Push every boundary.™

CSR µEnergy®™

CSRmesh™ 1.0 Control Application

Application Note

Issue 1

Document History

Revision	Date	History
1	25 JUN 14	Original publication of this document

Contacts

General information	www.csr.com
Information on this product	sales@csr.com
Customer support for this product	www.csrsupport.com
More detail on compliance and standards	product.compliance@csr.com
Help with this document	comments@csr.com

Trademarks, Patents and Licences

Safety-critical Applications

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

Performance and Conformance

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

Contents

Tables, Figures and Equations

1. Introduction

This document describes the CSRmesh™ Control Application v1.0 for Android

The application demonstrates the following use cases:

- **Device Association**: Associating discovered CSRmesh devices
- **Setting up a Network**: Associating devices and configuring groups of devices.
- **Light Control**: Switching on and off, setting colour and brightness of individual or group of lights.

This application connects to a CSRmesh device which implements one of the following applications

- **CSRmesh Light Application**: Implements the CSRmesh Light device and bridge.
- **CSRmesh Switch Application**: Implements the CSRmesh Switch device and bridge.
- **CSRmesh Bridge Application**: Implements the CSRmesh Bridge device.

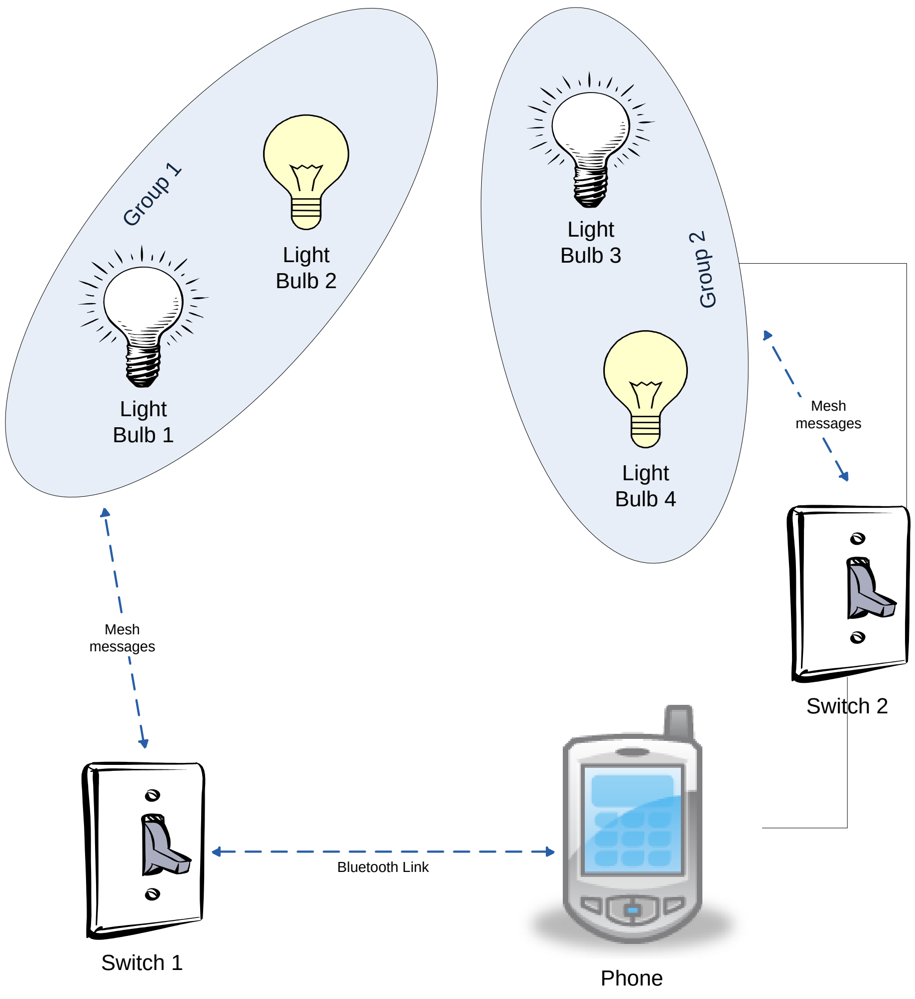


Figure 1.1: CSRmesh Use Case

1.1. Application Overview

The CSRmesh Control Application runs on a phone or a tablet which has the Android version 4.3 or higher. It communicates with the CSRmesh devices by connecting to one of the devices which supports CSR custom defined CSRmesh Control Profile.

