanation.
CsrUint8	normalgoalvalue Default is 60. See tuning doc for explanation.
CsrUint8	highgoalvalue Default is 75. See tuning doc for explanation.
CsrUint8	currentscandutycycle Duty cycle 1..100% or 101..255 (x-100) per mille.
CsrUint8	tid Transaction ID. The TID or transaction identifier field is an 8-bit integer that identifies a given message with a known transaction from a source device.

218 CSRMESH_TUNING_GET_STATS_T Struct Reference

Data Fields

Data	Field
`CsrUint32`	missingreplyparts Bitset of missing reply parts (0 means send all)

219 CSRMESH_TUNING_PROBE_T Struct Reference

Detailed Description

Tuning Probe: The Tuning Probe message is sent to discover neighbours. This messages is issued by devices wanting to determine their density metrics.n The message is sent in two forms. A short form omitting both ScanDutyCycle and BatteryState with a TTL=0. This allows immediate neighbours to perform various calculations and in turn provide their own PROBE messages. The long version is only provided with TTL>0. This cannot be used for immediate neighbour density determination, but can be used to determine the overall network density. The ability to identify if a node is a potential pinch-point in the Mesh network can be achieved through the comparison of immediate and average number of neighbours within the network. The usage of the PROBE message with TTL=0 or TTL>0 is a way to perform these computations. It is worth noting that the periodicity of these two types of messages are different; messages with TTL>0 is much more infrequent than messages with TTL=0. Furthermore, it is wise not to use messages for TTL>0 and embedded values in the determination of the average values. The AverageNumNeighbour field is fixed point 6.2 format encoded. The ScanDutyCycle is expressing percentage for numbers from 1 to 100 and (x-100)/10 percentage for numbers from 101 to 255.

220 CSRMESH_TUNING_SET_CONFIG_T Struct Reference

Data Fields

Data	Field
CsrUint8	numeratorhightrafficneighratio Default is 14. See tuning doc for explanation.
CsrUint8	numeratornormaltrafficneighratio Default is 12. See tuning doc for explanation.
CsrUint8	denominatortrafficneighratio Default is 10. See tuning doc for explanation.
CsrUint8	normalgoalvalue Default is 60. Set to 0 to disable/suspend tuning.
CsrUint8	highgoalvalue Default is 75. Set to 254 to suspend, 255 to disable tuning.
CsrUint8	currentscandutycycle Duty cycle 1..100% or 101..255 (x-100) per mille.
CsrBool	ackrequest Whether ack required.
CsrUint8	tid Transaction ID. The TID or transaction identifier field is an 8-bit integer that identifies a given message with a known transaction from a source device.

221 CSRMESH_TUNING_STATS_T Struct Reference

Data Fields

Data	Field
CsrUint8	partnumber Part number (0..31). Bit 7 means more to come.
CsrUint16	neighbourid1 ID of first neighbour in this message.
CsrUint8	neighbourrate1 Average rate neighbour 1 is seen, 6.2 fixed point format, 0..63.75.
CsrInt8	neighbourrssi1 Average RSSI of neighbour 1.
CsrUint16	neighbourid2 ID of second neighbour in this message.
CsrUint8	neighbourrate2 Average rate neighbour 2 is seen, 6.2 fixed point format, 0..63.75.
CsrInt8	neighbourrssi2 Average RSSI of neighbour 2.

222 CSRMESH_WATCHDOG_INTERVAL_T Struct Reference

Data Fields

Data	Field
`CsrUint16`	interval The current watchdog Interval in seconds that contains the maximum interval that watchdog messages should be sent.
`CsrUint16`	activeaftertime The current watchdog ActiveAfterTime that contains the period of time that a device will listen for responses after sending a watchdog message.
`CsrUint8`	tid Transaction ID. The TID or transaction identifier field is an 8-bit integer that identifies a given message with a known transaction from a source device.

223 CSRMESH_WATCHDOG_MESSAGE_T Struct Reference

Detailed Description

Upon receiving a WATCHDOG_MESSAGE message, if the RspSize field is set to a non-zero value, then the device shall respond with a WATCHDOG_MESSAGE with the RspSize field set to zero, and RspSize -1 octets of additional RandomData.

224 CSRMESH_WATCHDOG_SET_INTERVAL_T Struct Reference

Data Fields

Data	Field
`CsrUint16`	interval The watchdog Interval being set in seconds and indicates the maximum interval that watchdog messages should be sent.
`CsrUint16`	activeaftertime The watchdog ActiveAfterTime being set in seconds that indicates the period of time that a device will listen for responses after sending a watchdog message.
`CsrUint8`	tid Transaction ID. The TID or transaction identifier field is an 8-bit integer that identifies a given message with a known transaction from a source device.

225 nvm_cs_header_t Struct Reference

Data Fields

Data	Field
uint16	len NVM Key identifier.

226 nvm_versioned_header_t Struct Reference

Versioned header information.

227 tuningStats_t Struct Reference

Data Fields

Data	Field
uint16	id ID of audible station. 0 means the entry is not.
uint16	rate time-weighted rate of hearing recently
int16	rssi average RSSI, weighted with time.
uint16	count count up how many heard this epoch
uint16	herProbePeriod Periodic sending period of the station.
int16	herTxPower Tx power of the station.

228 File List

Here is a list of all documented files with brief descriptions:

Name	Description
action_client.h	This files provides the prototypes of the client functions defined in the CSRmesh Action model
action_model.h	Defines CSRmesh Action Model specific data structures
action_server.h	This file provides the prototypes of the server functions defined in the CSRmesh Action model
actuator_client.h	This files provides the prototypes of the client functions defined in the CSRmesh Actuator model
actuator_model.h	Defines CSRmesh Actuator Model specific data structures
actuator_server.h	This file provides the prototypes of the server functions defined in the CSRmesh Actuator model
asset_client.h	This files provides the prototypes of the client functions defined in the CSRmesh Asset model
asset_model.h	Defines CSRmesh Asset Model specific data structures
asset_server.h	This file provides the prototypes of the server functions defined in the CSRmesh Asset model
attention_client.h	This files provides the prototypes of the client functions defined in the CSRmesh Attention model
attention_model.h	Defines CSRmesh Attention Model specific data structures
attention_server.h	This file provides the prototypes of the server functions defined in the CSRmesh Attention model
battery_client.h	This files provides the prototypes of the client functions defined in the CSRmesh Battery model
battery_model.h	Defines CSRmesh Battery Model specific data structures
battery_server.h	This file provides the prototypes of the server functions defined in the CSRmesh Battery model
beacon_client.h	This files provides the prototypes of the client functions defined in the CSRmesh Beacon model

Name	Description
beacon_model.h	Defines CSRmesh Beacon Model specific data structures
beacon_server.h	This file provides the prototypes of the server functions defined in the CSRmesh Beacon model
beaconproxy_client.h	This files provides the prototypes of the client functions defined in the CSRmesh BeaconProxy model
beaconproxy_model.h	Defines CSRmesh BeaconProxy Model specific data structures
beaconproxy_server.h	This file provides the prototypes of the server functions defined in the CSRmesh BeaconProxy model
bearer_client.h	This files provides the prototypes of the client functions defined in the CSRmesh Bearer model
bearer_model.h	Defines CSRmesh Bearer Model specific data structures
bearer_server.h	This file provides the prototypes of the server functions defined in the CSRmesh Bearer model
config_client.h	This files provides the prototypes of the client functions defined in the CSRmesh Config model
config_model.h	Defines CSRmesh Config Model specific data structures
config_server.h	This file provides the prototypes of the server functions defined in the CSRmesh Config model
csr_macro.h	Defines and functions for csr_macro.h
csr_mesh.h	CSRmesh library configuration and control functions
csr_mesh_model_common.h	This files defines data types and constants that are commonly used across models
csr_mesh_result.h	CSRmesh library result type
csr_mesh_types.h	CSRmesh library data types
csr_sched.h	CSRmesh library LE bearer scheduling configuration and control functions
csr_sched_types.h	CSRmesh Scheduler data types
csr_types.h	CSRmesh library data types
data_client.h	This files provides the prototypes of the client functions defined in the CSRmesh Data model
data_model.h	Defines CSRmesh Data Model specific data structures

Name	Description
data_server.h	This file provides the prototypes of the server functions defined in the CSRmesh Data model
diagnostic_client.h	This files provides the prototypes of the client functions defined in the CSRmesh Diagnostic model
diagnostic_model.h	Defines CSRmesh Diagnostic Model specific data structures
diagnostic_server.h	This file provides the prototypes of the server functions defined in the CSRmesh Diagnostic model
extension_client.h	This files provides the prototypes of the client functions defined in the CSRmesh Extension model
extension_model.h	Defines CSRmesh Extension Model specific data structures
extension_server.h	This file provides the prototypes of the server functions defined in the CSRmesh Extension model
firmware_client.h	This files provides the prototypes of the client functions defined in the CSRmesh Firmware model
firmware_model.h	Defines CSRmesh Firmware Model specific data structures
firmware_server.h	This file provides the prototypes of the server functions defined in the CSRmesh Firmware model
group_client.h	This files provides the prototypes of the client functions defined in the CSRmesh Group model
group_model.h	Defines CSRmesh Group Model specific data structures
group_server.h	This file provides the prototypes of the server functions defined in the CSRmesh Group model
largeobjecttransfer_client.h	This files provides the prototypes of the client functions defined in the CSRmesh LargeObjectTransfer model
largeobjecttransfer_model.h	Defines CSRmesh LargeObjectTransfer Model specific data structures
largeobjecttransfer_server.h	This file provides the prototypes of the server functions defined in the CSRmesh LargeObjectTransfer model
light_client.h	This files provides the prototypes of the client functions defined in the CSRmesh Light model
light_model.h	Defines CSRmesh Light Model specific data structures
light_server.h	This file provides the prototypes of the server functions defined in the CSRmesh Light model
nvm_access.h	Header definitions for NVM usage
ping_client.h	This files provides the prototypes of the client functions defined in the CSRmesh Ping model

Name	Description
ping_model.h	Defines CSRmesh Ping Model specific data structures
ping_server.h	This file provides the prototypes of the server functions defined in the CSRmesh Ping model
power_client.h	This files provides the prototypes of the client functions defined in the CSRmesh Power model
power_model.h	Defines CSRmesh Power Model specific data structures
power_server.h	This file provides the prototypes of the server functions defined in the CSRmesh Power model
sensor_client.h	This files provides the prototypes of the client functions defined in the CSRmesh Sensor model
sensor_model.h	Defines CSRmesh Sensor Model specific data structures
sensor_server.h	This file provides the prototypes of the server functions defined in the CSRmesh Sensor model
sensor_types.h	Defines the types used in the CSRmesh Sensor and actuator Model
Switch_client.h	This files provides the prototypes of the client functions defined in the CSRmesh Switch model
switch_model.h	Defines CSRmesh Switch Model specific data structures
switch_server.h	This file provides the prototypes of the server functions defined in the CSRmesh Switch model
time_client.h	This files provides the prototypes of the client functions defined in the CSRmesh Time model
time_model.h	Defines CSRmesh Time Model specific data structures
time_server.h	This file provides the prototypes of the server functions defined in the CSRmesh Time model
tracker_client.h	This files provides the prototypes of the client functions defined in the CSRmesh Tracker model
tracker_model.h	Defines CSRmesh Tracker Model specific data structures
tracker_server.h	This file provides the prototypes of the server functions defined in the CSRmesh Tracker model
tuning_client.h	This files provides the prototypes of the client functions defined in the CSRmesh Tuning model
tuning_model.h	Defines CSRmesh Tuning Model specific data structures
tuning_server.h	This file provides the prototypes of the server functions defined in the CSRmesh Tuning model
watchdog_client.h	This files provides the prototypes of the client functions defined in the CSRmesh Watchdog model

Name	Description
watchdog_model.h	Defines CSRmesh Watchdog Model specific data structures
watchdog_server.h	This file provides the prototypes of the server functions defined in the CSRmesh Watchdog model

229 action_client.h File Reference

Detailed Description

This files provides the prototypes of the client functions defined in the CSRmesh Action model.

229.1 Functions of action_client.h File Reference

```
CSRmeshResult ActionModelClientInit (CSRMESH_MODEL_CALLBACK_T app_callback)
```

Initialises Action Model Client functionality.

```
CSRmeshResult ActionClientPrepareAction (void)
```

To set an action on a node the following functions should be called in sequence. ActionClientPrepareAction() followed by the model API whose action need to be set followed by ActionClientSetAction(). ActionClientPrepareAction() function indicates the action model and the mesh stack that next model message being prepared is for action set message and would not be transmitted over the air.

```
CSRmeshResult ActionClientSetAction (CsrUint8 nw_id, CsrUint16 dest_id,
CsrUint8 ttl, CSRMESH_ACTION_SET_ACTION_INFO_T *p_params)
```

To set an action on a node the following functions should be called in sequence. ActionClientPrepareAction() followed by the model API whose action need to be set followed by ActionClientSetAction(). ActionClientSetAction() function forms the action set msgs with the action characteristics and the model message API previously called and sends it over the network onto the node with dest_id address. This is a multiple part message, where each part will be acknowledged through an ACTION_SET_ACK (0x51).

```
CSRmeshResult ActionGetActionStatus (CsrUint8 nw_id, CsrUint16 dest_id,
CsrUint8 ttl, CSRMESH_ACTION_GET_ACTION_STATUS_T *p_params)
```

Request towards a node about the various Actions currently handled. This will be answered through an ACTION_STATUS. A Transaction ID is provided to ensure identification of response with request.

```
CSRmeshResult ActionDelete (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8 ttl,
CSRMESH_ACTION_DELETE_T *p_params)
```

This message allows a set of actions to be removed. Actions are identified through a bitmask reflecting the ActionID one wishes to delete. This message will be acknowledged through ACTION_DELETE_ACK.

```
CSRmeshResult ActionGet (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8 ttl,
CSRMESH_ACTION_GET_T *p_params)
```

Using this message, it is possible to interrogate a node about the specific payload associated with an Action ID. This will be answered by and ACTION_SET_ACTION message, ACTION_SET_ACTION_ACK will need to be issued in order to collect all parts.

230 action_model.h File Reference

Detailed Description

Defines CSRmesh Action Model specific data structures .

230.1 Data Structures of action_model.h File Reference

`struct CSRMESH_ACTION_SET_ACTION_INFO_T`

CSRmesh Action Model message types.

`struct CSRMESH_ACTION_SET_ACTION_INFO_EXT_T`

This message defines the action to be sent when GET_ACTION is received. This is a multiple parts message, each part will be acknowledged through an ACTION_SET_ACK (0x51).This also contains the buffer to send the action message through it.

`struct CSRMESH_ACTION_SET_ACTION_T`

This message defines the action which needs to be taken. This is a multiple parts message, each part will be acknowledged through an ACTION_SET_ACK (0x51).

`struct CSRMESH_ACTION_SET_ACTION_ACK_T`

Every parts received is acknowledged, providing both Transaction ID and ActionID (which includes Part Number)

`struct CSRMESH_ACTION_GET_ACTION_STATUS_T`

Request towards a node about the various Actions currently handled. This will be answered through an ACTION_STATUS. A Transaction ID is provided to ensure identification of response with request.

`struct CSRMESH_ACTION_ACTION_STATUS_T`

This provides a bitmask identifying all the currently allocated ActionID. The Action Supported field provides an indication on how many actions this device has capacity for. Note that a device will never have more than 32 actions.

`struct CSRMESH_ACTION_DELETE_T`

This message allows a set of actions to be removed. Actions are identified through a bitmask reflecting the ActionID one wishes to delete. This message will be acknowledged through ACTION_DELETE_ACK.

`struct CSRMESH_ACTION_DELETE_ACK_T`

This message confirms the delete commands and report which actions have been removed. The provided transaction ID will match the one supplied by ACTION_DELETE.

`struct CSRMESH_ACTION_GET_T`

Using this message, it is possible to interrogate a node about the specific payload associated with an Action ID. This will be answered by and ACTION_SET_ACTION message, ACTION_SET_ACTION_ACK will need to be issued in order to collect all parts.

231 action_server.h File Reference

Detailed Description

This file provides the prototypes of the server functions defined in the CSRmesh Action model.

231.1 Functions of action_server.h File Reference

```
CSRmeshResult ActionModelInit (CsrUint8 nw_id, CsrUint16 *group_id_list,
CsrUint16 num_groups, CSRMESH_MODEL_CALLBACK_T app_callback)
```

Model Initialisation.

```
CSRmeshResult ActionPrepareAction (void)
```

To set an action on a node the following functions should be called in sequence. ActionPrepareAction() followed by the model API whose action need to be set followed by ActionSetAction(). ActionPrepareAction function indicates the action model and the mesh stack that next model message being prepared is for action set message and would not be transmitted over the air.

```
CSRmeshResult ActionSetAction (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8 ttl,
CSRMESH_ACTION_SET_ACTION_INFO_T *p_params)
```

To set an action on a node the following functions should be called in sequence. ActionPrepareAction() followed by the model API whose action need to be set followed by ActionSetAction(). ActionSetAction() function forms the action set msgs with the action characteristics and the model message API previously called and sends it over the network onto the node with dest_id address. This is a multiple part message, where each part will be acknowledged through an ACTION_SET_ACK (0x51).

```
CSRmeshResult ActionSetActionAck (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8
ttl, CSRMESH_ACTION_SET_ACTION_ACK_T *p_params)
```

Every parts received is acknowledged, providing both Transaction ID and ActionID (which includes Part Number)

```
CSRmeshResult ActionActionStatus (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8
ttl, CSRMESH_ACTION_ACTION_STATUS_T *p_params)
```

This provides a bitmask identifying all the currently allocated ActionID. The Action Supported field provides an indication on how many actions this device has capacity for. Note that a device will never have more than 32 actions.

```
CSRmeshResult ActionDeleteAck (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8 ttl,
CSRMESH_ACTION_DELETE_ACK_T *p_params)
```

This message confirms the delete commands and report which actions have been removed. The provided transaction ID will match the one supplied by ACTION_DELETE.

```
CSRmeshResult ActionSendMessage (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8
ttl, CsrUint8 *msg, CsrUint8 len)
```

Sends CSRmesh message that was received as an action.

232 actuator_client.h File Reference

Detailed Description

This files provides the prototypes of the client functions defined in the CSRmesh Actuator model.

232.1 Functions of actuator_client.h File Reference

```
CSRmeshResult ActuatorModelClientInit (CSRMESH_MODEL_CALLBACK_T app_callback)
```

Initialises Actuator Model Client functionality.

```
CSRmeshResult ActuatorGetTypes (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8
ttl, CSRMESH_ACTUATOR_GET_TYPES_T *p_params)
```

Upon receiving an ACTUATOR_GET_TYPES message, the device responds with an ACTUATOR_TYPES message with a list of supported types greater than or equal to the FirstType field. If the device does not support any types greater than or equal to FirstType, it sends an ACTUATOR_TYPES message with a zero length Types field.

```
CSRmeshResult ActuatorSetValue (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8
ttl, CSRMESH_ACTUATOR_SET_VALUE_T *p_params, bool request_ack)
```

Get sensor state. Upon receiving an ACTUATOR_SET_VALUE_NO_ACK message, where the destination address is the device ID of this device and the Type field is a supported actuator type, the device shall immediately use the Value field for the given Type field. The meaning of this actuator value is not defined in this specification. Upon receiving an ACTUATOR_SET_VALUE_NO_ACK message, where the destination address is is the device ID of this device but the Type field is not a supported actuator type, the device shall ignore the message.

```
CSRmeshResult ActuatorGetValueAck (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8
ttl, CSRMESH_ACTUATOR_GET_VALUE_ACK_T *p_params)
```

Get Current sensor state.

233 actuator_model.h File Reference

Detailed Description

Defines CSRmesh Actuator Model specific data structures .

233.1 Data Structures of actuator_model.h File Reference

`struct CSRMESH_ACTUATOR_GET_TYPES_T`

CSRmesh Actuator Model message types.