1.2. CSRmesh Control Profile

The CSRmesh Control Profile defines the following behaviour when

- A network of devices (lights and switches) needs to be created.
- Controlling the device after a network has been created. For example, controlling on/off, intensity or the colour of a light.
- Read the status of a device in the network, i.e. on/off state, colour or intensity of a light device.

The Profile defines the following two roles:

Role	Description
CSRmesh Bridge Device	Receives commands from host and sends them over the CSRmesh network to associated devices. It receives responses from associated devices over the CSRmesh and forwards them onto the host over a Bluetooth Smart connection.
CSRmesh Control Device	The CSRmesh Control Device provides the interface to create a network of devices and to control the associated devices. The control commands are sent over a Bluetooth Smart connection to the CSRmesh Bridge.

Table 1: CSRmesh Control Profile Roles

The CSRmesh Control Application implements the CSRmesh Control Device role

1.2.1. Application Topology

The CSRmesh Control Application uses the following topology:

Role	Mesh Control Service	GAP Service	GATT Service	Device Information Service
GATT Role	GATT Client	GATT Client	GATT Client	GATT Client
GAP Role	Central	Central	Central	Central

Table 2: Application Topology

Role	Responsibility
GATT Client	Initiates commands and request towards a server and receives responses, notifications and indications sent by the server.
GAP Central	Initiates a connection towards the peer device and acts as a Master device in the connection.

Table 3: Roles and Responsibilities

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

1.2.2. Services supported in GATT Client Role

The application acts as GATT client for the following services:

- Mesh Control Service
- GATT Service
- GAP Service

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

See Figure 1.2 for the services supported in the GATT Client Role.

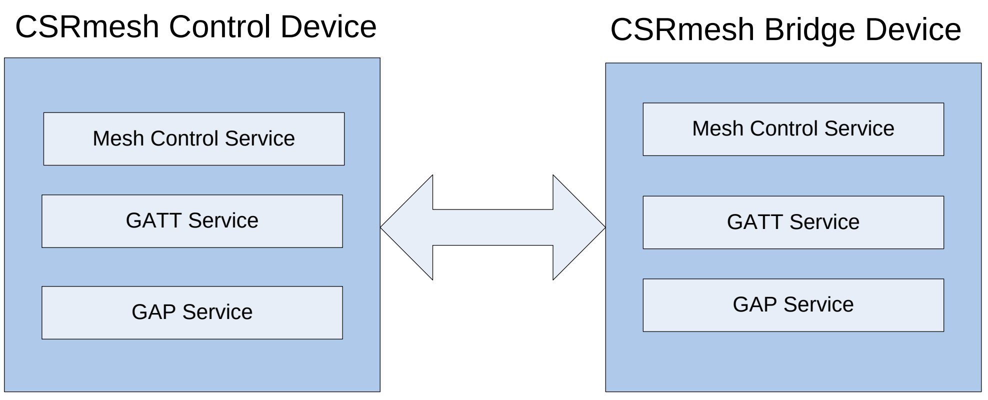


Figure 1.2: Primary Services

2. Using the Application

This section describes how the application can be used with CSRmesh on-chip Applications to setup, configure and control a CSRmesh network.

2.1. CSRmesh Components

This application can be demonstrated using the following components.

Component	Hardware	Application
CSRmesh Control Device	Android Bluetooth LE Device	CSRmesh Control Application v1.0
CSRmesh Light Device	CSRmesh Development Board DB-CSR1010-10185-1A	CSRmesh Light Application v1.0

Table 4: Application Components

2.2. Getting started

- When the app is launched the initial screen shows a **Scan and Connect** button.
- Press **Scan and Connect** to discover devices that support "CSRmesh" service and connect to the nearest device.
- The application automatically connects to the device with the highest RSSI.

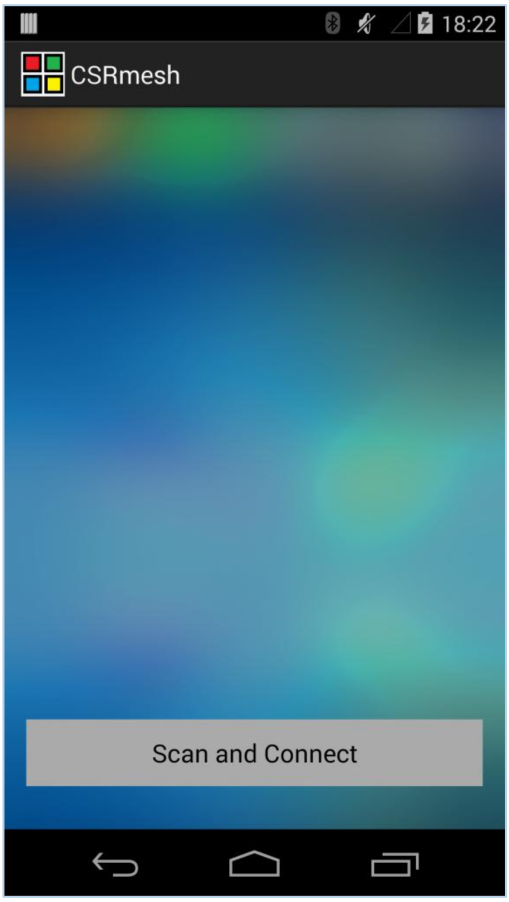


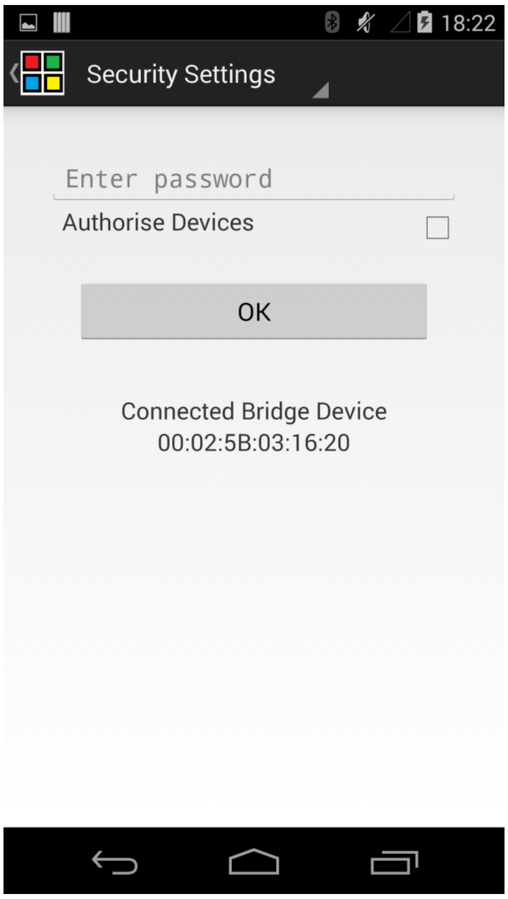
Figure 2.1: Initial Screen

2.3. Setting up a network

To setup a CSRmesh network a Network Key needs to be generated from a passphrase. This section describes the steps for generating a network key to start setting up a new network

- Once the application connects to a CSRmesh Device a drop down menu is enabled which can be used to navigate to any of the mesh control pages.
- Select the security settings page shown in Figure 2.2. If the application is run for the first time, it jumps to this page automatically.
- Enter a passphrase and press **OK** to setup a new network key.
- This network key will be used to associate new devices to the network.
- This page shows the Bluetooth Address of the connected bridge device.

It is important to un-check 'Authorise Devices' if the CSRmesh devices do not support Authorisation code otherwise it will not be possible to associate devices.