`struct CSRMESH_ACTUATOR_TYPES_T`

Actuator types.

`struct CSRMESH_ACTUATOR_SET_VALUE_T`

Get sensor state. Upon receiving an ACTUATOR_SET_VALUE_NO_ACK message, where the destination address is the device ID of this device and the Type field is a supported actuator type, the device shall immediately use the Value field for the given Type field. The meaning of this actuator value is not defined in this specification. Upon receiving an ACTUATOR_SET_VALUE_NO_ACK message, where the destination address is is the device ID of this device but the Type field is not a supported actuator type, the device shall ignore the message.

`struct CSRMESH_ACTUATOR_VALUE_ACK_T`

Current sensor state.

`struct CSRMESH_ACTUATOR_GET_VALUE_ACK_T`

Get Current sensor state.

234 actuator_server.h File Reference

Detailed Description

This file provides the prototypes of the server functions defined in the CSRmesh Actuator model.

234.1 Functions of actuator_server.h File Reference

```
CSRmeshResult ActuatorModelInit (CsrUint8 nw_id, CsrUint16 *group_id_list,
CsrUint16 num_groups, CSRMESH_MODEL_CALLBACK_T app_callback)
```

Model Initialisation.

```
CSRmeshResult ActuatorTypes (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8 ttl,
CSRMESH_ACTUATOR_TYPES_T *p_params)
```

Actuator types.

```
CSRmeshResult ActuatorValueAck (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8
ttl, CSRMESH_ACTUATOR_VALUE_ACK_T *p_params)
```

Current sensor state.

235 asset_client.h File Reference

Detailed Description

This files provides the prototypes of the client functions defined in the CSRmesh Asset model.

235.1 Functions of asset_client.h File Reference

```
CSRmeshResult AssetModelClientInit (CSRMESH_MODEL_CALLBACK_T app_callback)
```

Initialises Asset Model Client functionality.

```
CSRmeshResult AssetSetState (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8 ttl,
CSRMESH_ASSET_SET_STATE_T *p_params)
```

Setting Asset State: Upon receiving an ASSET_SET_STATE message, the device saves the Interval, SideEffects, ToDestination, TxPower, Number of Announcements and AnnounceInterval fields into the appropriate state values. It then responds with an ASSET_STATE message with the current state information.

```
CSRmeshResult AssetGetState (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8 ttl,
CSRMESH_ASSET_GET_STATE_T *p_params)
```

Getting Asset State: Upon receiving an ASSET_GET_STATE message, the device responds with an ASSET_STATE message with the current state information.

236 asset_model.h File Reference

Detailed Description

Defines CSRmesh Asset Model specific data structures .

236.1 Data Structures of asset_model.h File Reference

`struct CSRMESH_ASSET_SET_STATE_T`

CSRmesh Asset Model message types.

`struct CSRMESH_ASSET_GET_STATE_T`

Getting Asset State: Upon receiving an ASSET_GET_STATE message, the device responds with an ASSET_STATE message with the current state information.

`struct CSRMESH_ASSET_STATE_T`

Current asset state.

`struct CSRMESH_ASSET_ANNOUNCE_T`

Asset announcement.

237 asset_server.h File Reference

Detailed Description

This file provides the prototypes of the server functions defined in the CSRmesh Asset model.

237.1 Functions of asset_server.h File Reference

```
CSRmeshResult AssetModelInit (CsrUint8 nw_id, CsrUint16 *group_id_list,
CsrUint16 num_groups, CSRMESH_MODEL_CALLBACK_T app_callback)
```

Model Initialisation.

```
CSRmeshResult AssetState (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8 ttl,
CSRMESH_ASSET_STATE_T *p_params)
```

Current asset state.

```
CSRmeshResult AssetAnnounce (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8 ttl,
CSRMESH_ASSET_ANNOUNCE_T *p_params)
```

Asset announcement.

238 attention_client.h File Reference

Detailed Description

This files provides the prototypes of the client functions defined in the CSRmesh Attention model.

238.1 Functions of attention_client.h File Reference

```
CSRmeshResult AttentionModelClientInit (CSRMESH_MODEL_CALLBACK_T app_callback)
```

Initialises Attention Model Client functionality.

```
CSRmeshResult AttentionSetState (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8
ttl, CSRMESH_ATTENTION_SET_STATE_T *p_params)
```

Setting Flashing State: Upon receiving an ATTENTION_SET_STATE message, the device saves the AttractAttention and AttentionDuration fields into the appropriate state value. It then responds with an ATTENTION_STATE message with the current state information. If the AttractAttention field is set to 0x01 and the AttentionDuration is not 0xFFFF, then any existing attention timer is cancelled and a new attention timer is started that will expire after AttentionDuration milliseconds. If the AttractAttention field is set to 0x01 and the AttentionDuration field is 0xFFFF, then the attention timer is ignored. If the AttractAttention field is set to 0x00, then the attention timer is cancelled if it is already running.

239 attention_model.h File Reference

Detailed Description

Defines CSRmesh Attention Model specific data structures .

239.1 Data Structures of attention_model.h File Reference

`struct CSRMESH_ATTENTION_SET_STATE_T`

CSRmesh Attention Model message types.

`struct CSRMESH_ATTENTION_STATE_T`

Current battery state.

240 attention_server.h File Reference

Detailed Description

This file provides the prototypes of the server functions defined in the CSRmesh Attention model.

240.1 Functions of attention_server.h File Reference

```
CSRmeshResult AttentionModelInit (CsrUint8 nw_id, CsrUint16 *group_id_list,
CsrUint16 num_groups, CSRMESH_MODEL_CALLBACK_T app_callback)
```

Model Initialisation.

```
CSRmeshResult AttentionState (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8 ttl,
CSRMESH_ATTENTION_STATE_T *p_params)
```

Current battery state.

241 battery_client.h File Reference

Detailed Description

This files provides the prototypes of the client functions defined in the CSRmesh Battery model.

241.1 Functions of battery_client.h File Reference

```
CSRmeshResult BatteryModelClientInit (CSRMESH_MODEL_CALLBACK_T app_callback)
```

Initialises Battery Model Client functionality.

```
CSRmeshResult BatteryGetState (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8 ttl,
CSRMESH_BATTERY_GET_STATE_T *p_params)
```

Getting Battery State: Upon receiving a BATTERY_GET_STATE message, the device responds with a BATTERY_STATE message with the current state information.

242 battery_model.h File Reference

Detailed Description

Defines CSRmesh Battery Model specific data structures .

242.1 Data Structures of battery_model.h File Reference

`struct CSRMESH_BATTERY_GET_STATE_T`

CSRmesh Battery Model message types.

`struct CSRMESH_BATTERY_STATE_T`

Current battery state.

243 battery_server.h File Reference

Detailed Description

This file provides the prototypes of the server functions defined in the CSRmesh Battery model.

243.1 Functions of battery_server.h File Reference

```
CSRmeshResult BatteryModelInit (CsrUint8 nw_id, CsrUint16 *group_id_list,
CsrUint16 num_groups, CSRMESH_MODEL_CALLBACK_T app_callback)
```

Model Initialisation.

```
CSRmeshResult BatteryState (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8 ttl,
CSRMESH_BATTERY_STATE_T *p_params)
```

Current battery state.

244 beacon_client.h File Reference

Detailed Description

This files provides the prototypes of the client functions defined in the CSRmesh Beacon model.

244.1 Functions of beacon_client.h File Reference

```
CSRmeshResult BeaconModelClientInit (CSRMESH_MODEL_CALLBACK_T app_callback)
```

Initialises Beacon Model Client functionality.

```
CSRmeshResult BeaconSetStatus (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8 ttl,
CSRMESH_BEACON_SET_STATUS_T *p_params)
```

May be sent to a beacon to set its status. More than one such command can be sent to set the intervals for different beacon types, which need not be the same. This message also allows for the wakeup intervals to be set if the beacon elects to only be on the mesh sporadically. The time is always with respect to the beacon's internal clock and has no absolute meaning. This message will be answered through BEACON_STATUS.

```
CSRmeshResult BeaconGetBeaconStatus (CsrUint8 nw_id, CsrUint16 dest_id,
CsrUint8 ttl, CSRMESH_BEACON_GET_BEACON_STATUS_T *p_params)
```

Message to be sent to a beacon in order to recover its current status. This message fetch information pertinent to a particular type. The transaction ID will help matching the pertinent BEACON_STATUS message: a different transaction ID shall be used for a different type (although the BEACON_STATUS message has the type pertinent to the status and thus could be used in conjunction to the Transaction ID for disambiguation).

```
CSRmeshResult BeaconGetTypes (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8 ttl,
CSRMESH_BEACON_GET_TYPES_T *p_params)
```

Message allowing a node to fetch all supported types and associated information by a given Beacon.

```
CSRmeshResult BeaconClientSetPayload (CsrUint8 nw_id, CsrUint16 dest_id,
CsrUint8 ttl, CSRMESH_BEACON_SET_PAYLOAD_T *p_params)
```

One or more of these packets may be sent to a beacon to set its payload. The content is either the raw advert data, or extra information (such as crypto cycle) which a beacon may require. The payload data is sent as a length and an offset, so the whole payload need not be sent if it has not changed. A beacon can support many beacon types - it can be sent as many different payloads as needed, one for each type. The first byte of the first payload packet contains the length and offset of the payload and the payload ID; this allows a beacon which already has the payload to send an immediate acknowledgement, saving traffic. This ID will be sent back in the 'Payload ACK' message if the beacon has received the whole payload. The payload may have to be split in several parts, in which case only the last part shall be acknowledged. Upon receiving a BEACON_SET_PAYLOAD, the beacon will update its corresponding beacon data: if data was previously available, it will be replaced by the provided payload, thus a beacon can only hold one payload per beacon type.

```
CSRmeshResult BeaconClientPayloadAck (CsrUint8 nw_id, CsrUint16 dest_id,
CsrUint8 ttl, CSRMESH_BEACON_PAYLOAD_ACK_T *p_params)
```

Only one Acknowledgement message will occur for multiple BEACON_SET_PAYLOAD messages sharing the same transaction ID. Only when the last segment is received that such acknowledgement will be issued. Where missing payload messages exist, the list of their indices will be provided in the Ack field.

```
CSRmeshResult BeaconGetPayload (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8
ttl, CSRMESH_BEACON_GET_PAYLOAD_T *p_params)
```

Message allowing a node to retrieve the current payload held on a given beacon. This message shall be answered by one or more BEACON_SET_PAYLOAD messages.

245 beacon_model.h File Reference

Detailed Description

Defines CSRmesh Beacon Model specific data structures .

245.1 Data Structures of beacon_model.h File Reference

`struct CSRMESH_BEACON_SET_STATUS_T`

CSRmesh Beacon Model message types.

`struct CSRMESH_BEACON_GET_BEACON_STATUS_T`

Message to be sent to a beacon in order to recover its current status. This message fetch information pertinent to a particular type. The transaction ID will help matching the pertinent BEACON_STATUS message: a different transaction ID shall be used for a different type (although the BEACON_STATUS message has the type pertinent to the status and thus could be used in conjunction to the Transaction ID for disambiguation).

`struct CSRMESH_BEACON_BEACON_STATUS_T`

This message issue issued by a Beacon as response to BEACON_SET_STATUS or BEACON_GET_STATUS. Furthermore, a Beacon appearing on the mesh sporadically, will issue such message (with destination ID set to 0) as soon as it re-joins the mesh. In this case, one message per active beacon type should be issued. This message will provide the Payload ID currently in used for the associated type.

`struct CSRMESH_BEACON_GET_TYPES_T`

Message allowing a node to fetch all supported types and associated information by a given Beacon.

`struct CSRMESH_BEACON_TYPES_T`

Provides information on the set of beacon supported by the node. The battery level is reported as well as time since last message.

`struct CSRMESH_BEACON_SET_PAYLOAD_T`

One or more of these packets may be sent to a beacon to set its payload. The content is either the raw advert data, or extra information (such as crypto cycle) which a beacon may require. The payload data is sent as a length and an offset, so the whole payload need not be sent if it has not changed. A beacon can support many beacon types - it can be sent as many different payloads as needed, one for each type. The first byte of the first payload packet contains the length and offset of the payload and the payload ID; this allows a beacon which already has the payload to send an immediate acknowledgement, saving traffic. This ID will be sent back in the 'Payload ACK' message if the beacon has received the whole payload. The payload may have to be split in several parts, in which case only the last part shall be acknowledged. Upon receiving a BEACON_SET_PAYLOAD, the beacon will update its corresponding beacon data: if data was previously available, it will be replaced by the provided payload, thus a beacon can only hold one payload per beacon type.

`struct CSRMESH_BEACON_PAYLOAD_ACK_T`

Only one Acknowledgement message will occur for multiple BEACON_SET_PAYLOAD messages sharing the same transaction ID. Only when the last segment is received that such acknowledgement will be issued. Where missing payload messages exist, the list of their indices will be provided in the Ack field.

`struct CSRMESH_BEACON_GET_PAYLOAD_T`

Message allowing a node to retrieve the current payload held on a given beacon. This message shall be answered by one or more BEACON_SET_PAYLOAD messages.

246 beacon_server.h File Reference

Detailed Description

This file provides the prototypes of the server functions defined in the CSRmesh Beacon model.

246.1 Functions of beacon_server.h File Reference

```
CSRmeshResult BeaconModelInit (CsrUint8 nw_id, CsrUint16 *group_id_list,
CsrUint16 num_groups, CSRMESH_MODEL_CALLBACK_T app_callback)
```

Model Initialisation.

```
CSRmeshResult BeaconBeaconStatus (CsrUint8 nw_id, CsrUint16 src_id, CsrUint16
dest_id, CsrUint8 ttl, CSRMESH_BEACON_BEACON_STATUS_T *p_params)
```

This message issue issued by a Beacon as response to BEACON_SET_STATUS or BEACON_GET_STATUS. Furthermore, a Beacon appearing on the mesh sporadically, will issue such message (with destination ID set to 0) as soon as it re-joins the mesh. In this case, one message per active beacon type should be issued. This message will provide the Payload ID currently in used for the associated type.

```
CSRmeshResult BeaconTypes (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8 ttl,
CSRMESH_BEACON_TYPES_T *p_params)
```

Provides information on the set of beacon supported by the node. The battery level is reported as well as time since last message.

```
CSRmeshResult BeaconSetPayload (CsrUint8 nw_id, CsrUint16 src_id, CsrUint16
dest_id, CsrUint8 ttl, CSRMESH_BEACON_SET_PAYLOAD_T *p_params)
```

One or more of these packets may be sent to a beacon to set its payload. The content is either the raw advert data, or extra information (such as crypto cycle) which a beacon may require. The payload data is sent as a length and an offset, so the whole payload need not be sent if it has not changed. A beacon can support many beacon types - it can be sent as many different payloads as needed, one for each type. The first byte of the first payload packet contains the length and offset of the payload and the payload ID; this allows a beacon which already has the payload to send an immediate acknowledgement, saving traffic. This ID will be sent back in the 'Payload ACK' message if the beacon has received the whole payload. The payload may have to be split in several parts, in which case only the last part shall be acknowledged. Upon receiving a BEACON_SET_PAYLOAD, the beacon will update its corresponding beacon data: if data was previously available, it will be replaced by the provided payload, thus a beacon can only hold one payload per beacon type.

```
CSRmeshResult BeaconPayloadAck (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8
ttl, CSRMESH_BEACON_PAYLOAD_ACK_T *p_params)
```

Only one Acknowledgement message will occur for multiple BEACON_SET_PAYLOAD messages sharing the same transaction ID. Only when the last segment is received that such acknowledgement will be issued. Where missing payload messages exist, the list of their indices will be provided in the Ack field.

247 beaconproxy_client.h File Reference

Detailed Description

This files provides the prototypes of the client functions defined in the CSRmesh BeaconProxy model.

247.1 Functions of beaconproxy_client.h File Reference

```
CSRmeshResult BeaconProxyModelClientInit (CSRMESH_MODEL_CALLBACK_T app_callback)
```

Initialises BeaconProxy Model Client functionality.

```
CSRmeshResult BeaconProxyAdd (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8 ttl,
CSRMESH_BEACONPROXY_ADD_T *p_params)
```

Message can be sent to a beacon-proxy to add to the list of devices it manages. This request will be acknowledged using a Proxy Command Status Devices. Up to four device ID can be specified in this message: Group can be defined. This message also permits the flushing of pending messages for managed IDs.

```
CSRmeshResult BeaconProxyRemove (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8
ttl, CSRMESH_BEACONPROXY_REMOVE_T *p_params)
```

May be sent to a proxy to remove from the list of devices it manages. Any message queued for those devices will be cleared. This request will be acknowledged using a Proxy Command Status Devices. Groups can be used, implying that all its members will be removed from the proxy management. Specifying 0 in the Device Addresses will be interpreted as stopping all Proxy activities on the targeted proxy.

```
CSRmeshResult BeaconProxyGetStatus (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8
ttl)
```

Fetch the current status - this will be answered by a BEACON_PROXY_STATUS.

```
CSRmeshResult BeaconProxyGetDevices (CsrUint8 nw_id, CsrUint16 dest_id,
CsrUint8 ttl, CSRMESH_BEACONPROXY_GET_DEVICES_T *p_params)
```

Fetch the current set of devices ID managed by a given proxy. This will be answered by one or more BEACON_PROXY_DEVICES messages.

248 beaconproxy_model.h File Reference

Detailed Description

Defines CSRmesh BeaconProxy Model specific data structures .

248.1 Data Structures of beaconproxy_model.h File Reference

`struct CSRMESH_BEACONPROXY_ADD_T`

CSRmesh BeaconProxy Model message types.

`struct CSRMESH_BEACONPROXY_REMOVE_T`

May be sent to a proxy to remove from the list of devices it manages. Any message queued for those devices will be cleared. This request will be acknowledged using a Proxy Command Status Devices. Groups can be used, implying that all its members will be removed from the proxy management. Specifying 0 in the Device Addresses will be interpreted as stopping all Proxy activities on the targeted proxy.

`struct CSRMESH_BEACONPROXY_COMMAND_STATUS_DEVICES_T`

Generic acknowledgement - the transaction ID permits reconciliation with sender.

`struct CSRMESH_BEACONPROXY_PROXY_STATUS_T`

Provides generic information on the internal states of a Proxy. Including number of managed nodes, groups, states of the various queues.

`struct CSRMESH_BEACONPROXY_GET_DEVICES_T`

Fetch the current set of devices ID managed by a given proxy. This will be answered by one or more BEACON_PROXY_DEVICES messages.

`struct CSRMESH_BEACONPROXY_DEVICES_T`

Provide the list of Devices and Groups currently managed by a given Proxy. This will be a multiple parts message.

249 beaconproxy_server.h File Reference

Detailed Description

This file provides the prototypes of the server functions defined in the CSRmesh BeaconProxy model.

249.1 Functions of beaconproxy_server.h File Reference

```
CSRmeshResult BeaconProxyModelInit (CsrUint8 nw_id, CsrUint16 *group_id_list,
CsrUint16 num_groups, CSRMESH_MODEL_CALLBACK_T app_callback)
```

Model Initialisation.

```
CSRmeshResult BeaconProxySetupBeaconList (CsrUint16 *beacon_list, CsrUint8
max_groups, CsrUint8 max_devices)
```

Sets up a list to manage added beacons.

```
CSRmeshResult BeaconProxyCommandStatusDevices (CsrUint8 nw_id, CsrUint16
dest_id, CsrUint8 ttl, CSRMESH_BEACONPROXY_COMMAND_STATUS_DEVICES_T *p_params)
```

Generic acknowledgement - the transaction ID permits reconciliation with sender.

```
CSRmeshResult BeaconProxyProxyStatus (CsrUint8 nw_id, CsrUint16 dest_id,
CsrUint8 ttl, CSRMESH_BEACONPROXY_PROXY_STATUS_T *p_params)
```

Provides generic information on the internal states of a Proxy. Including number of managed nodes, groups, states of the various queues.

```
CSRmeshResult BeaconProxyDevices (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8
ttl, CSRMESH_BEACONPROXY_DEVICES_T *p_params)
```

Provide the list of Devices and Groups currently managed by a given Proxy. This will be a multiple parts message.