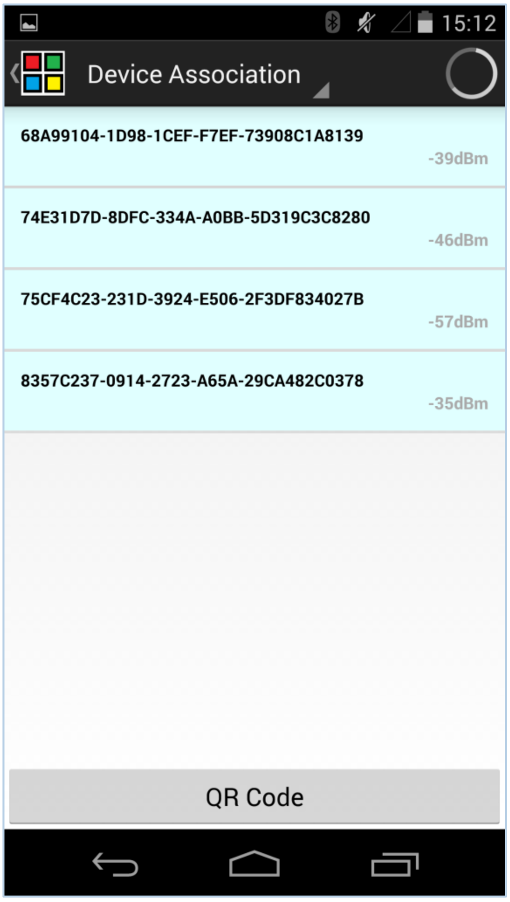


Connected Bridge Device
00:02:5B:03:16:20

Figure 2.2: Security Settings Screen

2.4. Associating devices to network

- Navigate to the device association screen from the main menu
- This screen lists the 128-bit UUIDs of the devices ready for association
- Select the UUID of the Device to associate it to the network


15:12

Device Association

68A99104-1D98-1CEF-F7EF-73908C1A8139 -39dBm

74E31D7D-8DFC-334A-A0BB-5D319C3C8280 -46dBm

75CF4C23-231D-3924-E506-2F3DF834027B -57dBm

8357C237-0914-2723-A65A-29CA482C0378 -35dBm

Figure 2.3: Device Association Screen

- A message box pops up to indicate association is in progress as shown Figure 2.4

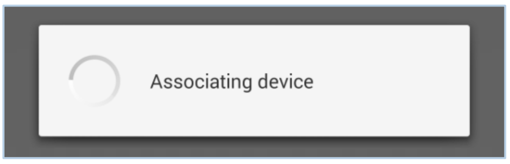

Associating device

Figure 2.4: Association in progress message

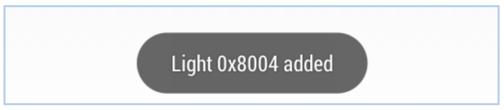

Light 0x8004 added

Figure 2.5: Device associated message

- Once the device is associated a message box is displayed with the 16-bit Device ID assigned to the device shown in Figure 2.5

2.5. Device authorisation

- When CSRmesh Devices support authorisation. The authorisation code is made available by the devices using one of the Authorisation Code Display Mechanisms.
- There are two methods for encoding the Authorization Code and Device UUID for a device, QR-Code and Short Code. See 3 for details on how to encode Authorisation code for Display.
- To authorise devices on the network check the **Authorise Devices** check box. See Figure 2.2
- When the '**Authorise devices**' is checked, the application allows the user to enter the ShortCode. See Figure 2.6

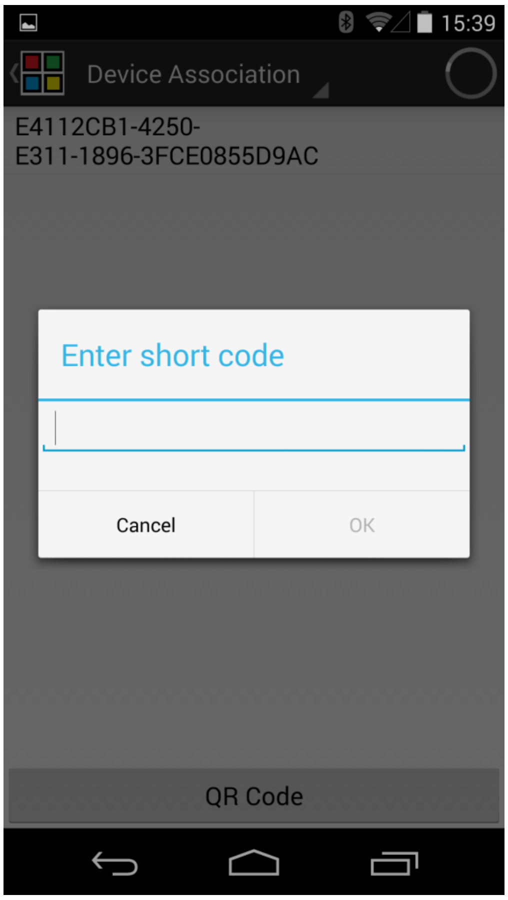


Figure 2.6: Device authorisation short code

- To enter the Authorisation code using a QR-Code, Cancel short code box and click the QR Code. This will lead to a QR-Code Scanner. See Figure 2.7