250 bearer_client.h File Reference

Detailed Description

This files provides the prototypes of the client functions defined in the CSRmesh Bearer model.

250.1 Functions of bearer_client.h File Reference

```
CSRmeshResult BearerModelClientInit (CSRMESH_MODEL_CALLBACK_T app_callback)
```

Initialises Bearer Model Client functionality.

```
CSRmeshResult BearerSetState (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8 ttl,
CSRMESH_BEARER_SET_STATE_T *p_params)
```

Setting Bearer State: Upon receiving a BEARER_SET_STATE message, where the destination address is the device ID of this device, the device saves the BearerRelayActive, BearerEnabled, and BearerPromiscuous fields into the appropriate state value. Then the device responds with a BEARER_STATE message with the current state information.

```
CSRmeshResult BearerGetState (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8 ttl,
CSRMESH_BEARER_GET_STATE_T *p_params)
```

Getting Bearer State: Upon receiving a BEARER_GET_STATE message, where the destination address is the device ID of this device, the device responds with a BEARER_STATE message with the current state information.

251 bearer_model.h File Reference

Detailed Description

Defines CSRmesh Bearer Model specific data structures .

251.1 Data Structures of bearer_model.h File Reference

`struct CSRMESH_BEARER_SET_STATE_T`

CSRmesh Bearer Model message types.

`struct CSRMESH_BEARER_GET_STATE_T`

Getting Bearer State: Upon receiving a BEARER_GET_STATE message, where the destination address is the device ID of this device, the device responds with a BEARER_STATE message with the current state information.

`struct CSRMESH_BEARER_STATE_T`

Set bearer state.

252 bearer_server.h File Reference

Detailed Description

This file provides the prototypes of the server functions defined in the CSRmesh Bearer model.

252.1 Functions of bearer_server.h File Reference

```
CSRmeshResult BearerModelInit (CsrUint8 nw_id, CsrUint16 *group_id_list,
CsrUint16 num_groups, CSRMESH_MODEL_CALLBACK_T app_callback)
```

Model Initialisation.

```
CSRmeshResult BearerState (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8 ttl,
CSRMESH_BEARER_STATE_T *p_params)
```

Set bearer state.

253 config_client.h File Reference

Detailed Description

This files provides the prototypes of the client functions defined in the CSRmesh Config model.

253.1 Functions of config_client.h File Reference

```
CSRmeshResult ConfigModelClientInit (CSRMESH_MODEL_CALLBACK_T app_callback)
```

Initialises Config Model Client functionality.

```
CSRmeshResult ConfigLastSequenceNumber (CsrUint8 nw_id, CsrUint16 dest_id,
CsrUint8 ttl, CSRMESH_CONFIG_LAST_SEQUENCE_NUMBER_T *p_params)
```

Upon receiving a CONFIG_LAST_SEQUENCE_NUMBER message from a trusted device, the local device updates the SequenceNumber to at least one higher than the LastSequenceNumber in the message. Note: A trusted device means a device that is not only on the same CSRmesh network, having the same network key, but also interacted with in the past. This message is most useful to check if a device has been reset, for example when the batteries of the device are changed, but it does not remember its last sequence number in non-volatile memory.

```
CSRmeshResult ConfigResetDevice (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8
ttl, CSRMESH_CONFIG_RESET_DEVICE_T *p_params)
```

Upon receiving a CONFIG_RESET_DEVICE message from a trusted device directed at only this device, the local device sets the DeviceID to zero, and forgets all network keys, associated NetworkIVs and other configuration information. The device may act as if it is not associated and use MASP to re-associate with a network. Note: If the CONFIG_RESET_DEVICE message is received on any other destination address than the DeviceID of the local device, it is ignored. This is typically used when selling a device, to remove the device from the network of the seller so that the purchaser can associate the device with their network.

```
CSRmeshResult ConfigSetDeviceIdentifier (CsrUint8 nw_id, CsrUint16 dest_id,
CsrUint8 ttl, CSRMESH_CONFIG_SET_DEVICE_IDENTIFIER_T *p_params)
```

When the device with a DeviceID of 0x0000 receives a CONFIG_SET_DEVICE_IDENTIFIER message and the DeviceHash of the message matches the DeviceHash of this device, the DeviceID of this device is set to the DeviceID field of this message. Then the device responds with the DEVICE_CONFIG_IDENTIFIER message using the new DeviceID as the source address. Note: This function is not necessary in normal operation of a CSRmesh network as DeviceID is distributed as part of the MASP protocol in the MASP_ID_DISTRIBUTION message.

```
CSRmeshResult ConfigSetParameters (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8
ttl, CSRMESH_CONFIG_SET_PARAMETERS_T *p_params)
```

Upon receiving a CONFIG_SET_PARAMETERS message, where the destination address is the DeviceID of this device, the device saves the TxInterval, TxDuration, RxDutyCycle, TxPower and TTL fields into the TransmitInterval, TransmitDuration, ReceiverDutyCycle, TransmitPower and DefaultTimeToLive state respectively. Then the device responds with a CONFIG_PARAMETERS message with the current configuration model state information.

```
CSRmeshResult ConfigGetParameters (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8
ttl, CSRMESH_CONFIG_GET_PARAMETERS_T *p_params)
```

Upon receiving a CONFIG_GET_PARAMETERS message, where the destination address is the DeviceID of this device, the device will respond with a CONFIG_PARAMETERS message with the current config model state information.

```
CSRmeshResult ConfigDiscoverDevice (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8
ttl, CSRMESH_CONFIG_DISCOVER_DEVICE_T *p_params)
```

Upon receiving a CONFIG_DISCOVER_DEVICE message directed at the 0x0000 group identifier or to DeviceID of this device, the device responds with a CONFIG_DEVICE_IDENTIFIER message.

```
CSRmeshResult ConfigGetInfo (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8 ttl,
CSRMESH_CONFIG_GET_INFO_T *p_params)
```

Upon receiving a CONFIG_GET_INFO message, directed at the DeviceID of this device, the device responds with a CONFIG_INFO message. The Info field of the CONFIG_GET_INFO message determines the information to be included in the CONFIG_INFO message. The following information values are defined: DeviceUUIDLow (0x00) contains the least significant eight octets of the DeviceUUID state value. DeviceUUIDHigh (0x01) contains the most significant eight octets of the DeviceUUID state value. ModelsLow (0x02) contains the least significant eight octets of the ModelsSupported state value. ModelsHigh (0x03) contains the most significant eight octets of the ModelsSupported state value.

```
CSRmeshResult ConfigSetMessageParams (CsrUint8 nw_id, CsrUint16 dest_id,
CsrUint8 ttl, CSRMESH_CONFIG_SET_MESSAGE_PARAMS_T *p_params)
```

This function packs the given parameters into a CSRmesh message and sends it over the network. When a response is received the application callback function is called to notify the CSRMESH_CONFIG_MESSAGE_PARAMS.

```
CSRmeshResult ConfigGetMessageParams (CsrUint8 nw_id, CsrUint16 dest_id,
CsrUint8 ttl, CSRMESH_CONFIG_GET_MESSAGE_PARAMS_T *p_params)
```

This function packs the given parameters into a CSRmesh message and sends it over the network. When a response is received the application callback function is called to notify the CSRMESH_CONFIG_MESSAGE_PARAMS.

254 config_model.h File Reference

Detailed Description

Defines CSRmesh Config Model specific data structures .

254.1 Data Structures of config_model.h File Reference

`struct CSRMESH_CONFIG_LAST_SEQUENCE_NUMBER_T`

CSRmesh Config Model message types.

`struct CSRMESH_CONFIG_RESET_DEVICE_T`

Upon receiving a CONFIG_RESET_DEVICE message from a trusted device directed at only this device, the local device sets the DeviceID to zero, and forgets all network keys, associated NetworkIVs and other configuration information. The device may act as if it is not associated and use MASP to re-associate with a network. Note: If the CONFIG_RESET_DEVICE message is received on any other destination address than the DeviceID of the local device, it is ignored. This is typically used when selling a device, to remove the device from the network of the seller so that the purchaser can associate the device with their network.

`struct CSRMESH_CONFIG_SET_DEVICE_IDENTIFIER_T`

When the device with a DeviceID of 0x0000 receives a CONFIG_SET_DEVICE_IDENTIFIER message and the DeviceHash of the message matches the DeviceHash of this device, the DeviceID of this device is set to the DeviceID field of this message. Then the device responds with the DEVICE_CONFIG_IDENTIFIER message using the new DeviceID as the source address. Note: This function is not necessary in normal operation of a CSRmesh network as DeviceID is distributed as part of the MASP protocol in the MASP_ID_DISTRIBUTION message.

`struct CSRMESH_CONFIG_SET_PARAMETERS_T`

Upon receiving a CONFIG_SET_PARAMETERS message, where the destination address is the DeviceID of this device, the device saves the TxInterval, TxDuration, RxDutyCycle, TxPower and TTL fields into the TransmitInterval, TransmitDuration, ReceiverDutyCycle, TransmitPower and DefaultTimeToLive state respectively. Then the device responds with a CONFIG_PARAMETERS message with the current configuration model state information.

`struct CSRMESH_CONFIG_GET_PARAMETERS_T`

Upon receiving a CONFIG_GET_PARAMETERS message, where the destination address is the DeviceID of this device, the device will respond with a CONFIG_PARAMETERS message with the current config model state information.

`struct CSRMESH_CONFIG_PARAMETERS_T`

Configuration parameters.

`struct CSRMESH_CONFIG_DISCOVER_DEVICE_T`

Upon receiving a CONFIG_DISCOVER_DEVICE message directed at the 0x0000 group identifier or to DeviceID of this device, the device responds with a CONFIG_DEVICE_IDENTIFIER message.

`struct CSRMESH_CONFIG_DEVICE_IDENTIFIER_T`

Device identifier.

`struct CSRMESH_CONFIG_GET_INFO_T`

Upon receiving a CONFIG_GET_INFO message, directed at the DeviceID of this device, the device responds with a CONFIG_INFO message. The Info field of the CONFIG_GET_INFO message determines the information to be included in the CONFIG_INFO message. The following information values are defined: DeviceUUIDLow (0x00) contains the least significant eight octets of the DeviceUUID state value. DeviceUUIDHigh (0x01) contains the most significant eight octets of the DeviceUUID state value. ModelsLow (0x02) contains the least significant eight octets of the ModelsSupported state value. ModelsHigh (0x03) contains the most significant eight octets of the ModelsSupported state value.

`struct CSRMESH_CONFIG_INFO_T`

Current device information.

255 config_server.h File Reference

Detailed Description

This file provides the prototypes of the server functions defined in the CSRmesh Config model.

255.1 Functions of config_server.h File Reference

```
CSRmeshResult ConfigModelInit (CsrUint8 nw_id, CsrUint16 *group_id_list,
CsrUint16 num_groups, CSRMESH_MODEL_CALLBACK_T app_callback)
```

Model Initialisation.

```
CSRmeshResult ConfigParameters (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8
ttl, CSRMESH_CONFIG_PARAMETERS_T *p_params)
```

Configuration parameters.

```
CSRmeshResult ConfigDeviceIdentifier (CsrUint8 nw_id, CsrUint16 dest_id,
CsrUint8 ttl, CSRMESH_CONFIG_DEVICE_IDENTIFIER_T *p_params)
```

Device identifier.

```
CSRmeshResult ConfigInfo (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8 ttl,
CSRMESH_CONFIG_INFO_T *p_params)
```

Current device information.

```
CSRmeshResult ConfigMessageParams (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8
ttl, CSRMESH_CONFIG_MESSAGE_PARAMS_T *p_params)
```

This function packs the given parameters into a CSRmesh message and sends it over the network.

256 csr_macro.h File Reference

Detailed Description

defines and functions for csr_macro.h

257 csr_mesh.h File Reference

Detailed Description

CSRmesh library configuration and control functions.

257.1 Functions of csr_mesh.h File Reference

```
CSRmeshResult CSRmeshInit (CSR_MESH_CONFIG_FLAG_T configFlag)
```

Initialize CSRmesh core mesh stack.

```
CSRmeshResult CSRmeshRegisterAppCallback (CSR_MESH_APP_CB_T callback)
```

Register application callback.

```
CSRmeshResult CSRmeshStart (void)
```

Start the CSRmesh system.

```
CSRmeshResult CSRmeshStop (void)
```

Stop the CSRmesh system.

```
CSRmeshResult CSRmeshReset (void)
```

Reset the CSRmesh library.

```
CSRmeshResult CSRmeshSetDefaultTTL (CsrUint8 ttl)
```

Sets a default ttl value.

```
CSRmeshResult CSRmeshGetDefaultTTL (CSR_MESH_APP_EVENT_DATA_T *eventData)
```

Gets the default ttl value.

```
CSRmeshResult CSRmeshRemoveNetwork (CsrUint8 netId)
```

Remove a network.

```
CSRmeshResult CSRmeshGetDeviceID (CsrUint8 netId, CSR_MESH_APP_EVENT_DATA_T
*eventData)
```

Gets the 16-bit Device Identifier of the CSRmesh device.

```
CSRmeshResult CSRmeshGetDeviceUUID (CSR_MESH_APP_EVENT_DATA_T *eventData)
```

Get the CSRmesh library 128 bit UUID.

```
CSRmeshResult CSRmeshSetTransmitState (CSR_MESH_TRANSMIT_STATE_T
*transmitStateArg, CSR_MESH_APP_EVENT_DATA_T *eventData)
```

CSRmeshSetTransmitState .

```
CSRmeshResult CSRmeshGetTransmitState (CsrUint8 netId,
CSR_MESH_APP_EVENT_DATA_T *eventData)
```

CSRmeshGetTransmitState .

```
CSRmeshResult CSRmeshAssociateToANetwork (CSR_MESH_DEVICE_APPEARANCE_T
*deviceAppearance, CsrUint8 ttl)
```

Advertises a CSRmesh device identification message.

```
void CSRmeshCalculateSHA256Hash (const CsrUint8 *string, CsrUint8 length,
CsrUint16 *hash)
```

This function generates a 32 byte hash.

```
CSRmeshResult CSRmeshSetMeshDuplicateMessageCacheSize (CsrUint8 cacheSize)
```

Controls the depth of the duplicate check cache in MTL.

```
CSRmeshResult CSRmeshSendKeyIVStatusRequest (const CSR_MESH_MASP_DEVICE_DATA_T
*devData, CsrUint8 ttl)
```

Transmit Key-IV request.

```
CSRmeshResult CSRmeshSetSrcSequenceCache (CsrUint8 netId, CSR_MESH_SEQ_CACHE_T
*seqCache)
```

CSRmeshSetSequenceSrcCache.

258 csr_mesh_model_common.h File Reference

Detailed Description

This files defines data types and constants that are commonly used across models.

258.1 Data Structures of csr_mesh_model_common.h File Reference

```
struct  CSRMESH_EVENT_DATA_T
```

CSRmesh Event Data.

258.2 Typedefs of csr_mesh_model_common.h File Reference

```
typedef CSRmeshResult(* CSRMESH_MODEL_CALLBACK_T) (CSRMESH_MODEL_EVENT_T
event_code, CSRMESH_EVENT_DATA_T *data, CsrUint16 length, void **state_data)
```

MCP Model handler function type.

258.3 Enumerations of csr_mesh_model_common.h File Reference

```
enum  csr_mesh_boolean_t { csr_mesh_boolean_false = 0,
csr_mesh_boolean_true = 1
}
```

CSRmesh boolean type.

```
enum  csr_mesh_power_state_t { csr_mesh_power_state_off = 0,
csr_mesh_power_state_on = 1,
csr_mesh_power_state_standby = 2,
csr_mesh_power_state_onfromstandby = 3
}
```

CSRmesh power_state type.

```
enum  csr_mesh_device_information_t {
  csr_mesh_device_information_uuid_low = 0,
csr_mesh_device_information_uuid_high = 1,
csr_mesh_device_information_model_low = 2,
csr_mesh_device_information_model_high = 3,
  csr_mesh_device_information_vid_pid_version = 4,
csr_mesh_device_information_appearance = 5,
csr_mesh_device_information_lastetag = 6
}
```

CSRmesh device_information type.

```
enum  csr_mesh_key_properties_t { csr_mesh_key_properties_administrator = 0,
csr_mesh_key_properties_user = 1,
csr_mesh_key_properties_guest = 2,
csr_mesh_key_properties_relayonly = 3
}
```

CSRmesh key_properties type.

```
enum  csr_mesh_month_of_year_t {
  csr_mesh_month_of_year_unknown = 0,
csr_mesh_month_of_year_january = 1,
csr_mesh_month_of_year_february = 2,
csr_mesh_month_of_year_march = 3,
  csr_mesh_month_of_year_april = 4,
csr_mesh_month_of_year_may = 5,
csr_mesh_month_of_year_june = 6,
csr_mesh_month_of_year_july = 7,
  csr_mesh_month_of_year_august = 8,
csr_mesh_month_of_year_september = 9,
csr_mesh_month_of_year_october = 10,
csr_mesh_month_of_year_november = 11,
  csr_mesh_month_of_year_december = 12
}
```

CSRmesh month_of_year type.

```
enum  csr_mesh_timer_mode_t { csr_mesh_timer_mode_programming = 0,
csr_mesh_timer_mode_active = 1,
csr_mesh_timer_mode_partly_random = 2,
csr_mesh_timer_mode_completely_random = 3
}
```

CSRmesh timer_mode type.

```
enum  csr_mesh_remote_code_t {
  csr_mesh_remote_code_number_0 = 0,
csr_mesh_remote_code_number_1 = 1,
csr_mesh_remote_code_number_2 = 2,
csr_mesh_remote_code_number_3 = 3,
  csr_mesh_remote_code_number_4 = 4,
csr_mesh_remote_code_number_5 = 5,
csr_mesh_remote_code_number_6 = 6,
csr_mesh_remote_code_number_7 = 7,
  csr_mesh_remote_code_number_8 = 8,
csr_mesh_remote_code_number_9 = 9,
csr_mesh_remote_code_direction_n = 16,
csr_mesh_remote_code_direction_e = 17,
  csr_mesh_remote_code_direction_s = 18,
csr_mesh_remote_code_direction_w = 19,
csr_mesh_remote_code_direction_ne = 20,
csr_mesh_remote_code_direction_se = 21,
  csr_mesh_remote_code_direction_sw = 22,
csr_mesh_remote_code_direction_nw = 23,
csr_mesh_remote_code_direction_nne = 24,
csr_mesh_remote_code_direction_ene = 25,
  csr_mesh_remote_code_direction_ese = 26,
csr_mesh_remote_code_direction_sse = 27,
csr_mesh_remote_code_direction_ssw = 28,
csr_mesh_remote_code_direction_wsw = 29,
  csr_mesh_remote_code_direction_wnw = 30,
csr_mesh_remote_code_direction_nnw = 31,
csr_mesh_remote_code_select = 32,
csr_mesh_remote_code_channel_up = 33,
  csr_mesh_remote_code_channel_down = 34,
csr_mesh_remote_code_volume_up = 35,
csr_mesh_remote_code_volume_down = 36,
csr_mesh_remote_code_volume_mute = 37,
  csr_mesh_remote_code_menu = 38,
csr_mesh_remote_code_back = 39,
csr_mesh_remote_code_guide = 40,
csr_mesh_remote_code_play = 41,
  csr_mesh_remote_code_pause = 42,
csr_mesh_remote_code_stop = 43,
csr_mesh_remote_code_fast_forward = 44,
csr_mesh_remote_code_fast_rewind = 45,
  csr_mesh_remote_code_skip_forward = 46,
csr_mesh_remote_code_skip_backward = 47
}
```

CSRmesh remote_code type.

```
enum  csr_mesh_sensor_type_t {
  csr_mesh_sensor_type_unknown = 0,
csr_mesh_sensor_type_internal_air_temperature = 1,
csr_mesh_sensor_type_external_air_temperature = 2,
csr_mesh_sensor_type_desired_air_temperature = 3,
  csr_mesh_sensor_type_internal_humidity = 4,
csr_mesh_sensor_type_external_humidity = 5,
csr_mesh_sensor_type_external_dewpoint = 6,
csr_mesh_sensor_type_internal_door = 7,
  csr_mesh_sensor_type_external_door = 8,
csr_mesh_sensor_type_internal_window = 9,
csr_mesh_sensor_type_external_window = 10,
csr_mesh_sensor_type_solar_energy = 11,
  csr_mesh_sensor_type_number_of_activations = 12,
csr_mesh_sensor_type_fridge_temperature = 13,
csr_mesh_sensor_type_desired_fridge_temperature = 14,
csr_mesh_sensor_type_freezer_temperature = 15,
  csr_mesh_sensor_type_desired_freezer_temperature = 16,
csr_mesh_sensor_type_oven_temperature = 17,
csr_mesh_sensor_type_desired_oven_temperature = 18,
csr_mesh_sensor_type_seat_occupied = 19,
  csr_mesh_sensor_type_washing_machine_state = 20,
csr_mesh_sensor_type_dish_washer_state = 21,
csr_mesh_sensor_type_clothes_dryer_state = 22,
csr_mesh_sensor_type_toaster_state = 23,
  csr_mesh_sensor_type_carbon_dioxide = 24,
csr_mesh_sensor_type_carbon_monoxide = 25,
csr_mesh_sensor_type_smoke = 26,
csr_mesh_sensor_type_water_level = 27,
  csr_mesh_sensor_type_hot_water_temperature = 28,
csr_mesh_sensor_type_cold_water_temperature = 29,
csr_mesh_sensor_type_desired_water_temperature = 30,
csr_mesh_sensor_type_cooker_hob_back_left_state = 31,
  csr_mesh_sensor_type_desired_cooker_hob_back_left_state = 32,
csr_mesh_sensor_type_cooker_hob_front_left_state = 33,
csr_mesh_sensor_type_desired_cooker_hob_front_left_state = 34,
csr_mesh_sensor_type_cooker_hob_back_middle_state = 35,
  csr_mesh_sensor_type_desired_cooker_hob_back_middle_state = 36,
csr_mesh_sensor_type_cooker_hob_front_middle_state = 37,
csr_mesh_sensor_type_desired_cooker_hob_front_middle_state = 38,
csr_mesh_sensor_type_cooker_hob_back_right_state = 39,
  csr_mesh_sensor_type_desired_cooker_hob_back_right_state = 40,
csr_mesh_sensor_type_cooker_hob_front_right_state = 41,
csr_mesh_sensor_type_desired_cooker_hob_front_right_state = 42,
csr_mesh_sensor_type_desired_wakeup_alarm_time = 43,
  csr_mesh_sensor_type_desired_second_wakeup_alarm_time = 44,
csr_mesh_sensor_type_passive_infrared_state = 45,
csr_mesh_sensor_type_water_flowing = 46,
csr_mesh_sensor_type_desired_water_flow = 47,
  csr_mesh_sensor_type_audio_level = 48,
csr_mesh_sensor_type_desired_audio_level = 49,
csr_mesh_sensor_type_fan_speed = 50,
csr_mesh_sensor_type_desired_fan_speed = 51,
```

```
  csr_mesh_sensor_type_wind_speed = 52,
csr_mesh_sensor_type_wind_speed_gust = 53,
csr_mesh_sensor_type_wind_direction = 54,
csr_mesh_sensor_type_wind_direction_gust = 55,
  csr_mesh_sensor_type_rain_fall_last_hour = 56,
csr_mesh_sensor_type_rain_fall_today = 57,
csr_mesh_sensor_type_barometric_pressure = 58,
csr_mesh_sensor_type_soil_temperature = 59,
  csr_mesh_sensor_type_soil_moisure = 60,
csr_mesh_sensor_type_window_cover_position = 61,
csr_mesh_sensor_type_desired_window_cover_position = 62,
csr_mesh_sensor_type_generic_1_byte = 63,
  csr_mesh_sensor_type_generic_2_byte = 64,
csr_mesh_sensor_type_generic_1_byte_typed = 250,
csr_mesh_sensor_type_generic_2_byte_typed = 251,
csr_mesh_sensor_type_generic_3_byte_typed = 252
}
```

CSRmesh sensor_type type.

```
enum  csr_mesh_beacon_type_t {
  csr_mesh_beacon_type_csr = 0,
csr_mesh_beacon_type_ibeacon = 1,
csr_mesh_beacon_type_eddystone_url = 2,
csr_mesh_beacon_type_eddystone_uid = 3,
  csr_mesh_beacon_type_lte_direct = 4,
csr_mesh_beacon_type_lumicast = 5
}
```

CSRmesh beacon_type type.

```
enum  csr_mesh_door_state_t { csr_mesh_door_state_open_unlocked = 2,
csr_mesh_door_state_closed_unlocked = 1,
csr_mesh_door_state_closed_locked = 0
}
```

CSRmesh door_state type.

```
enum  csr_mesh_window_state_t { csr_mesh_window_state_closed_insecure = 1,
csr_mesh_window_state_open_insecure = 3,
csr_mesh_window_state_open_secure = 2,
csr_mesh_window_state_closed_secure = 0
}
```

CSRmesh window_state type.

```
enum  csr_mesh_seat_state_t { csr_mesh_seat_state_empty = 0,
csr_mesh_seat_state_continuously_occupied = 2,
csr_mesh_seat_state_occupied = 1
}
```

CSRmesh seat_state type.

```
enum  csr_mesh_appliance_state_t {
  csr_mesh_appliance_state_pre_wash = 3,
csr_mesh_appliance_state_idle = 0,
csr_mesh_appliance_state_post_wash = 5,
csr_mesh_appliance_state_washing = 4,
  csr_mesh_appliance_state_heating_up = 1,
csr_mesh_appliance_state_cooling_down = 2
}
```

CSRmesh appliance_state type.

```
enum  csr_mesh_cooker_hob_state_t {
  csr_mesh_cooker_hob_state_level_2 = 2,
csr_mesh_cooker_hob_state_off_hot = 10,
csr_mesh_cooker_hob_state_level_1 = 1,
csr_mesh_cooker_hob_state_level_7 = 7,
  csr_mesh_cooker_hob_state_level_9 = 9,
csr_mesh_cooker_hob_state_off_cold = 0,
csr_mesh_cooker_hob_state_level_8 = 8,
csr_mesh_cooker_hob_state_level_5 = 5,
  csr_mesh_cooker_hob_state_level_4 = 4,
csr_mesh_cooker_hob_state_level_3 = 3,
csr_mesh_cooker_hob_state_level_6 = 6
}
```

CSRmesh cooker_hob_state type.

```
enum  csr_mesh_movement_state_t { csr_mesh_movement_state_no_movement = 0,
csr_mesh_movement_state_movement_detected = 1
}
```

CSRmesh movement_state type.

```
enum  csr_mesh_forward_backward_t { csr_mesh_forward_backward_forward = 1,
csr_mesh_forward_backward_backwards = 2
}
```

CSRmesh forward_backward type.

```
enum  csr_mesh_direction_t {
  csr_mesh_direction_north_of_north_east = 16,
csr_mesh_direction_north = 0,
csr_mesh_direction_north_east = 32,
csr_mesh_direction_east = 64,
  csr_mesh_direction_north_west = 224,
csr_mesh_direction_north_of_north_west = 240,
csr_mesh_direction_east_of_north_east = 48,
csr_mesh_direction_west = 192,
  csr_mesh_direction_south_east = 96,
csr_mesh_direction_east_of_south_east = 80,
csr_mesh_direction_south_of_south_east = 112,
csr_mesh_direction_south = 128,
  csr_mesh_direction_west_of_north_west = 208,
csr_mesh_direction_south_of_south_west = 144,
csr_mesh_direction_west_of_south_west = 176,
csr_mesh_direction_south_west = 160
}
```

CSRmesh direction type.

```
enum  CSRMESH_MODEL_TYPE_T {
  CSRMESH_WATCHDOG_MODEL = 0,
CSRMESH_CONFIG_MODEL = 1,
CSRMESH_GROUP_MODEL = 2,
CSRMESH_SENSOR_MODEL = 4,
  CSRMESH_ACTUATOR_MODEL = 5,
CSRMESH_DATA_MODEL = 8,
CSRMESH_BEARER_MODEL = 11,
CSRMESH_PING_MODEL = 12,
  CSRMESH_BATTERY_MODEL = 13,
CSRMESH_ATTENTION_MODEL = 14,
CSRMESH_POWER_MODEL = 19,
CSRMESH_LIGHT_MODEL = 20,
  CSRMESH_ASSET_MODEL = 6,
CSRMESH_TRACKER_MODEL = 7,
CSRMESH_TIME_MODEL = 16,
CSRMESH_SWITCH_MODEL = 21,
  CSRMESH_TUNING_MODEL = 28,
CSRMESH_EXTENSION_MODEL = 29,
CSRMESH_LARGEOBJECTTRANSFER_MODEL = 30,
CSRMESH_FIRMWARE_MODEL = 9,
  CSRMESH_DIAGNOSTIC_MODEL = 10,
CSRMESH_ACTION_MODEL = 34,
CSRMESH_BEACON_MODEL = 32,
CSRMESH_BEACONPROXY_MODEL = 33,
  CSRMESH_ALL_MODELS = 255
}
```

CSRmesh Model types.

```
enum  CSRMESH_MODEL_EVENT_T {
  CSRMESH_WATCHDOG_MESSAGE = 0x00,
CSRMESH_WATCHDOG_SET_INTERVAL = 0x01,
CSRMESH_WATCHDOG_INTERVAL = 0x02,
CSRMESH_CONFIG_LAST_SEQUENCE_NUMBER = 0x03,
  CSRMESH_CONFIG_RESET_DEVICE = 0x04,
CSRMESH_CONFIG_SET_DEVICE_IDENTIFIER = 0x05,
CSRMESH_CONFIG_SET_PARAMETERS = 0x06,
CSRMESH_CONFIG_GET_PARAMETERS = 0x07,
  CSRMESH_CONFIG_PARAMETERS = 0x08,
CSRMESH_CONFIG_DISCOVER_DEVICE = 0x09,
CSRMESH_CONFIG_DEVICE_IDENTIFIER = 0x0A,
CSRMESH_CONFIG_GET_INFO = 0x0B,
  CSRMESH_CONFIG_INFO = 0x0C
, CSRMESH_GROUP_GET_NUMBER_OF_MODEL_GROUPIDS = 0x0D,
CSRMESH_GROUP_NUMBER_OF_MODEL_GROUPIDS = 0x0E,
CSRMESH_GROUP_SET_MODEL_GROUPID = 0x0F,
  CSRMESH_GROUP_GET_MODEL_GROUPID = 0x10,
CSRMESH_GROUP_MODEL_GROUPID = 0x11,
CSRMESH_SENSOR_GET_TYPES = 0x20,
CSRMESH_SENSOR_TYPES = 0x21,
  CSRMESH_SENSOR_SET_STATE = 0x22,
CSRMESH_SENSOR_GET_STATE = 0x23,
CSRMESH_SENSOR_STATE = 0x24,
CSRMESH_SENSOR_WRITE_VALUE_NO_ACK = 0x26,
  CSRMESH_SENSOR_WRITE_VALUE = 0x25,
CSRMESH_SENSOR_READ_VALUE = 0x27,
CSRMESH_SENSOR_VALUE = 0x28,
CSRMESH_SENSOR_MISSING = 0x29,
  CSRMESH_ACTUATOR_GET_TYPES = 0x30,
CSRMESH_ACTUATOR_TYPES = 0x31,
CSRMESH_ACTUATOR_SET_VALUE_NO_ACK = 0x32,
CSRMESH_ACTUATOR_SET_VALUE = 0x33,
  CSRMESH_ACTUATOR_VALUE_ACK = 0x35,
CSRMESH_ACTUATOR_GET_VALUE_ACK = 0x34,
CSRMESH_DATA_STREAM_FLUSH = 0x70,
CSRMESH_DATA_STREAM_SEND = 0x71,
  CSRMESH_DATA_STREAM_RECEIVED = 0x72,
CSRMESH_DATA_BLOCK_SEND = 0x73,
CSRMESH_BEARER_SET_STATE = 0x8100,
CSRMESH_BEARER_GET_STATE = 0x8101,
  CSRMESH_BEARER_STATE = 0x8102,
CSRMESH_PING_REQUEST = 0x8200,
CSRMESH_PING_RESPONSE = 0x8201,
CSRMESH_BATTERY_GET_STATE = 0x8300,
  CSRMESH_BATTERY_STATE = 0x8301,
CSRMESH_ATTENTION_SET_STATE = 0x8400,
CSRMESH_ATTENTION_STATE = 0x8401,
CSRMESH_POWER_SET_STATE_NO_ACK = 0x8900,
  CSRMESH_POWER_SET_STATE = 0x8901,
CSRMESH_POWER_TOGGLE_STATE_NO_ACK = 0x8902,
CSRMESH_POWER_TOGGLE_STATE = 0x8903,
CSRMESH_POWER_GET_STATE = 0x8904,
```

```
  CSRMESH_POWER_STATE_NO_ACK = 0x8906,
CSRMESH_POWER_STATE = 0x8905,
CSRMESH_LIGHT_SET_LEVEL_NO_ACK = 0x8A00,
CSRMESH_LIGHT_SET_LEVEL = 0x8A01,
  CSRMESH_LIGHT_SET_RGB_NO_ACK = 0x8A02,
CSRMESH_LIGHT_SET_RGB = 0x8A03,
CSRMESH_LIGHT_SET_POWER_LEVEL_NO_ACK = 0x8A04,
CSRMESH_LIGHT_SET_POWER_LEVEL = 0x8A05,
  CSRMESH_LIGHT_SET_COLOR_TEMP = 0x8A06,
CSRMESH_LIGHT_GET_STATE = 0x8A07,
CSRMESH_LIGHT_STATE_NO_ACK = 0x8A09,
CSRMESH_LIGHT_STATE = 0x8A08,
  CSRMESH_LIGHT_SET_WHITE_NO_ACK = 0x8A0B,
CSRMESH_LIGHT_SET_WHITE = 0x8A0A,
CSRMESH_LIGHT_GET_WHITE = 0x8A0C,
CSRMESH_LIGHT_WHITE_NO_ACK = 0x8A0E,
  CSRMESH_LIGHT_WHITE = 0x8A0D,
CSRMESH_ASSET_SET_STATE = 0x40,
```

```
CSRMESH_ASSET_GET_STATE = 0x41,
CSRMESH_ASSET_STATE = 0x42,
  CSRMESH_ASSET_ANNOUNCE = 0x43,
CSRMESH_TRACKER_FIND = 0x44,
CSRMESH_TRACKER_FOUND = 0x45,
CSRMESH_TRACKER_REPORT = 0x46,
  CSRMESH_TRACKER_CLEAR_CACHE = 0x47,
CSRMESH_TRACKER_SET_PROXIMITY_CONFIG = 0x48,
CSRMESH_TIME_SET_STATE = 0x75,
CSRMESH_TIME_GET_STATE = 0x76,
  CSRMESH_TIME_STATE = 0x77,
CSRMESH_TIME_BROADCAST = 0x7F,
CSRMESH_TUNING_PROBE = 0xEF01,
CSRMESH_TUNING_GET_STATS = 0xEF02,
  CSRMESH_TUNING_STATS = 0xEF03,
CSRMESH_TUNING_ACK_CONFIG = 0xEF04,
CSRMESH_TUNING_SET_CONFIG = 0xEF05,
CSRMESH_EXTENSION_REQUEST = 0xEF10,
  CSRMESH_EXTENSION_CONFLICT = 0xEF11,
CSRMESH_LARGEOBJECTTRANSFER_ANNOUNCE = 0x1A,
CSRMESH_LARGEOBJECTTRANSFER_INTEREST = 0xEF30,
CSRMESH_FIRMWARE_GET_VERSION = 0x78,
  CSRMESH_FIRMWARE_VERSION = 0x79,
CSRMESH_FIRMWARE_UPDATE_REQUIRED = 0x7A,
CSRMESH_FIRMWARE_UPDATE_ACKNOWLEDGED = 0x7b,
CSRMESH_DIAGNOSTIC_STATE = 0xEF40
,
  CSRMESH_ACTION_SET_ACTION = 0x50,
CSRMESH_ACTION_SET_ACTION_ACK = 0x51,
CSRMESH_ACTION_GET_ACTION_STATUS = 0x52,
CSRMESH_ACTION_ACTION_STATUS = 0x53,
  CSRMESH_ACTION_DELETE = 0x54,
CSRMESH_ACTION_DELETE_ACK = 0x55,
CSRMESH_ACTION_GET = 0x56,
CSRMESH_BEACON_SET_STATUS = 0x60,
  CSRMESH_BEACON_GET_BEACON_STATUS = 0x62,
CSRMESH_BEACON_BEACON_STATUS = 0x63,
CSRMESH_BEACON_GET_TYPES = 0x64,
CSRMESH_BEACON_TYPES = 0x65,
  CSRMESH_BEACON_SET_PAYLOAD = 0x66,
CSRMESH_BEACON_PAYLOAD_ACK = 0x67,
CSRMESH_BEACON_GET_PAYLOAD = 0x68,
CSRMESH_BEACONPROXY_ADD = 0x69,
  CSRMESH_BEACONPROXY_REMOVE = 0x6A,
CSRMESH_BEACONPROXY_COMMAND_STATUS_DEVICES = 0x6B,
CSRMESH_BEACONPROXY_GET_STATUS = 0x6C,
CSRMESH_BEACONPROXY_PROXY_STATUS = 0x6D,
  CSRMESH_BEACONPROXY_GET_DEVICES = 0x6E,
CSRMESH_BEACONPROXY_DEVICES = 0x6F
}
```

CSRmesh Model Event Code types.

259 csr_mesh_result.h File Reference

Detailed Description

CSRmesh library result type.

259.1 Macros of csr_mesh_result.h File Reference

```
#define CSR_MESH_RESULT_SUCCESS   ((CSRmeshResult) 0x0000)
```

CSR Mesh Operation status Success.

```
#define CSR_MESH_RESULT_INPROGRESS   ((CSRmeshResult) 0x0001)
```

CSR Mesh Operation status InProgress.

```
#define CSR_MESH_RESULT_MESH_INVALID_STATE   ((CSRmeshResult) 0x0002)
```

CSR Mesh Operation status Mesh_Invalid_State.

```
#define CSR_MESH_RESULT_MODEL_NOT_REGISTERED   ((CSRmeshResult) 0x0003)
```

CSR Mesh Operation statusModel_Not_Registered.

```
#define CSR_MESH_RESULT_MODEL_ALREADY_REGISTERD   ((CSRmeshResult) 0x0004)
```

CSR Mesh Operation status Model_Already_Registered.

```
#define CSR_MESH_RESULT_ROLE_NOT_SUPPORTED   ((CSRmeshResult) 0x0005)
```

CSR Mesh Operation status Role_Not_Supported.

```
#define CSR_MESH_RESULT_INVALID_NWK_ID   ((CSRmeshResult) 0x0006)
```

CSR MeshOperation status Invalid_Network_id.

```
#define CSR_MESH_RESULT_EXCEED_MAX_NO_OF_NWKS   ((CSRmeshResult) 0x0007)
```

CSR MeshOperation status Exceed_Max_No_Of_Network.

```
#define CSR_MESH_RESULT_NOT_READY   ((CSRmeshResult) 0x0008)
```

CSR Mesh Operation statusNot_ready.

```
#define CSR_MESH_RESULT_MASP_ALREADY_ASSOCIATING   ((CSRmeshResult) 0x0009)
```

CSR Mesh Operation status Masp_Already_Associating.

```
#define CSR_MESH_RESULT_API_NOT_SUPPORTED   ((CSRmeshResult) 0x000A)
```

CSR Mesh Operation status Api_Not_Supported.

```
#define CSR_MESH_RESULT_INVALID_ARG   ((CSRmeshResult) 0x000B)
```

CSR Mesh Operation status Invalid argument.

```
#define CSR_MESH_RESULT_RADIO_BUSY   ((CSRmeshResult) 0x000C)
```

CSR Mesh Operation status radio busy.

```
#define CSR_MESH_RESULT_KEY_ALREADY_EXISTS   ((CSRmeshResult) 0x000D)
```

CSR Mesh Operation status key already present.

```
#define CSR_MESH_RESULT_FAILURE   ((CSRmeshResult) 0xFFFF)
```

CSR Mesh Operation status Failure.