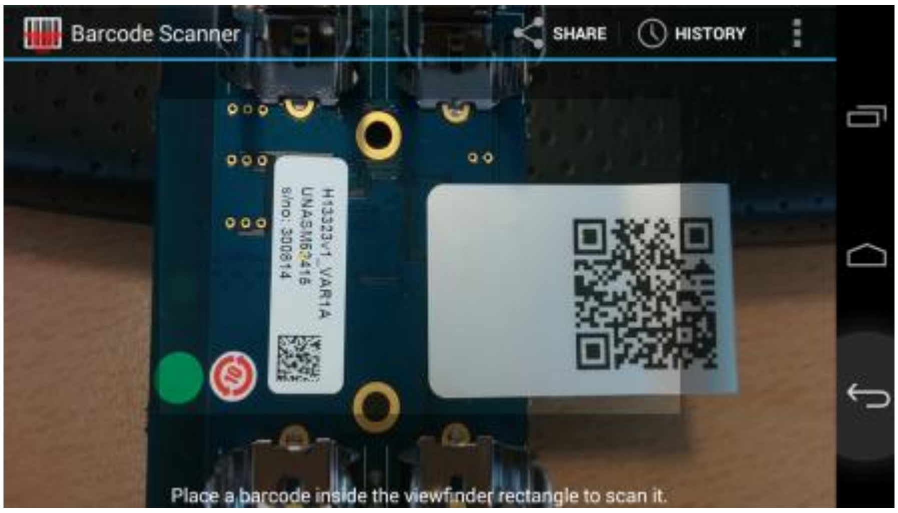


Figure 2.7: Device authorisation QR-Code

- Scan the QR-Code. Once the QR-Code is scanned correctly the application will immediately start associating the device.

Note:

By default the CSRmesh Light and the CSRmesh Switch Applications define the Authorisation Code as 0x7856341278563412

Note:

The QR code scanning by the application requires that the following QR code app to be installed on the phone https://play.google.com/store/apps/details?id=com.google.zxing.client.android

2.6. Controlling Lights

- The light control page lists all associated lights and configured groups.
- Select an individual light or 'All lights' or a Group from the drop down list as shown in Figure 2.8 to control the lights
- Tap the colour wheel to set colour, and move the slider at the bottom to adjust brightness of the selected lights. See Figure 2.8
- Slide the Power button to turn ON or turn OFF the lights.