259.2 Typedefs of csr_mesh_result.h File Reference

```
typedef CsrUint16 CSRmeshResult
```

CSR Mesh operation status Type.

260 csr_mesh_types.h File Reference

Detailed Description

CSRmesh library data types.

260.1 Data Structures of csr_mesh_types.h File Reference

`struct CSR_MESH_UUID_T`

128-bit UUID type

`struct CSR_MESH_VID_PID_VERSION_T`

CSRmesh Product ID, Vendor ID and Version Information.

`struct CSR_MESH_CONFORMANCE_SIGNATURE_T`

CSRmesh conformance signature Information.

`struct CSR_MESH_STACK_VERSION_T`

CSRmesh stack version Information.

`struct CSR_MESH_CC_INFO_T`

Client config info.

`struct CSR_MESH_AUTH_CODE_T`

64 bit Authorisation Code type

`struct CSR_MESH_CONFIG_BEARER_PARAM_T`

CSRmesh Scan and Advertising Parameters.

`struct CSR_MESH_CONFIG_MSG_PARAMS_T`

CSRmesh Scan and Advertising Parameters.

`struct CSR_MESH_NETID_LIST_T`

CSR Mesh Network Id List.

`struct CSR_MESH_NW_RELAY_T`

CSR Mesh transmit state.

`struct CSR_MESH_TRANSMIT_STATE_T`

CSR Mesh transmit state.

`struct CSR_MESH_TTL_T`

CSR Mesh TTL.

`struct CSR_MESH_APPEARANCE_T`

CSRmesh Device Appearance. The Appearance is a 24-bit value that is composed of an "organization" and an "organization appearance".

`struct CSR_MESH_DEVICE_APPEARANCE_T`

Device Appearance data type.

`struct CSR_MESH_MASP_DEVICE_IND_T`

Device indication data type.

`struct CSR_MESH_ASSOCIATION_ATTENTION_DATA_T`

Association attention request data type.

`struct CSR_MESH_GROUP_ID_RELATED_DATA_T`

CSR Mesh Group Id Related Data - To provide to app while handling Group model data in core mesh.

`struct CSR_MESH_BEARER_STATE_DATA_T`

CSR Mesh Bearer state Type.

`struct CSR_MESH_SQN_LOOKUP_TABLE_T`

CSR Mesh <<SRC,SEQ>> Lookup Table.

`struct CSR_MESH_SEQ_CACHE_T`

CSR mesh <<SRC,SEQ>> Cache Structure.

`struct CSR_MESH_MESH_ID_DATA_T`

CSR Mesh mesh_id data Type.

`struct CSR_MESH_DEVICE_ID_UPDATE_T`

CSR Mesh device id update data Type.

`struct CSR_MESH_APP_EVENT_DATA_T`

CSR Mesh App Event Data - to provide event, status * app data to app.

`struct CSR_MESH_MASP_DEVICE_DATA_T`

CSRmesh Device Identifier Data and Signature.

`struct CSR_MESH_MASP_DEVICE_SIGN_LIST_NODE_T`

CSRmesh Device Signature Data Linked List node.

260.2 Macros of csr_mesh_types.h File Reference

```
#define CSR_MESH_MAX_NO_TIMERS   (7)
```

Number of timers required for CSRmesh library to be reserved by the application.

```
#define CSR_MESH_NO_OF_NWKS   (1)
```

Maximum number of mesh network the CSRmesh stack supports. The CSRmesh Stack library is built with this configuration. ** This should not be modified **.

```
#define CSR_MESH_INVALID_NWK_ID   (0xFF)
```

Invalid network id.

```
#define CSR_MESH_SHORT_NAME_LENGTH   (9)
```

short name for the device

```
#define MAX_ADV_DATA_LEN   (31)
```

This constant is used in the main server app to define array that is large enough to hold the advertisement data.

```
#define CSR_MESH_NETWORK_KEY_SIZE_BYTES   (16)
```

CSRmesh Network key in bytes.

```
#define CSR_MESH_MESH_ID_SIZE_BYTES   (16)
```

CSRmesh Mesh_Id size in bytes.

```
#define LE_BEARER_ACTIVE   (1)
```

LE Bearer Type.

```
#define GATT_SERVER_BEARER_ACTIVE   (2)
```

GATT Bearer Type.

```
#define CSR_MESH_DEVICE_SIGN_SIZE_IN_BYTES   (24)
```

Device Signature Size in Words and Bytes.

260.3 Typedefs of csr_mesh_types.h File Reference

```
typedef void(* CSR_MESH_APP_CB_T) (CSR_MESH_APP_EVENT_DATA_T eventDataCallback)
```

CSRmesh Application callback handler function.

260.4 Enumerations of csr_mesh_types.h File Reference

```
enum  CSR_MESH_CONFIG_FLAG_T { CSR_MESH_CONFIG_DEVICE = 0,
CSR_MESH_NON_CONFIG_DEVICE = 1,
CSR_MESH_CONFIG_DEVICE_WITH_AUTH_CODE = 2,
CSR_MESH_NON_CONFIG_DEVICE_WITH_AUTH_CODE = 3
}
```

Flag determining the type of the device.

```
enum  CSR_MESH_OPERATION_STATUS_T {
  CSR_MESH_OPERATION_SUCCESS = 0x00,
CSR_MESH_OPERATION_STACK_NOT_INITIALIZED = 0x01,
CSR_MESH_OPERATION_NOT_PERMITTED = 0x02,
CSR_MESH_OPERATION_MEMORY_FULL = 0x03,
  CSR_MESH_OPERATION_REQUEST_FOR_INFO = 0x04,
CSR_MESH_OPERATION_GENERIC_FAIL = 0xFF
}
```

Operation status passed with App-Callback to indicate the result of an asynchronous operation or to inform the App that the callback is mader to request info.

```
enum  CSR_MESH_MESSAGE_T { CSR_MESH_MESSAGE_ASSOCIATION,
CSR_MESH_MESSAGE_CONTROL
}
```

CSRmesh Message types.

```
enum  CSR_MESH_EVENT_T {
  CSR_MESH_INIT_EVENT = 0x0001,
CSR_MESH_REGISTER_APP_CB_EVENT = 0x0002,
CSR_MESH_RESET_EVENT = 0x0003,
CSR_MESH_START_EVENT = 0x0004,
  CSR_MESH_STOP_EVENT = 0x0005,
CSR_MESH_TRANSMIT_STATE_EVENT = 0x0006,
CSR_MESH_START_DEVICE_INQUIRY_EVENT = 0x0007,
CSR_MESH_ASSOC_STARTED_EVENT = 0x0008,
  CSR_MESH_ASSOC_COMPLETE_EVENT = 0x0009,
CSR_MESH_SEND_ASSOC_COMPLETE_EVENT = 0x000A,
CSR_MESH_GET_DEVICE_ID_EVENT = 0x000B,
CSR_MESH_GET_DEVICE_UUID_EVENT = 0x000C,
  CSR_MESH_MASP_DEVICE_IND_EVENT = 0x000D,
CSR_MESH_MASP_DEVICE_APPEARANCE_EVENT = 0x000E,
CSR_MESH_NETWORK_ID_LIST_EVENT = 0x000F,
CSR_MESH_SET_MAX_NO_OF_NETWORK_EVENT = 0x0010,
  CSR_MESH_SET_PASSPHRASE_EVENT = 0x0011,
CSR_MESH_SET_NETWORK_KEY_EVENT = 0x0012,
CSR_MESH_CONFIG_RESET_DEVICE_EVENT = 0x0013,
CSR_MESH_CONFIG_SET_PARAMS_EVENT = 0x0014,
  CSR_MESH_CONFIG_GET_PARAMS_EVENT = 0x0015,
CSR_MESH_GET_VID_PID_VERSTION_EVENT = 0x0016,
CSR_MESH_GET_DEVICE_APPEARANCE_EVENT = 0x0017,
CSR_MESH_GROUP_SET_MODEL_GROUPID_EVENT = 0x0018,
  CSR_MESH_SEND_RAW_MCP_MSG_EVENT = 0x0019,
CSR_MESH_SEND_MCP_MSG_EVENT = 0x001A,
CSR_MESH_MCP_REGISTER_MODEL_EVENT = 0x001B,
CSR_MESH_MCP_REGISTER_MODEL_CLIENT_EVENT = 0x001C,
  CSR_MESH_REMOVE_NETWORK_EVENT = 0x001D,
CSR_MESH_GET_DIAG_DATA_EVENT = 0x001E,
CSR_MESH_RESET_DIAG_DATA_EVENT = 0x001F,
CSR_MESH_REGISTER_SNIFFER_APP_CB_EVENT = 0x0020,
  CSR_MESH_GET_MESH_ID_EVENT = 0x0021,
CSR_MESH_GET_NET_ID_FROM_MESH_ID_EVENT = 0x0022,
CSR_MESH_ASSOCIATION_ATTENTION_EVENT = 0x0023,
CSR_MESH_BEARER_STATE_EVENT = 0x0024,
  CSR_MESH_NW_IV_UPDATED_ALREADY = 0x0025,
CSR_MESH_NW_IV_UPDATE_STARTED = 0x0026,
CSR_MESH_NW_IV_WRONG_RSP_RECEIVED = 0x0027,
CSR_MESH_NW_KEY_IV_COMPLETE_EVENT = 0x0028,
  CSR_MESH_NW_KEY_IV_FAILED_EVENT = 0x0029,
CSR_MESH_GET_CONFORMANCE_SIGNATURE_EVENT = 0x0030,
CSR_MESH_GET_NETWORK_KEY_EVENT = 0x0031,
CSR_MESH_GET_DHM_KEY_EVENT = 0x0032,
  CSR_MESH_DEVICE_ID_UPDATE_EVENT = 0x0033,
CSR_MESH_GET_DEFAULT_TTL_EVENT = 0x0034,
CSR_MESH_CONFIG_SET_MSG_PARAMS_EVENT = 0x0035,
CSR_MESH_CONFIG_GET_MSG_PARAMS_EVENT = 0x0036,
  CSR_MESH_SEQ_NUM_ROLL_BACK_EVENT = 0x0037,
CSR_MESH_MASP_DEVICE_CONF_ARGS_EVENT = 0x0038,
CSR_MESH_MASP_CLIENT_CONFIRMATION_EVENT = 0x0039,
```

```
CSR_MESH_INVALID_EVENT = 0xFFFF
}
```

CSRmesh event types.

261 csr_sched.h File Reference

Detailed Description

CSRmesh library LE bearer scheduling configuration and control functions.

261.1 Functions of csr_sched.h File Reference

```
CSRSchedResult CSRSchedSetScanDutyCycle (CsrUint16 scan_duty_cycle, CsrUint16
new_scan_slot)
```

Scan duty cycle can be configured using this API.

```
CsrUint16 CSRSchedGetScanDutyCycle (void)
```

Current scan duty cycle can be fetched using this API.

```
CSRSchedResult CSRSchedSetConfigParams (CSR_SCHED_LE_PARAMS_T *le_params)
```

Configure Mesh and Generic LE parameters.

```
CSRSchedResult CSRSchedSendUserAdv (CSR_SCHED_ADV_DATA_T *le_adv_data,
CSR_SCHED_USER_ADV_NOTIFY_CB_T callBack)
```

Transmit Non-Connectable/Connectable Application data.

```
CSRSchedResult CSRSchedGetConfigParams (CSR_SCHED_LE_PARAMS_T *le_params)
```

Get scheduler configuration parameters.

```
void CSRSchedRegisterPriorityMsgCb (CSR_SCHED_PRIORITY_MSG_CB_T cb_ptr)
```

Register callback function to handle radio busy error for priority message. The callback will be called when the radio is available for transmission.

```
CSRSchedResult CSRSchedSendPriorityMsg (CsrUint8 mesh_packet_type, CsrUint8
*payload, CsrUint8 length)
```

This function sends a CSRmesh message with priority.

```
CSRSchedResult CSRSchedHandleIncomingData (CSR_SCHED_INCOMING_DATA_EVENT_T
data_event, CsrUint8 *data, CsrUint8 length, CsrInt8 rssi)
```

Forward Mesh messages to the scheduler.

```
CSRSchedResult CSRSchedEnableListening (CsrBool enable)
```

Starts or stops LE scan operation.

```
CsrBool IsCSRSchedRunning (void)
```

Indicates about the scheduler state.

```
CSRSchedResult CSRSchedStart (void)
```

Start scheduling of LE scan and advertisement.

```
void CSRSchedNotifyGattEvent (CSR_SCHED_GATT_EVENT_T gatt_event_type,
CSR_SCHED_GATT_EVENT_DATA_T *gatt_event_data, CSR_SCHED_NOTIFY_GATT_CB_T
call_back)
```

Initialize CSRmesh core mesh stack.

```
void CsrSchedSetTxPower (CsrUint8 level)
```

Sets the Transmit Power for the Device.

262 csr_sched_types.h File Reference

Detailed Description

CSRmesh Scheduler data types.

262.1 Data Structures of csr_sched_types.h File Reference

`struct CSR_SCHED_SCAN_WINDOW_PARAM_T`

CSR Mesh Scheduling Scan Window Parameter Type.

`struct CSR_SCHED_SCAN_DUTY_PARAM_T`

CSR Mesh Scheduling Scan Duty Parameter Type.

`struct CSR_SCHED_GENERIC_LE_PARAM_T`

CSR Mesh Scheduling Generic LE Parameter Type.

`struct CSR_SCHED_MESH_TX_BUFFER_T`

CSR Mesh Sched TX queue buffer Type.

`struct CSR_SCHED_MESH_TX_PARAM_T`

CSR Mesh Scheduling Tx Parameter Type.

`struct CSR_SCHED_MESH_LE_PARAM_T`

CSR Mesh Scheduling Mesh-LE Parameter Type.

`struct CSR_SCHED_LE_PARAMS_T`

CSR Mesh Scheduling LE Parameter Type.

`struct CSR_SCHED_ADV_PARAMS_T`

CSR Mesh Scheduling LE Advertisment Parameter Type.

`struct CSR_SCHED_ADV_DATA_T`

CSR Mesh Scheduling LE Advertising Data.

`struct CSR_SCHED_GATT_EVENT_DATA_T`

CSR Mesh Scheduling GATT Event Type.

262.2 Macros of csr_sched_types.h File Reference

```
#define MAX_USER_ADV_DATA_LEN   (31)
```

Maximum User Advertising data length.

262.3 Typedefs of csr_sched_types.h File Reference

```
typedef void(* CSR_SCHED_NOTIFY_GATT_CB_T) (CsrUint16 ucid, CsrUint8 *mtl_msg,
CsrUint8 length)
```

CSR Mesh Scheduler GATT notify handler function.

```
typedef void(* CSR_SCHED_USER_ADV_NOTIFY_CB_T) (CSR_SCHED_USER_ADV_STATUS_T
status, CsrUint32 time_us)
```

CSR Mesh Application callback handler function.

```
typedef void(* CSR_SCHED_PRIORITY_MSG_CB_T) (void)
```

CSR Mesh Application callback handler function.

262.4 Enumerations of csr_sched_types.h File Reference

```
enum  CSR_SCHED_SCAN_TYPE_T { CSR_SCHED_SCAN_WINDOW_PARAM = 0x00,
CSR_SCHED_SCAN_DUTY_PARAM = 0x01
}
```

CSR Mesh Scheduling Scan Parameter Type.

```
enum  CSR_SCHED_GATT_EVENT_T { CSR_SCHED_GATT_CONNECTION_EVENT,
CSR_SCHED_GATT_STATE_CHANGE_EVENT,
CSR_SCHED_GATT_CCCD_STATE_CHANGE_EVENT
}
```

CSR Mesh Scheduling LE Event Type.

```
enum  CSR_SCHED_INCOMING_DATA_EVENT_T { CSR_SCHED_INCOMING_LE_MESH_DATA_EVENT =
1,
CSR_SCHED_INCOMING_GATT_MESH_DATA_EVENT = 2,
CSR_SCHED_SET_LE_ADV_PKT_EVENT = 3
}
```

CSR Mesh Scheduling LE Incoming data event Type.

```
enum  CSRSchedResult {
  CSR_SCHED_RESULT_SUCCESS = 0x0000,
CSR_SCHED_RESULT_INVALID_HANDLE = 0x0001,
CSR_SCHED_RESULT_UNACCEPTABLE_ADV_DURATION = 0x0002,
CSR_SCHED_RESULT_UNACCEPTABLE_ADV_INTERVAL = 0x0003,
  CSR_SCHED_RESULT_INCORRECT_SCAN_PARAM = 0x0004,
CSR_SCHED_RESULT_SCAN_NOT_STARTED = 0x0005
, CSR_SCHED_RESULT_FAILURE = 0xFFFF
}
```

CSR Mesh Scheduling operation status.

263 csr_types.h File Reference

Detailed Description

CSRmesh library data types.

264 data_client.h File Reference

Detailed Description

This files provides the prototypes of the client functions defined in the CSRmesh Data model.

264.1 Functions of data_client.h File Reference

```
CSRmeshResult DataModelClientInit (CSRMESH_MODEL_CALLBACK_T app_callback)
```

Initialises Data Model Client functionality.

```
CSRmeshResult DataStreamFlush (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8 ttl,
CSRMESH_DATA_STREAM_FLUSH_T *p_params)
```

Flushing Data: Upon receiving a DATA_STREAM_FLUSH message, the device saves the StreamSN field into the StreamSequenceNumber model state and responds with DATA_STREAM_RECEIVED with the StreamNESN field set to the value of the StreamSequenceNumber model state. The device also flushes all partially received stream data from this peer device.

```
CSRmeshResult DataStreamSend (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8 ttl,
CSRMESH_DATA_STREAM_SEND_T *p_params)
```

Sending Data: Upon receiving a DATA_STREAM_SEND message, the device first checks if the StreamSN field is the same as the StreamSequenceNumber model state. If these values are the same, the device passes the StreamOctets field up to the application for processing, and increments StreamSequenceNumber by the length of the StreamOctets field. It then responds with a DATA_STREAM_RECEIVED message with the current value of the StreamSequenceNumber. Note: The DATA_STREAM_RECEIVED message is sent even if the StreamSN received is different from the StreamSequenceNumber state. This allows missing packets to be detected and retransmitted by the sending device.

```
CSRmeshResult DataBlockSend (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8 ttl,
CSRMESH_DATA_BLOCK_SEND_T *p_params)
```

A block of data, no acknowledgement. Upon receiving a DATA_BLOCK_SEND message, the device passes the DatagramOctets field up to the application for processing.

265 data_model.h File Reference

Detailed Description

Defines CSRmesh Data Model specific data structures .

265.1 Data Structures of data_model.h File Reference

`struct CSRMESH_DATA_STREAM_FLUSH_T`

CSRmesh Data Model message types.