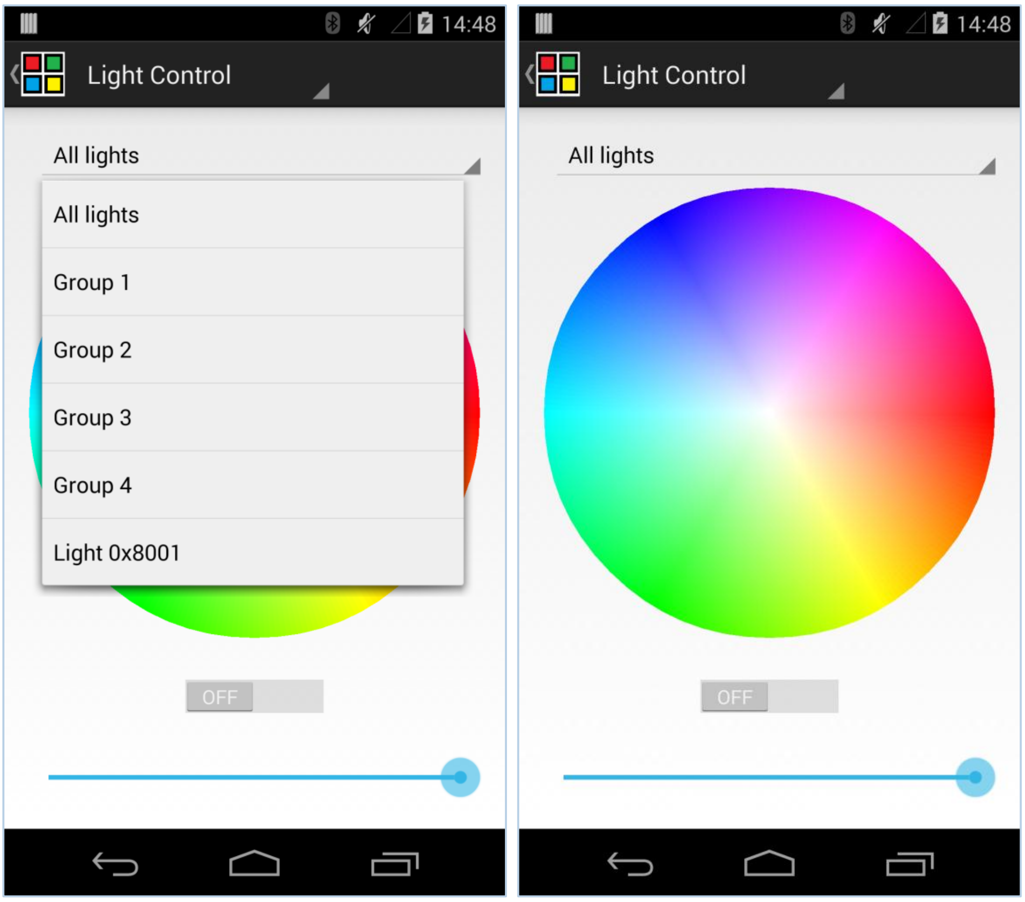
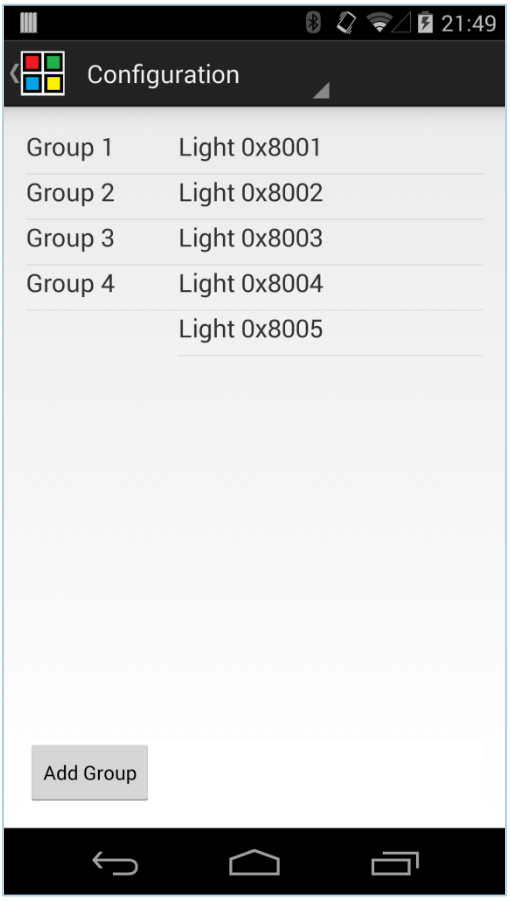


Figure 2.8: Controlling lights

2.7. Configuration

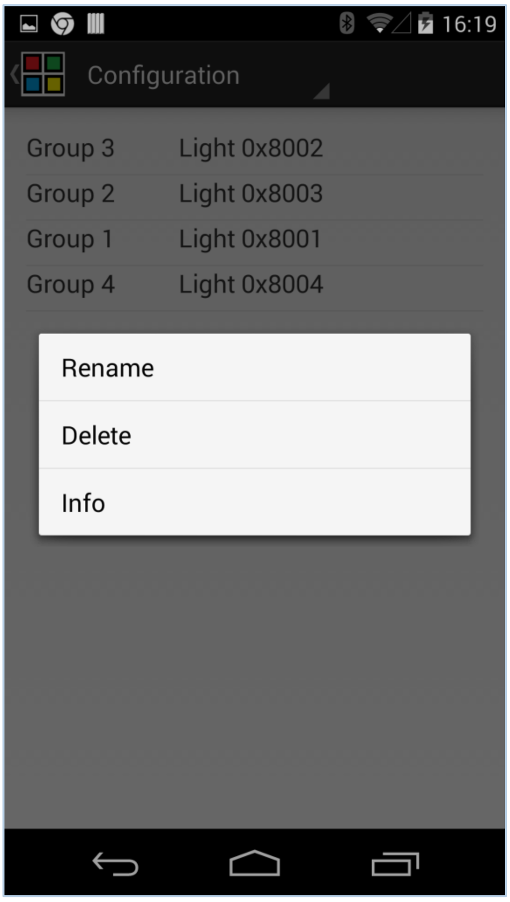
- Select the configuration page from the application main menu
- Configuration screen lists the Groups and associated lights.
- By default the application provides four groups to which no light will be assigned.



Group 1	Light 0x8001
Group 2	Light 0x8002
Group 3	Light 0x8003
Group 4	Light 0x8004
	Light 0x8005

Figure 2.9: Configuration screen

- Touch and hold a light to popup a menu to configure the light. See Figure 2.10.
- Select **Rename** to rename a light
- Select **Delete** to remove the light from the network.
- Select **Info** to see information on the light.
- Touch and hold a group to popup a menu to rename or delete a group.



16:19

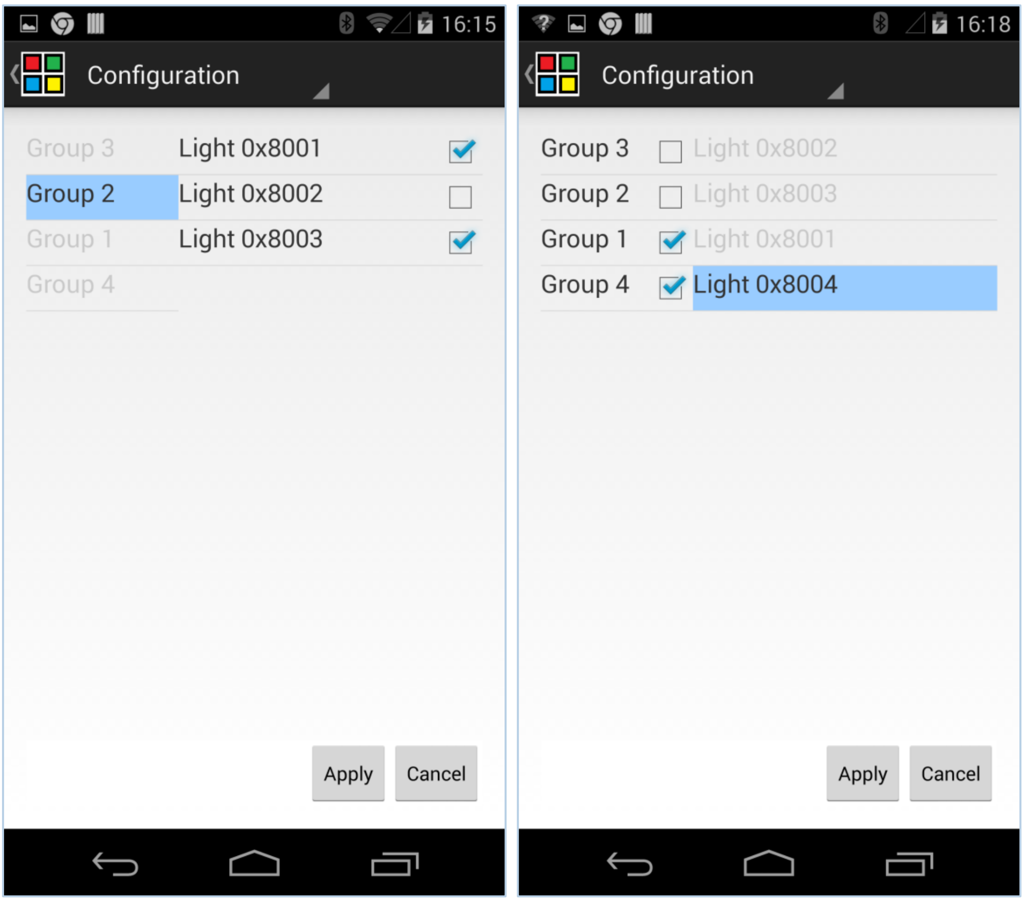
Configuration

Group 3	Light 0x8002
Group 2	Light 0x8003
Group 1	Light 0x8001
Group 4	Light 0x8004