`struct CSRMESH_DATA_STREAM_SEND_T`

Sending Data: Upon receiving a DATA_STREAM_SEND message, the device first checks if the StreamSN field is the same as the StreamSequenceNumber model state. If these values are the same, the device passes the StreamOctets field up to the application for processing, and increments StreamSequenceNumber by the length of the StreamOctets field. It then responds with a DATA_STREAM_RECEIVED message with the current value of the StreamSequenceNumber. Note: The DATA_STREAM_RECEIVED message is sent even if the StreamSN received is different from the StreamSequenceNumber state. This allows missing packets to be detected and retransmitted by the sending device.

`struct CSRMESH_DATA_STREAM_RECEIVED_T`

Acknowledgement of data received.

`struct CSRMESH_DATA_BLOCK_SEND_T`

A block of data, no acknowledgement. Upon receiving a DATA_BLOCK_SEND message, the device passes the DatagramOctets field up to the application for processing.

266 data_server.h File Reference

Detailed Description

This file provides the prototypes of the server functions defined in the CSRmesh Data model.

266.1 Functions of data_server.h File Reference

```
CSRmeshResult DataModelInit (CsrUint8 nw_id, CsrUint16 *group_id_list,
CsrUint16 num_groups, CSRMESH_MODEL_CALLBACK_T app_callback)
```

Model Initialisation.

```
CSRmeshResult DataStreamReceived (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8
ttl, CSRMESH_DATA_STREAM_RECEIVED_T *p_params)
```

Acknowledgement of data received.

267 diagnostic_client.h File Reference

Detailed Description

This files provides the prototypes of the client functions defined in the CSRmesh Diagnostic model.

267.1 Functions of diagnostic_client.h File Reference

```
CSRmeshResult DiagnosticModelClientInit (CSRMESH_MODEL_CALLBACK_T app_callback)
```

Initialises Diagnostic Model Client functionality.

```
CSRmeshResult DiagnosticState (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8 ttl,
CSRMESH_DIAGNOSTIC_STATE_T *p_params)
```

When received this message is interpreted as to reconfigure the set of information collected. Statistics gathering can be turned on/off ? in the off mode no measurement of messages count and RSSI measurements will be made. RSSI binning can be stored, such that collection ALL messages? RSSI (MASP/MCP, irrespective of encoding) are split between a given number of bin, each of equal dimensions. Masking of individual broadcast channel can be specified, resulting in the collection of information specifically on the selected channels. A REST bit is also available. When present all the accumulated data will be cleared and all counters restarted. Note that it is possible to change various configurations without the RESET flag, this will result in the continuation of accumulation and therefore incoherent statistics.

```
CSRmeshResult DiagnosticGetStats (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8
ttl, CSRMESH_DIAGNOSTIC_GET_STATS_T *p_params)
```

This function packs the given parameters into a CSRmesh message and sends it over the network. When a response is received the application callback function is called to notify the CSRMESH_DIAGNOSTIC_STATS.

268 diagnostic_model.h File Reference

Detailed Description

Defines CSRmesh Diagnostic Model specific data structures .

268.1 Data Structures of diagnostic_model.h File Reference

`struct CSRMESH_DIAGNOSTIC_STATE_T`

CSRmesh Diagnostic Model message types.

269 diagnostic_server.h File Reference

Detailed Description

This file provides the prototypes of the server functions defined in the CSRmesh Diagnostic model.

269.1 Functions of diagnostic_server.h File Reference

```
CSRmeshResult DiagnosticModelInit (CsrUint8 nw_id, CsrUint16 *group_id_list,
CsrUint16 num_groups, CSRMESH_MODEL_CALLBACK_T app_callback)
```

Model Initialisation.

```
CSRmeshResult DiagnosticStats (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8 ttl,
CSRMESH_DIAGNOSTIC_STATS_T *p_params)
```

This function packs the given parameters into a CSRmesh message and sends it over the network.

270 extension_client.h File Reference

Detailed Description

This files provides the prototypes of the client functions defined in the CSRmesh Extension model.

270.1 Functions of extension_client.h File Reference

```
CSRmeshResult ExtensionModelClientInit (CSRMESH_MODEL_CALLBACK_T app_callback)
```

Initialises Extension Model Client functionality.

```
CSRmeshResult ExtensionRequest (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8
ttl, CSRMESH_EXTENSION_REQUEST_T *p_params)
```

Request for Extension OpCode to be approved by the whole Mesh. A device wanting to use an OpCode, makes a request to the entire Mesh Network. This message is issued to target identity 0. The device waits some time, proportional to the size of the Mesh network and only after this period, messages using these proposed OpCode are used. Device receiving this message and wanting to oppose the usage of such code will respond to the source node with a CONFLICT. In case no conflict is known and the OpCode is for a message the node is interested in implementing (through comparison with hash value), a record of the OpCode and its mapping is kept. Request messages are relayed in cases of absence of conflict. The hash function is SHA-256, padded as per SHA-256 specifications2, for which the least significant 6 bytes will be used in the message. The range parameter indicates the maximum number of OpCode reserved from the based provided in the Proposed OpCode field. The last OpCode reserved is determined through the sum of the Proposed OpCode with the range value. This range parameter varies from 0 to 127, leaving the top bit free.

```
CSRmeshResult ExtensionClientSetupOpcodeList (uint16 *opcode_list, CsrUint8
max_opcode_ranges)
```

Set Up Opcode List to be accessible by the extension model.

```
CSRmeshResult ExtensionSendMessage (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8
ttl, CsrUint8 *msg, CsrUint8 len)
```

Sends CSRmesh message with an extension opcode.

271 extension_model.h File Reference

Detailed Description

Defines CSRmesh Extension Model specific data structures .

271.1 Data Structures of extension_model.h File Reference

`struct CSRMESH_EXTENSION_REQUEST_T`

CSRmesh Extension Model message types.

`struct CSRMESH_EXTENSION_CONFLICT_T`

Response to a REQUEST - only issued if a conflict is noticed. This message indicates that the proposed OpCode is already in use within the node processing the request message. Nodes receiving conflict extension will process this message and remove the conflicting OpCode from the list of OpCodes to handle. All conflict messages are relayed, processed or not. If a node receiving a REQUEST is able to match the hash of the provider previously assigned to an existing OpCode, but different to the proposed one, it responds with a CONFLICT with a reason combining the top bit with the previously associated range (0x80 | <old range>=""">). In such cases, the previously used OpCode (start of range) will be placed in the ProposedOpCode. Nodes receiving this conflict message with the top bit raised, will discard the initially proposed OpCode and replace it with the proposed code supplied in the conflict message.

272 extension_server.h File Reference

Detailed Description

This file provides the prototypes of the server functions defined in the CSRmesh Extension model.

272.1 Functions of extension_server.h File Reference

```
CSRmeshResult ExtensionModelInit (CsrUint8 nw_id, CsrUint16 *group_id_list,
CsrUint16 num_groups, CSRMESH_MODEL_CALLBACK_T app_callback)
```

Model Initialisation.

```
CSRmeshResult ExtensionConflict (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8
ttl, CSRMESH_EXTENSION_CONFLICT_T *p_params)
```

Response to a REQUEST - only issued if a conflict is noticed. This message indicates that the proposed OpCode is already in use within the node processing the request message. Nodes receiving conflict extension will process this message and remove the conflicting OpCode from the list of OpCodes to handle. All conflict messages are relayed, processed or not. If a node receiving a REQUEST is able to match the hash of the provider previously assigned to an existing OpCode, but different to the proposed one, it responds with a CONFLICT with a reason combining the top bit with the previously associated range (0x80 | <old range>=""). In such cases, the previously used OpCode (start of range) will be placed in the ProposedOpCode. Nodes receiving this conflict message with the top bit raised, will discard the initially proposed OpCode and replace it with the proposed code supplied in the conflict message.

```
bool ExtensionVerifyOpcodeConflictWithMesh (CsrUint16 opcode_start, CsrUint16
opcode_end)
```

Verify the opcodes passed Conflict with Mesh Opcodes.

```
CSRmeshResult ExtensionServerSetupOpcodeList (uint16 *opcode_list, CsrUint8
max_opcode_ranges)
```

Set Up Opcode List to be accessible by the extension model.

273 firmware_client.h File Reference

Detailed Description

This files provides the prototypes of the client functions defined in the CSRmesh Firmware model.

273.1 Functions of firmware_client.h File Reference

```
CSRmeshResult FirmwareModelClientInit (CSRMESH_MODEL_CALLBACK_T app_callback)
```

Initialises Firmware Model Client functionality.

```
CSRmeshResult FirmwareGetVersion (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8
ttl, CSRMESH_FIRMWARE_GET_VERSION_T *p_params)
```

Get firmwre verison: Upon receiving a FIRMWARE_GET_VERSION the device reponds with a FIRMWARE_VERSION message containing the current firmware version.

```
CSRmeshResult FirmwareUpdateRequired (CsrUint8 nw_id, CsrUint16 dest_id,
CsrUint8 ttl, CSRMESH_FIRMWARE_UPDATE_REQUIRED_T *p_params)
```

Requesting a firmware update. Upon receiving this message, the device moves to a state where it is ready for receiving a firmware update.

274 firmware_model.h File Reference

Detailed Description

Defines CSRmesh Firmware Model specific data structures .

274.1 Data Structures of firmware_model.h File Reference

`struct CSRMESH_FIRMWARE_GET_VERSION_T`

CSRmesh Firmware Model message types.

`struct CSRMESH_FIRMWARE_VERSION_T`

Firmware version information.

`struct CSRMESH_FIRMWARE_UPDATE_REQUIRED_T`

Requesting a firmware update. Upon receiving this message, the device moves to a state where it is ready for receiving a firmware update.

`struct CSRMESH_FIRMWARE_UPDATE_ACKNOWLEDGED_T`

Acknowledgement message to the firmware update request.

275 firmware_server.h File Reference

Detailed Description

This file provides the prototypes of the server functions defined in the CSRmesh Firmware model.

275.1 Functions of firmware_server.h File Reference

```
CSRmeshResult FirmwareModelInit (CsrUint8 nw_id, CsrUint16 *group_id_list,
CsrUint16 num_groups, CSRMESH_MODEL_CALLBACK_T app_callback)
```

Model Initialisation.

```
CSRmeshResult FirmwareVersion (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8 ttl,
CSRMESH_FIRMWARE_VERSION_T *p_params)
```

Firmware version information.

```
CSRmeshResult FirmwareUpdateAcknowledged (CsrUint8 nw_id, CsrUint16 dest_id,
CsrUint8 ttl, CSRMESH_FIRMWARE_UPDATE_ACKNOWLEDGED_T *p_params)
```

Acknowledgement message to the firmware update request.

276 group_client.h File Reference

Detailed Description

This files provides the prototypes of the client functions defined in the CSRmesh Group model.

276.1 Functions of group_client.h File Reference

```
CSRmeshResult GroupModelClientInit (CSRMESH_MODEL_CALLBACK_T app_callback)
```

Initialises Group Model Client functionality.

```
CSRmeshResult GroupGetNumberOfModelGroupids (CsrUint8 nw_id, CsrUint16 dest_id,
CsrUint8 ttl, CSRMESH_GROUP_GET_NUMBER_OF_MODEL_GROUPIDS_T *p_params)
```

Getting Number of Group IDs: Upon receiving a GROUP_GET_NUMBER_OF_MODEL_GROUPS message, where the destination address is the DeviceID of this device, the device responds with a GROUP_NUMBER_OF_MODEL_GROUPS message with the number of Group IDs that the given model supports on this device.

```
CSRmeshResult GroupSetModelGroupid (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8
ttl, CSRMESH_GROUP_SET_MODEL_GROUPID_T *p_params)
```

Setting Model Group ID: Upon receiving a GROUP_SET_MODEL_GROUPID message, where the destination address is the DeviceID of this device, the device saves the Instance and GroupID fields into the appropriate state value determined by the Model and GroupIndex fields. It then responds with a GROUP_MODEL_GROUPID message with the current state information held for the given model and the GroupIndex values.

```
CSRmeshResult GroupGetModelGroupid (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8
ttl, CSRMESH_GROUP_GET_MODEL_GROUPID_T *p_params)
```

Getting Model Group ID: Upon receiving a GROUP_GET_MODEL_GROUPID message, where the destination address is the DeviceID of this device, the device responds with a GROUP_MODEL_GROUPID message with the current state information held for the given Model and GroupIndex values.

277 group_model.h File Reference

Detailed Description

Defines CSRmesh Group Model specific data structures .

277.1 Data Structures of group_model.h File Reference

`struct CSRMESH_GROUP_GET_NUMBER_OF_MODEL_GROUPIDS_T`

CSRmesh Group Model message types.

`struct CSRMESH_GROUP_NUMBER_OF_MODEL_GROUPIDS_T`

Get number of groups supported by the model.

`struct CSRMESH_GROUP_SET_MODEL_GROUPID_T`

Setting Model Group ID: Upon receiving a GROUP_SET_MODEL_GROUPID message, where the destination address is the DeviceID of this device, the device saves the Instance and GroupID fields into the appropriate state value determined by the Model and GroupIndex fields. It then responds with a GROUP_MODEL_GROUPID message with the current state information held for the given model and the GroupIndex values.

`struct CSRMESH_GROUP_GET_MODEL_GROUPID_T`

Getting Model Group ID: Upon receiving a GROUP_GET_MODEL_GROUPID message, where the destination address is the DeviceID of this device, the device responds with a GROUP_MODEL_GROUPID message with the current state information held for the given Model and GroupIndex values.

`struct CSRMESH_GROUP_MODEL_GROUPID_T`

GroupID of a model.

278 group_server.h File Reference

Detailed Description

This file provides the prototypes of the server functions defined in the CSRmesh Group model.

278.1 Functions of group_server.h File Reference

```
CSRmeshResult GroupModelInit (CsrUint8 nw_id, CsrUint16 *group_id_list,
CsrUint16 num_groups, CSRMESH_MODEL_CALLBACK_T app_callback)
```

Model Initialisation.

```
CSRmeshResult GroupNumberOfModelGroupids (CsrUint8 nw_id, CsrUint16 dest_id,
CsrUint8 ttl, CSRMESH_GROUP_NUMBER_OF_MODEL_GROUPIDS_T *p_params)
```

Get number of groups supported by the model.

```
CSRmeshResult GroupModelGroupid (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8
ttl, CSRMESH_GROUP_MODEL_GROUPID_T *p_params)
```

GroupID of a model.

279 largeobjecttransfer_client.h File Reference

Detailed Description

This files provides the prototypes of the client functions defined in the CSRmesh LargeObjectTransfer model.

279.1 Functions of largeobjecttransfer_client.h File Reference

```
CSRmeshResult LargeObjectTransferModelClientInit (CSRMESH_MODEL_CALLBACK_T
app_callback)
```

Initialises LargeObjectTransfer Model Client functionality.

```
CSRmeshResult LargeObjectTransferAnnounce (CsrUint8 nw_id, CsrUint16 dest_id,
CsrUint8 ttl, CSRMESH_LARGEOBJECTTRANSFER_ANNOUNCE_T *p_params)
```

A node wanting to provide a large object to neighbouring Mesh Nodes issues an ANNOUNCE with the associated content type. This message will have TTL=0, thus will only be answered by its immediate neighbours. The ANNOUNCE has the total size of the packet to be issued. The format and encoding of the large object is subject to the provided type and is out of scope of this document. The destination ID can either be 0, a group or a specific Device ID. In case the destination ID is not zero, only members of the group (associated with the LOT model) or the device with the specified Device ID responds with the intent to download the object for their own consumption. Every other node either ignores or accepts the offer for the purpose of relaying the packet.

280 largeobjecttransfer_model.h File Reference

Detailed Description

Defines CSRmesh LargeObjectTransfer Model specific data structures .

280.1 Data Structures of largeobjecttransfer_model.h File Reference

`struct CSRMESH_LARGEOBJECTTRANSFER_ANNOUNCE_T`

CSRmesh LargeObjectTransfer Model message types.

`struct CSRMESH_LARGEOBJECTTRANSFER_INTEREST_T`

In case a node is ready to receive the proposed object, it responds with this message. The intended behaviour of the Large Object Transfer is to allow a Peer-to-Peer connection between the consumer and the producer. The consumer uses a ServiceID, part of which is randomly selected. The top 64 bits are 0x1122334455667788, the least significant 63 bits are randomly selected by the consumer node. The most significant bit of the least significant 64 bits is an encoding of the intent to relay the received data. Once this message has been issued, the consumer node starts advertising a connectable service with the 128-bits service composed through the concatenation of the fixed 64 bits and the randomly selected 63 bits. The duration of the advertisement is an implementation decision.

281 largeobjecttransfer_server.h File Reference

Detailed Description

This file provides the prototypes of the server functions defined in the CSRmesh LargeObjectTransfer model.

281.1 Functions of largeobjecttransfer_server.h File Reference

```
CSRmeshResult LargeObjectTransferModelInit (CsrUint8 nw_id, CsrUint16
*group_id_list, CsrUint16 num_groups, CSRMESH_MODEL_CALLBACK_T app_callback)
```

Model Initialisation.

```
CSRmeshResult LargeObjectTransferInterest (CsrUint8 nw_id, CsrUint16 dest_id,
CsrUint8 ttl, CSRMESH_LARGEOBJECTTRANSFER_INTEREST_T *p_params)
```

In case a node is ready to receive the proposed object, it responds with this message. The intended behaviour of the Large Object Transfer is to allow a Peer-to-Peer connection between the consumer and the producer. The consumer uses a ServiceID, part of which is randomly selected. The top 64 bits are 0x1122334455667788, the least significant 63 bits are randomly selected by the consumer node. The most significant bit of the least significant 64 bits is an encoding of the intent to relay the received data. Once this message has been issued, the consumer node starts advertising a connectable service with the 128-bits service composed through the concatenation of the fixed 64 bits and the randomly selected 63 bits. The duration of the advertisement is an implementation decision.

282 light_client.h File Reference

Detailed Description

This files provides the prototypes of the client functions defined in the CSRmesh Light model.

282.1 Functions of light_client.h File Reference

```
CSRmeshResult LightModelClientInit (CSRMESH_MODEL_CALLBACK_T app_callback)
```

Initialises Light Model Client functionality.

```
CSRmeshResult LightSetLevel (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8 ttl,
CSRMESH_LIGHT_SET_LEVEL_T *p_params, bool request_ack)
```

Setting Light Level: Upon receiving a LIGHT_SET_LEVEL_NO_ACK message, the device saves the Level field into the CurrentLevel model state. LevelSDState should be set to Idle. If ACK is requested, the device should respond with a LIGHT_STATE message.

```
CSRmeshResult LightSetRgb (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8 ttl,
CSRMESH_LIGHT_SET_RGB_T *p_params, bool request_ack)
```

Setting Light Colour: Upon receiving a LIGHT_SET_RGB_NO_ACK message, the device saves the Level, Red, Green, and Blue fields into the TargetLevel, TargetRed, TargetGreen, and TargetBlue variables respectively. LevelSDState should be set to Attacking. If the Duration field is zero, then the device saves the Level, Red, Green, and Blue fields into the CurrentLevel, CurrentRed, CurrentGreen and CurrentBlue variables, and sets the DeltaLevel, DeltaRed, DeltaGreen, and DeltaBlue variables to zero. If the Duration field is greater than zero, then the device calculates the DeltaLevel, DeltaRed, DeltaGreen, and DeltaBlue levels from the differences between the Current values and the Target values divided by the Duration field, so that over Duration seconds, the CurrentLevel, CurrentRed, CurrentGreen, and CurrentBlue variables are changed smoothly to the TargetLevel, TargetRed, TargetGreen and TargetBlue values. If ACK is requested, the device responds with a LIGHT_STATE message.

```
CSRmeshResult LightSetPowerLevel (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8
ttl, CSRMESH_LIGHT_SET_POWER_LEVEL_T *p_params, bool request_ack)
```

Setting Light Power and Light Level: Upon receiving a LIGHT_SET_POWER_LEVEL_NO_ACK message, the device sets the current PowerState to the Power field, the TargetLevel variable to the Level field, the DeltaLevel to the difference between TargetLevel and CurrentLevel divided by the LevelDuration field, saves the Sustain and Decay fields into the LevelSustain and LevelDecay variables, and sets LevelSDState to the Attacking state. If ACK is requested, the device should respond with a LIGHT_STATE message.

```
CSRmeshResult LightSetColorTemp (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8
ttl, CSRMESH_LIGHT_SET_COLOR_TEMP_T *p_params)
```

Setting Light Colour Temperature: Upon receiving a LIGHT_SET_COLOR_TEMP message, the device saves the ColorTemperature field into the TargetColorTemperature state variable. If the TempDuration field is zero, the CurrentColorTemperature variable is set to TargetColorTemperature and DeltaColorTemperature is set to zero. If the TempDuration field is greater than zero, then the device calculates the difference between TargetColorTemperature and CurrentColorTemperature, over the TempDuration field and store this into a DeltaColorTemperature state variable, so that over TempDuration seconds, CurrentColorTemperature changes smoothly to TargetColorTemperature. The device then responds with a LIGHT_STATE message.

```
CSRmeshResult LightGetState (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8 ttl,
CSRMESH_LIGHT_GET_STATE_T *p_params)
```

Getting Light State: Upon receiving a LIGHT_GET_STATE message, the device responds with a LIGHT_STATE message.

```
CSRmeshResult LightSetWhite (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8 ttl,
CSRMESH_LIGHT_SET_WHITE_T *p_params, bool request_ack)
```

Setting Light White level.

```
CSRmeshResult LightGetWhite (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8 ttl,
CSRMESH_LIGHT_GET_WHITE_T *p_params)
```

Setting Light White level.

283 light_model.h File Reference

Detailed Description

Defines CSRmesh Light Model specific data structures .

283.1 Data Structures of light_model.h File Reference

`struct CSRMESH_LIGHT_SET_LEVEL_T`

CSRmesh Light Model message types.

`struct CSRMESH_LIGHT_SET_RGB_T`

Setting Light Colour: Upon receiving a LIGHT_SET_RGB_NO_ACK message, the device saves the Level, Red, Green, and Blue fields into the TargetLevel, TargetRed, TargetGreen, and TargetBlue variables respectively. LevelSDState should be set to Attacking. If the Duration field is zero, then the device saves the Level, Red, Green, and Blue fields into the CurrentLevel, CurrentRed, CurrentGreen and CurrentBlue variables, and sets the DeltaLevel, DeltaRed, DeltaGreen, and DeltaBlue variables to zero. If the Duration field is greater than zero, then the device calculates the DeltaLevel, DeltaRed, DeltaGreen, and DeltaBlue levels from the differences between the Current values and the Target values divided by the Duration field, so that over Duration seconds, the CurrentLevel, CurrentRed, CurrentGreen, and CurrentBlue variables are changed smoothly to the TargetLevel, TargetRed, TargetGreen and TargetBlue values. If ACK is requested, the device responds with a LIGHT_STATE message.

`struct CSRMESH_LIGHT_SET_POWER_LEVEL_T`

Setting Light Power and Light Level: Upon receiving a LIGHT_SET_POWER_LEVEL_NO_ACK message, the device sets the current PowerState to the Power field, the TargetLevel variable to the Level field, the DeltaLevel to the difference between TargetLevel and CurrentLevel divided by the LevelDuration field, saves the Sustain and Decay fields into the LevelSustain and LevelDecay variables, and sets LevelSDState to the Attacking state. If ACK is requested, the device should respond with a LIGHT_STATE message.

`struct CSRMESH_LIGHT_SET_COLOR_TEMP_T`

Setting Light Colour Temperature: Upon receiving a LIGHT_SET_COLOR_TEMP message, the device saves the ColorTemperature field into the TargetColorTemperature state variable. If the TempDuration field is zero, the CurrentColorTemperature variable is set to TargetColorTemperature and DeltaColorTemperature is set to zero. If the TempDuration field is greater than zero, then the device calculates the difference between TargetColorTemperature and CurrentColorTemperature, over the TempDuration field and store this into a DeltaColorTemperature state variable, so that over TempDuration seconds, CurrentColorTemperature changes smoothly to TargetColorTemperature. The device then responds with a LIGHT_STATE message.

`struct CSRMESH_LIGHT_GET_STATE_T`

Getting Light State: Upon receiving a LIGHT_GET_STATE message, the device responds with a LIGHT_STATE message.

`struct CSRMESH_LIGHT_STATE_T`

Current light state.

`struct CSRMESH_LIGHT_SET_WHITE_T`

Setting Light White level.

`struct CSRMESH_LIGHT_GET_WHITE_T`

Setting Light White level.

`struct CSRMESH_LIGHT_WHITE_T`

Setting Light White level.

284 light_server.h File Reference

Detailed Description

This file provides the prototypes of the server functions defined in the CSRmesh Light model.

284.1 Functions of light_server.h File Reference

```
CSRmeshResult LightModelInit (CsrUint8 nw_id, CsrUint16 *group_id_list,
CsrUint16 num_groups, CSRMESH_MODEL_CALLBACK_T app_callback)
```

Model Initialisation.

```
CSRmeshResult LightState (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8 ttl,
CSRMESH_LIGHT_STATE_T *p_params, bool request_ack)
```

Current light state.

```
CSRmeshResult LightWhite (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8 ttl,
CSRMESH_LIGHT_WHITE_T *p_params, bool request_ack)
```

Setting Light White level.

285 nvm_access.h File Reference

Detailed Description

Header definitions for NVM usage.

285.1 Data Structures of nvm_access.h File Reference

`struct nvm_cs_header_t`

NVM key descriptor.

`struct nvm_versioned_header_t`

Versioned header information.

285.2 Macros of nvm_access.h File Reference

```
#define NVM_PAD_ROUND_UP_TO_16(x)    ((sizeof(nvm_versioned_header_t) + x + 15)
& ~0x0f)
```

Macros to round a number up to the next 16 or 32 after including the Versioned Header size.

285.3 Typedefs of nvm_access.h File Reference

```
typedef bool(* migrate_handler_t) (uint16 version_in_nvm, uint16
nvm_app_data_offset, uint16 nvm_data_length)
```

Function to convert data in the NVM to a newer version.

285.4 Enumerations of nvm_access.h File Reference

`enum nvm_data_id_t`

NVM key identifiers.

285.5 Functions of nvm_access.h File Reference

```
bool Nvm_ValidateVersionedHeader (uint16 *nvm_offset, nvm_data_id_t key_id,
uint16 data_len, uint16 version, migrate_handler_t migrate_handler)
```

Verifies that the data block in NVM matches the expected identifier.

```
void Nvm_WriteVersionedHeader (uint16 *nvm_offset, nvm_data_id_t key_id, uint16
data_len, uint16 version)
```

Writes a versioned header to NVM.

```
void Nvm_Init (store_id_t id, uint16 nvm_sanity, uint16 *nvm_offset)
```

Initialises the NVM.

```
void AppNvmReady (bool nvm_fresh, uint16 nvm_offset)
```

Called when the NVM initialisation is done.

```
void Nvm_Read (uint16 *buffer, uint16 length, uint16 offset)
```

Read words from the NVM store after preparing the NVM to be readable.

```
void Nvm_Write (uint16 *buffer, uint16 length, uint16 offset)
```

Write words to the NVM store after preparing the NVM to be writable.

```
void NvmProcessEvent (msg_t *msg)
```

Handles the user store messages.

```
sys_status Nvm_SetMemType (memory_type_t type)
```

Sets the memory device type for the user store.

286 ping_client.h File Reference

Detailed Description

This files provides the prototypes of the client functions defined in the CSRmesh Ping model.

286.1 Functions of ping_client.h File Reference

```
CSRmeshResult PingModelClientInit (CSRMESH_MODEL_CALLBACK_T app_callback)
```

Initialises Ping Model Client functionality.

```
CSRmeshResult PingRequest (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8 ttl,
CSRMESH_PING_REQUEST_T *p_params)
```

Ping Request: Upon receiving a PING_REQUEST message, the device responds with a PING_RESPONSE message with the TTLAtRx field set to the TTL value from the PING_REQUEST message, and the RSSIAtRx field set to the RSSI value of the PING_REQUEST message. If the bearer used to receive the PING_REQUEST message does not have an RSSI value, then the value 0x00 is used.

287 ping_model.h File Reference

Detailed Description

Defines CSRmesh Ping Model specific data structures .

287.1 Data Structures of ping_model.h File Reference

`struct CSRMESH_PING_REQUEST_T`

CSRmesh Ping Model message types.

`struct CSRMESH_PING_RESPONSE_T`

Ping response.

288 ping_server.h File Reference

Detailed Description

This file provides the prototypes of the server functions defined in the CSRmesh Ping model.

288.1 Functions of ping_server.h File Reference

```
CSRmeshResult PingModelInit (CsrUint8 nw_id, CsrUint16 *group_id_list,
CsrUint16 num_groups, CSRMESH_MODEL_CALLBACK_T app_callback)
```

Model Initialisation.

```
CSRmeshResult PingResponse (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8 ttl,
CSRMESH_PING_RESPONSE_T *p_params)
```

Ping response.

289 power_client.h File Reference

Detailed Description

This files provides the prototypes of the client functions defined in the CSRmesh Power model.

289.1 Functions of power_client.h File Reference

```
CSRmeshResult PowerModelClientInit (CSRMESH_MODEL_CALLBACK_T app_callback)
```

Initialises Power Model Client functionality.

```
CSRmeshResult PowerSetState (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8 ttl,
CSRMESH_POWER_SET_STATE_T *p_params, bool request_ack)
```

Setting Power State: Upon receiving a POWER_SET_STATE_NO_ACK message, the device sets the PowerState state value to the PowerState field. It then responds with a POWER_STATE message with the current state information.

```
CSRmeshResult PowerToggleState (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8
ttl, CSRMESH_POWER_TOGGLE_STATE_T *p_params, bool request_ack)
```

Toggling Power State: Upon receiving a POWER_Toggle_STATE_NO_ACK message, the device sets the PowerState state value as defined: 1.If the current PowerState is 0x00, Off, then PowerState should be set to 0x01, On. 2.If the current PowerState is 0x01, On, then PowerState should be set to 0x00, Off. 3.If the current PowerState is 0x02, Standby, then PowerState should be set to 0x03, OnFromStandby. 4.If the current PowerState is 0x03, OnFromStandby, then PowerState should be set to 0x02, Standby. Then the device responds with a POWER_STATE message with the current state information.

```
CSRmeshResult PowerGetState (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8 ttl,
CSRMESH_POWER_GET_STATE_T *p_params)
```

Getting Power State: Upon receiving a POWER_GET_STATE message, the device responds with a POWER_STATE message with the current state information.

290 power_model.h File Reference

Detailed Description

Defines CSRmesh Power Model specific data structures .

290.1 Data Structures of power_model.h File Reference

`struct CSRMESH_POWER_SET_STATE_T`

CSRmesh Power Model message types.

`struct CSRMESH_POWER_TOGGLE_STATE_T`

Toggling Power State: Upon receiving a POWER_Toggle_STATE_NO_ACK message, the device sets the PowerState state value as defined: 1.If the current PowerState is 0x00, Off, then PowerState should be set to 0x01, On. 2.If the current PowerState is 0x01, On, then PowerState should be set to 0x00, Off. 3.If the current PowerState is 0x02, Standby, then PowerState should be set to 0x03, OnFromStandby. 4.If the current PowerState is 0x03, OnFromStandby, then PowerState should be set to 0x02, Standby. Then the device responds with a POWER_STATE message with the current state information.

`struct CSRMESH_POWER_GET_STATE_T`

Getting Power State: Upon receiving a POWER_GET_STATE message, the device responds with a POWER_STATE message with the current state information.

`struct CSRMESH_POWER_STATE_T`

Current power state.

291 power_server.h File Reference

Detailed Description

This file provides the prototypes of the server functions defined in the CSRmesh Power model.

291.1 Functions of power_server.h File Reference

```
CSRmeshResult PowerModelInit (CsrUint8 nw_id, CsrUint16 *group_id_list,
CsrUint16 num_groups, CSRMESH_MODEL_CALLBACK_T app_callback)
```

Model Initialisation.

```
CSRmeshResult PowerState (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8 ttl,
CSRMESH_POWER_STATE_T *p_params, bool request_ack)
```

Current power state.

292 sensor_client.h File Reference

Detailed Description

This files provides the prototypes of the client functions defined in the CSRmesh Sensor model.

292.1 Functions of sensor_client.h File Reference

```
CSRmeshResult SensorModelClientInit (CSRMESH_MODEL_CALLBACK_T app_callback)
```

Initialises Sensor Model Client functionality.

```
CSRmeshResult SensorGetTypes (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8 ttl,
CSRMESH_SENSOR_GET_TYPES_T *p_params)
```

Upon receiving a SENSOR_GET_TYPES message, the device responds with a SENSOR_TYPES message with the list of supported types greater than or equal to the FirstType field. If the device does not support any types greater than or equal to the FirstType field, then it sends a SENSOR_TYPES message with zero-length Types field.

```
CSRmeshResult SensorSetState (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8 ttl,
CSRMESH_SENSOR_SET_STATE_T *p_params)
```

Setting Sensor State: Upon receiving a SENSOR_SET_STATE message, where the destination address is the device ID of this device and the Type field is a supported sensor type, the device saves the RxDutyCycle field and responds with a SENSOR_STATE message with the current state information of the sensor type. If the Type field is not a supported sensor type, the device ignores the message.

```
CSRmeshResult SensorGetState (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8 ttl,
CSRMESH_SENSOR_GET_STATE_T *p_params)
```

Getting Sensor State: Upon receiving a SENSOR_GET_STATE message, where the destination address is the deviceID of this device and the Type field is a supported sensor type, the device shall respond with a SENSOR_STATE message with the current state information of the sensor type. Upon receiving a SENSOR_GET_STATE message, where the destination address is the device ID of this device but the Type field is not a supported sensor type, the device shall ignore the message.

```
CSRmeshResult SensorWriteValue (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8
ttl, CSRMESH_SENSOR_WRITE_VALUE_T *p_params, bool request_ack)
```

Writing Sensor Value: Upon receiving a SENSOR_WRITE_VALUE message, where the Type field is a supported sensor type, the device saves the value into the current value of the sensor type on this device and responds with a SENSOR_VALUE message with the current value of this sensor type.

```
CSRmeshResult SensorReadValue (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8 ttl,
CSRMESH_SENSOR_READ_VALUE_T *p_params)
```

Getting Sensor Value: Upon receiving a SENSOR_READ_VALUE message, where the Type field is a supported sensor type, the device responds with a SENSOR_VALUE message with the value of the sensor type. Proxy Behaviour: Upon receiving a SENSOR_GET_STATE where the destination of the message and the sensor type correspond to a previously received SENSOR_BROADCAST_VALUE or SENSOR_BROADCAST_NEW message, the device responds with a SENSOR_VALUE message with the remembered values.

293 sensor_model.h File Reference

Detailed Description

Defines CSRmesh Sensor Model specific data structures .

293.1 Data Structures of sensor_model.h File Reference

`struct CSRMESH_SENSOR_GET_TYPES_T`

CSRmesh Sensor Model message types.

`struct CSRMESH_SENSOR_TYPES_T`

Sensor types.

`struct CSRMESH_SENSOR_SET_STATE_T`

Setting Sensor State: Upon receiving a SENSOR_SET_STATE message, where the destination address is the device ID of this device and the Type field is a supported sensor type, the device saves the RxDutyCycle field and responds with a SENSOR_STATE message with the current state information of the sensor type. If the Type field is not a supported sensor type, the device ignores the message.

`struct CSRMESH_SENSOR_GET_STATE_T`

Getting Sensor State: Upon receiving a SENSOR_GET_STATE message, where the destination address is the deviceID of this device and the Type field is a supported sensor type, the device shall respond with a SENSOR_STATE message with the current state information of the sensor type. Upon receiving a SENSOR_GET_STATE message, where the destination address is the device ID of this device but the Type field is not a supported sensor type, the device shall ignore the message.

`struct CSRMESH_SENSOR_STATE_T`

Current sensor state.

`struct CSRMESH_SENSOR_WRITE_VALUE_T`

Writing Sensor Value: Upon receiving a SENSOR_WRITE_VALUE message, where the Type field is a supported sensor type, the device saves the value into the current value of the sensor type on this device and responds with a SENSOR_VALUE message with the current value of this sensor type.

`struct CSRMESH_SENSOR_READ_VALUE_T`

Getting Sensor Value: Upon receiving a SENSOR_READ_VALUE message, where the Type field is a supported sensor type, the device responds with a SENSOR_VALUE message with the value of the sensor type. Proxy Behaviour: Upon receiving a SENSOR_GET_STATE where the destination of the message and the sensor type correspond to a previously received SENSOR_BROADCAST_VALUE or SENSOR_BROADCAST_NEW message, the device responds with a SENSOR_VALUE message with the remembered values.

`struct CSRMESH_SENSOR_VALUE_T`

Current sensor value.

`struct CSRMESH_SENSOR_MISSING_T`

Sensor data is missing. Proxy Behaviour: Upon receiving a SENSOR_MISSING message, the proxy determines if it has remembered this type and value and then writes that type and value to the device that sent the message.

294 sensor_server.h File Reference

Detailed Description

This file provides the prototypes of the server functions defined in the CSRmesh Sensor model.

294.1 Functions of sensor_server.h File Reference

```
CSRmeshResult SensorModelInit (CsrUint8 nw_id, CsrUint16 *group_id_list,
CsrUint16 num_groups, CSRMESH_MODEL_CALLBACK_T app_callback)
```

Model Initialisation.

```
CSRmeshResult SensorTypes (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8 ttl,
CSRMESH_SENSOR_TYPES_T *p_params)
```

Sensor types.

```
CSRmeshResult SensorState (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8 ttl,
CSRMESH_SENSOR_STATE_T *p_params)
```

Current sensor state.

```
CSRmeshResult SensorValue (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8 ttl,
CSRMESH_SENSOR_VALUE_T *p_params)
```

Current sensor value.

```
CSRmeshResult SensorMissing (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8 ttl,
CSRMESH_SENSOR_MISSING_T *p_params)
```

Sensor data is missing. Proxy Behaviour: Upon receiving a SENSOR_MISSING message, the proxy determines if it has remembered this type and value and then writes that type and value to the device that sent the message.

295 sensor_types.h File Reference

Detailed Description

Defines the types used in the CSRmesh Sensor and actuator Model.

295.1 Typedefs of sensor_types.h File Reference

```
typedef uint16 SENSOR_FORMAT_TEMPERATURE_T
```

CSRmesh Temperature format.

295.2 Enumerations of sensor_types.h File Reference

```
enum  sensor_type_t {
  sensor_type_invalid = 0,
sensor_type_internal_air_temperature = 1,
sensor_type_external_air_temperature = 2,
sensor_type_desired_air_temperature = 3,
  sensor_type_internal_humidity = 4,
sensor_type_external_humidity = 5,
sensor_type_external_dewpoint = 6,
sensor_type_internal_door = 7,
  sensor_type_external_door = 8,
sensor_type_internal_window = 9,
sensor_type_external_window = 10,
sensor_type_solar_energy = 11,
  sensor_type_number_of_activations = 12,
sensor_type_fridge_temperature = 13,
sensor_type_desired_fridge_temperature = 14,
sensor_type_freezer_temperature = 15,
  sensor_type_desired_freezer_temperature = 16,
sensor_type_oven_temperature = 17,
sensor_type_desired_oven_temperature = 18,
sensor_type_seat_occupied = 19,
  sensor_type_washing_machine_state = 20,
sensor_type_dish_washer_state = 21,
sensor_type_clothes_dryer_state = 22,
sensor_type_toaster_state = 23,
  sensor_type_carbon_dioxide = 24,
sensor_type_carbon_monoxide = 25,
sensor_type_smoke = 26,
sensor_type_water_level = 27,
  sensor_type_hot_water_temperature = 28,
sensor_type_cold_water_temperature = 29,
sensor_type_desired_water_temperature = 30,
sensor_type_cooker_hob_back_left_state = 31,
  sensor_type_desired_cooker_hob_back_left_state = 32,
sensor_type_cooker_hob_front_left_state = 33,
sensor_type_desired_cooker_hob_front_left_state = 34,
sensor_type_cooker_hob_back_middle_state = 35,
  sensor_type_desired_cooker_hob_back_middle_state = 36,
sensor_type_cooker_hob_front_middle_state = 37,
sensor_type_desired_cooker_hob_front_middle_state = 38,
sensor_type_cooker_hob_back_right_state = 39,
  sensor_type_desired_cooker_hob_back_right_state = 40,
sensor_type_cooker_hob_front_right_state = 41,
sensor_type_desired_cooker_hob_front_right_state = 42,
sensor_type_desired_wakeup_alarm_time = 43,
  sensor_type_desired_second_wakeup_alarm_time = 44,
sensor_type_passive_infrared_state = 45,
sensor_type_water_flowing = 46,
sensor_type_desired_water_flow = 47,
```

```
  sensor_type_audio_level = 48,
sensor_type_desired_audio_level = 49,
sensor_type_fan_speed = 50,
sensor_type_desired_fan_speed = 51,
  sensor_type_wind_speed = 52,
sensor_type_wind_speed_gust = 53,
sensor_type_wind_direction = 54,
sensor_type_wind_direction_gust = 55,
  sensor_type_rain_fall_last_hour = 56,
sensor_type_rain_fall_today = 57,
sensor_type_barometric_pressure = 58,
sensor_type_soil_temperature = 59,
  sensor_type_soil_moisture = 60,
sensor_type_window_cover_position = 61,
sensor_type_desired_window_cover_position = 62
}
```

CSRmesh Sensor Type.

296 Switch_client.h File Reference

Detailed Description

This files provides the prototypes of the client functions defined in the CSRmesh Switch model.

296.1 Functions of Switch_client.h File Reference

```
CSRmeshResult SwitchModelClientInit (CSRMESH_MODEL_CALLBACK_T app_callback)
```

Initialises Switch Model Client functionality.

297 switch_model.h File Reference

Detailed Description

Defines CSRmesh Switch Model specific data structures .

298 switch_server.h File Reference

Detailed Description

This file provides the prototypes of the server functions defined in the CSRmesh Switch model.

298.1 Functions of switch_server.h File Reference

```
CSRmeshResult SwitchModelInit (CsrUint8 nw_id, CsrUint16 *group_id_list,
CsrUint16 num_groups, CSRMESH_MODEL_CALLBACK_T app_callback)
```

Model Initialisation.

299 time_client.h File Reference

Detailed Description

This files provides the prototypes of the client functions defined in the CSRmesh Time model.

299.1 Functions of time_client.h File Reference

```
CSRmeshResult TimeModelClientInit (CSRMESH_MODEL_CALLBACK_T app_callback)
```

Initialises Time Model Client functionality.

```
CSRmeshResult TimeSetState (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8 ttl,
CSRMESH_TIME_SET_STATE_T *p_params)
```

Setting Time Broadcast Interval: Upon receiving a TIME_SET_STATE message, the device saves the TimeInterval field into the appropriate state value. It then responds with a TIME_STATE message with the current state information.

```
CSRmeshResult TimeGetState (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8 ttl,
CSRMESH_TIME_GET_STATE_T *p_params)
```

Getting Time Broadcast Interval: Upon receiving a TIME_GET_STATE message, the device responds with a TIME_STATE message with the current state information.

300 time_model.h File Reference

Detailed Description

Defines CSRmesh Time Model specific data structures .

300.1 Data Structures of time_model.h File Reference

`struct CSRMESH_TIME_SET_STATE_T`

CSRmesh Time Model message types.

`struct CSRMESH_TIME_GET_STATE_T`

Getting Time Broadcast Interval: Upon receiving a TIME_GET_STATE message, the device responds with a TIME_STATE message with the current state information.

`struct CSRMESH_TIME_STATE_T`

Set time broadcast interval.

`struct CSRMESH_TIME_BROADCAST_T`

Synchronise wall clock time from client device: This message is always sent with TTL=0. This message is sent at intervals by the clock master. It is always sent with TTL=0. It is repeated, but the time is updated before each repeat is sent. The clock master repeats the message 5 times, relaying stations repeat it 3 times. When a node receives a clock broadcast its behaviour depends on the current clock state: n MASTER: Ignore broadcasts.n INIT: Start the clock; relay this message. Set state to NO_RELAY if MasterFlag set, otherwise RELAY_MASTER. Start relay timer.n RELAY: Correct clock if required. Relay this message. Set state to NO_RELAY if MasterFlag set, otherwise RELAY_MASTER. Start relay timer.n NO_RELAY: Ignore. State will be reset to RELAY when the relay timer goes off.n RELAY_MASTER: Relay message only if it is from the clock master and set state to NO_RELAY.n The relay timer is by default 1/4 of the clock broadcast interval (15 seconds if the interval is 60 seconds). This means that each node will relay a message only once, and will give priority to messages from the clock master (which always causes the clock to be corrected). Messages from other nodes will only cause clock correction if they exceed the max clock skew (250ms).

301 time_server.h File Reference

Detailed Description

This file provides the prototypes of the server functions defined in the CSRmesh Time model.

301.1 Functions of time_server.h File Reference

```
CSRmeshResult TimeModelInit (CsrUint8 nw_id, CsrUint16 *group_id_list,
CsrUint16 num_groups, CSRMESH_MODEL_CALLBACK_T app_callback)
```

Model Initialisation.

```
CSRmeshResult TimeState (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8 ttl,
CSRMESH_TIME_STATE_T *p_params)
```

Set time broadcast interval.

```
CSRmeshResult TimeBroadcast (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8 ttl,
CSRMESH_TIME_BROADCAST_T *p_params)
```

Synchronise wall clock time from client device: This message is always sent with TTL=0. This message is sent at intervals by the clock master. It is always sent with TTL=0. It is repeated, but the time is updated before each repeat is sent. The clock master repeats the message 5 times, relaying stations repeat it 3 times. When a node receives a clock broadcast its behaviour depends on the current clock state: n MASTER: Ignore broadcasts.n INIT: Start the clock; relay this message. Set state to NO_RELAY if MasterFlag set, otherwise RELAY_MASTER. Start relay timer.n RELAY: Correct clock if required. Relay this message. Set state to NO_RELAY if MasterFlag set, otherwise RELAY_MASTER. Start relay timer.n NO_RELAY: Ignore. State will be reset to RELAY when the relay timer goes off.n RELAY_MASTER: Relay message only if it is from the clock master and set state to NO_RELAY.n The relay timer is by default 1/4 of the clock broadcast interval (15 seconds if the interval is 60 seconds). This means that each node will relay a message only once, and will give priority to messages from the clock master (which always causes the clock to be corrected). Messages from other nodes will only cause clock correction if they exceed the max clock skew (250ms).

302 tracker_client.h File Reference

Detailed Description

This files provides the prototypes of the client functions defined in the CSRmesh Tracker model.

302.1 Functions of tracker_client.h File Reference

```
CSRmeshResult TrackerModelClientInit (CSRMESH_MODEL_CALLBACK_T app_callback)
```

Initialises Tracker Model Client functionality.

```
CSRmeshResult TrackerFind (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8 ttl,
CSRMESH_TRACKER_FIND_T *p_params)
```

Finding an Asset: Upon receiving a TRACKER_FIND message, the server checks its tracker cache to see if it has received an ASSET_ANNOUNCE message recently that has the same DeviceID. If it finds one, it will send a TRACKER_FOUND message with the cached information.

```
CSRmeshResult TrackerClearCache (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8
ttl)
```

Clear tracker cache.

```
CSRmeshResult TrackerSetProximityConfig (CsrUint8 nw_id, CsrUint16 dest_id,
CsrUint8 ttl, CSRMESH_TRACKER_SET_PROXIMITY_CONFIG_T *p_params)
```

Set tracker proximity config.

303 tracker_model.h File Reference

Detailed Description

Defines CSRmesh Tracker Model specific data structures .

303.1 Data Structures of tracker_model.h File Reference

`struct CSRMESH_TRACKER_FIND_T`

CSRmesh Tracker Model message types.

`struct CSRMESH_TRACKER_FOUND_T`

Asset found.

`struct CSRMESH_TRACKER_REPORT_T`

Asset report.

`struct CSRMESH_TRACKER_SET_PROXIMITY_CONFIG_T`

Set tracker proximity config.

304 tracker_server.h File Reference

Detailed Description

This file provides the prototypes of the server functions defined in the CSRmesh Tracker model.

304.1 Functions of tracker_server.h File Reference

```
CSRmeshResult TrackerModelInit (CsrUint8 nw_id, CsrUint16 *group_id_list,
CsrUint16 num_groups, CSRMESH_MODEL_CALLBACK_T app_callback)
```

Model Initialisation.

```
CSRmeshResult TrackerFound (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8 ttl,
CSRMESH_TRACKER_FOUND_T *p_params)
```

Asset found.

```
CSRmeshResult TrackerReport (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8 ttl,
CSRMESH_TRACKER_REPORT_T *p_params)
```

Asset report.

305 tuning_client.h File Reference

Detailed Description

This files provides the prototypes of the client functions defined in the CSRmesh Tuning model.

305.1 Functions of tuning_client.h File Reference

```
CSRmeshResult TuningModelClientInit (CSRMESH_MODEL_CALLBACK_T app_callback)
```

Initialises Tuning Model Client functionality.

```
CSRmeshResult TuningProbe (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8 ttl,
CSRMESH_TUNING_PROBE_T *p_params)
```

Tuning Probe: The Tuning Probe message is sent to discover neighbours. This messages is issued by devices wanting to determine their density metrics.n The message is sent in two forms. A short form omitting both ScanDutyCycle and BatteryState with a TTL=0. This allows immediate neighbours to perform various calculations and in turn provide their own PROBE messages. The long version is only provided with TTL>0. This cannot be used for immediate neighbour density determination, but can be used to determine the overall network density. The ability to identify if a node is a potential pinch-point in the Mesh network can be achieved through the comparison of immediate and average number of neighbours within the network. The usage of the PROBE message with TTL=0 or TTL>0 is a way to perform these computations. It is worth noting that the periodicity of these two types of messages are different; messages with TTL>0 is much more infrequent than messages with TTL=0. Furthermore, it is wise not to use messages for TTL>0 and embedded values in the determination of the average values. The AverageNumNeighbour field is fixed point 6.2 format encoded. The ScanDutyCycle is expressing percentage for numbers from 1 to 100 and (x-100)/10 percentage for numbers from 101 to 255.

```
CSRmeshResult TuningGetStats (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8 ttl,
CSRMESH_TUNING_GET_STATS_T *p_params)
```

Getting Tuning Stats: These messages are aimed at collecting statistics from specific nodes. This message allows for the request of all information or for some of its parts. Responses are multi-parts, each identified with an index (combining a continuation flag - top bit). MissingReplyParts for the required field serves at determining the specific responses one would like to collect. If instead all the information is requested, setting this field to zero will inform the destination device to send all messages. Importantly, response (STATS_RESPONSE) messages will not necessarily come back in order, or all reach the requestor. It is essential to handle these cases in the treatment of the collected responses.

```
CSRmeshResult TuningSetConfig (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8 ttl,
CSRMESH_TUNING_SET_CONFIG_T *p_params)
```

Setting (or reading) tuning config: Omitted or zero fields mean do not change. This message enforces the state of the recipient. The state is defined by two goals, normal and high traffic, and their associated number of neighbour to decide which cases to follow. Goals are expressed with unit-less values ranging from 0 to 255. These goals relate to metrics calculated on the basis of density computed at the node and across the network. The expectation is for these goals to be maintained through modification of the receive duty cycle. The average of number of neighbours for high and normal traffic is expressed as a ratio, both numbers sharing the same denominator and each representing their respective numerators. The duty cycle encoding follows the same rules as per duty cycle encoding encountered in PROBE message. This message comes in two formats. A fully truncated form containing only the OpCode (thus of length 2) is used to indicate a request for information. This message should be answered by the appropriate ACK_CONFIG. Further interpretations of this message are: 1. Missing ACK field implies that a request for ACK_CONFIG is made. Thus, this is a special case of the fully truncated mode. However, the provided fields are meant to be used in the setting of goals. 2. Individual fields with zero value are meant NOT to be changed in the received element. Same as for missing fields in truncated messages. Furthermore, in order to improve testing, a combination of values for main and high goals are conventionally expected to be used for defining two behaviours: 1. Suspended: Tuning Probe messages (TTL=0) should be sent and statistics maintained, but the duty cycle should not be changed - thus goals will never be achieved. The encoding are: Main Goal = 0x00 and High Goal = 0xFE. 2. Disable: No Tuning Probe message should be sent and statistics should not be gathered - averaged values should decay. The encoding are: Main Goal = 0x00 and High Goal = 0xFF.

306 tuning_model.h File Reference

Detailed Description

Defines CSRmesh Tuning Model specific data structures .

306.1 Data Structures of tuning_model.h File Reference

`struct CSRMESH_TUNING_PROBE_T`

CSRmesh Tuning Model message types.

`struct CSRMESH_TUNING_GET_STATS_T`

Getting Tuning Stats: These messages are aimed at collecting statistics from specific nodes. This message allows for the request of all information or for some of its parts. Responses are multi-parts, each identified with an index (combining a continuation flag - top bit). MissingReplyParts for the required field serves at determining the specific responses one would like to collect. If instead all the information is requested, setting this field to zero will inform the destination device to send all messages. Importantly, response (STATS_RESPONSE) messages will not necessarily come back in order, or all reach the requestor. It is essential to handle these cases in the treatment of the collected responses.

`struct CSRMESH_TUNING_STATS_T`

Current Asset State: Response to the request. The PartNumber indicates the current index of the message, the top bit of this field is used to indicate that more messages are available after this part. For example, a response made up of three messages will have part numbers 0x80, 0x81 and 0x02. Each message has a maximum of two neighbours. The combination of these responses and PROBE (TTL>0) are a means to establish an overall perspective of the entire Mesh Network.

`struct CSRMESH_TUNING_ACK_CONFIG_T`

Current tuning config for a device: This message comes as a response to a SET_CONFIG. Encoding its various fields follow the same convention as the ones exposed in SET_CONFIG.

`struct CSRMESH_TUNING_SET_CONFIG_T`

Setting (or reading) tuning config: Omitted or zero fields mean do not change. This message enforces the state of the recipient. The state is defined by two goals, normal and high traffic, and their associated number of neighbour to decide which cases to follow. Goals are expressed with unit-less values ranging from 0 to 255. These goals relate to metrics calculated on the basis of density computed at the node and across the network. The expectation is for these goals to be maintained through modification of the receive duty cycle. The average of number of neighbours for high and normal traffic is expressed as a ratio, both numbers sharing the same denominator and each representing their respective numerators. The duty cycle encoding follows the same rules as per duty cycle encoding encountered in PROBE message. This message comes in two formats. A fully truncated form containing only the OpCode (thus of length 2) is used to indicate a request for information. This message should be answered by the appropriate ACK_CONFIG. Further interpretations of this message are: 1. Missing ACK field implies that a request for ACK_CONFIG is made. Thus, this is a special case of the fully truncated mode. However, the provided fields are meant to be used in the setting of goals. 2. Individual fields with zero value are meant NOT to be changed in the received element. Same as for missing fields in truncated messages. Furthermore, in order to improve testing, a combination of values for main and high goals are conventionally expected to be used for defining two behaviours: 1. Suspended: Tuning Probe messages (TTL=0) should be sent and statistics maintained, but the duty cycle should not be changed - thus goals will never be achieved. The encoding are: Main Goal = 0x00 and High Goal = 0xFE. 2. Disable: No Tuning Probe message should be sent and statistics should not be gathered - averaged values should decay. The encoding are: Main Goal = 0x00 and High Goal = 0xFF.

307 tuning_server.h File Reference

Detailed Description

This file provides the prototypes of the server functions defined in the CSRmesh Tuning model.

307.1 Data Structures of tuning_server.h File Reference

`struct tuningStats_t`

Tuning statistics of neighbouring devices.

307.2 Functions of tuning_server.h File Reference

```
CSRmeshResult TuningModelInit (CsrUint8 nw_id, CsrUint16 *group_id_list,
CsrUint16 num_groups, CSRMESH_MODEL_CALLBACK_T app_callback)
```

Model Initialisation.

```
void TuningModelStart (uint16 probePeriod, uint16 reportPeriod)
```

Starts sending TUNING_PROBE messages and adjusts the scan duty cycle based on the analysis of the received messages.

```
void TuningModelStop (void)
```

Stops auto-tuning of rx duty cycle.

```
uint16 TuningReadStats (tuningStats_t **stats)
```

Returns the tuning statistics of the neighbouring devices.

308 watchdog_client.h File Reference

Detailed Description

This files provides the prototypes of the client functions defined in the CSRmesh Watchdog model.

308.1 Functions of watchdog_client.h File Reference

```
CSRmeshResult WatchdogModelClientInit (CSRMESH_MODEL_CALLBACK_T app_callback)
```

Initialises Watchdog Model Client functionality.

```
CSRmeshResult WatchdogClientMessage (CsrUint8 nw_id, CsrUint16 dest_id,
CsrUint8 ttl, CSRMESH_WATCHDOG_MESSAGE_T *p_params)
```

Upon receiving a WATCHDOG_MESSAGE message, if the RspSize field is set to a non-zero value, then the device shall respond with a WATCHDOG_MESSAGE with the RspSize field set to zero, and RspSize -1 octets of additional RandomData.

```
CSRmeshResult WatchdogSetInterval (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8
ttl, CSRMESH_WATCHDOG_SET_INTERVAL_T *p_params)
```

Upon receiving a WATCHDOG_SET_INTERVAL message, the device shall save the Interval and ActiveAfterTime fields into the Interval and ActiveAfterTime variables and respond with a WATCHDOG_INTERVAL message with the current variable values.

309 watchdog_model.h File Reference

Detailed Description

Defines CSRmesh Watchdog Model specific data structures .

309.1 Data Structures of watchdog_model.h File Reference

`struct CSRMESH_WATCHDOG_MESSAGE_T`

CSRmesh Watchdog Model message types.

`struct CSRMESH_WATCHDOG_SET_INTERVAL_T`

Upon receiving a WATCHDOG_SET_INTERVAL message, the device shall save the Interval and ActiveAfterTime fields into the Interval and ActiveAfterTime variables and respond with a WATCHDOG_INTERVAL message with the current variable values.

`struct CSRMESH_WATCHDOG_INTERVAL_T`

Watchdog interval state.

310 watchdog_server.h File Reference

Detailed Description

This file provides the prototypes of the server functions defined in the CSRmesh Watchdog model.

310.1 Functions of watchdog_server.h File Reference

```
CSRmeshResult WatchdogModelInit (CsrUint8 nw_id, CsrUint16 *group_id_list,
CsrUint16 num_groups, CSRMESH_MODEL_CALLBACK_T app_callback)
```

Model Initialisation.

```
CSRmeshResult WatchdogMessage (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8 ttl,
CSRMESH_WATCHDOG_MESSAGE_T *p_params)
```

Upon receiving a WATCHDOG_MESSAGE message, if the RspSize field is set to a non-zero value, then the device shall respond with a WATCHDOG_MESSAGE with the RspSize field set to zero, and RspSize -1 octets of additional RandomData.

```
CSRmeshResult WatchdogInterval (CsrUint8 nw_id, CsrUint16 dest_id, CsrUint8
ttl, CSRMESH_WATCHDOG_INTERVAL_T *p_params)
```

Watchdog interval state.