Figure 2.10: Light configuration

2.8. Group Setup

- Lights and Switches can be grouped into different groups. See Figure 2.11.
- Select a group to show the check boxes against the lights and switches. Check the lights and/or switches to be assigned to the group and click **Apply**.
- Select a light or a switch to show check boxes against the groups. Check the groups to which the light has to be assigned and click **Apply**
- Once group assignment is complete, **'Groups updated OK'** message will be displayed.



Group 3 Light 0x8001 ✓
Group 2 Light 0x8002 ☐
Group 1 Light 0x8003 ✓
Group 4

Group 3 ☐ Light 0x8002
Group 2 ☐ Light 0x8003
Group 1 ✓ Light 0x8001
Group 4 ✓ Light 0x8004

Figure 2.11: Group Setup

3. Using the Library

This section describes how to use the CSRmesh Android library that can be used to create an Android application that can associate and control CSRmesh devices.

3.1. Overview

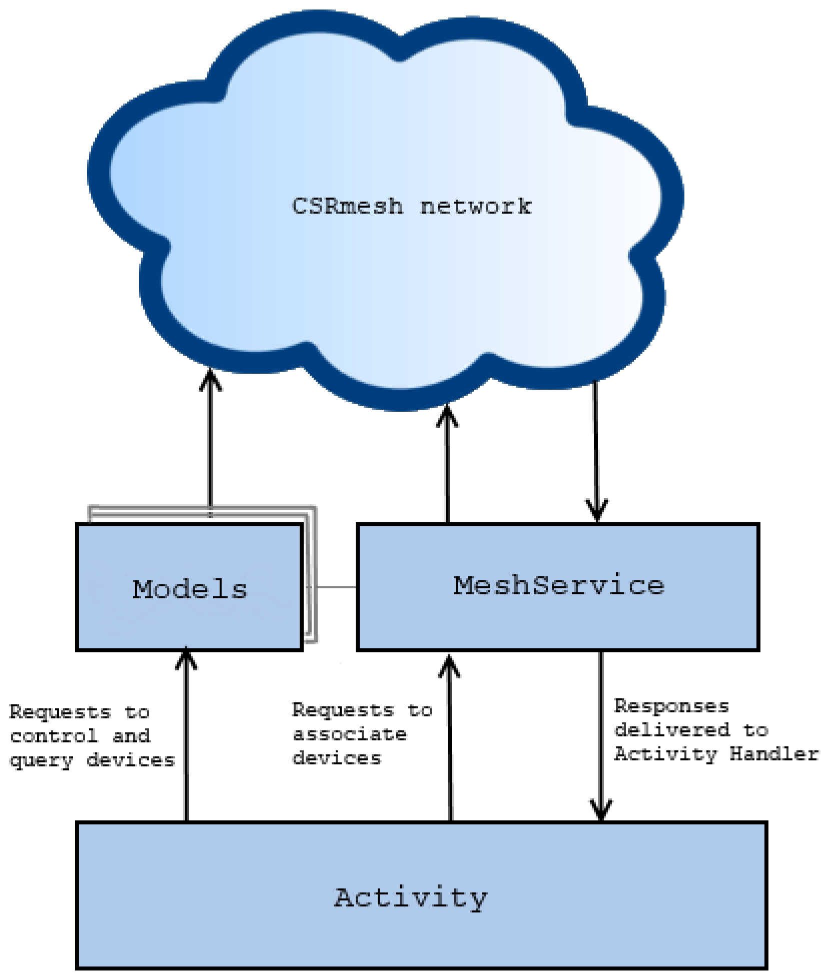

CSRmesh network
Models
MeshService
Requests to control and query devices
Requests to associate devices
Responses delivered to Activity Handler
Activity

Figure 3.1: CSRmesh library components

Android applications communicate with CSRmesh via the `MeshService` and the model classes. Figure 3.1 shows the components and how they are connected. To simplify this diagram the bridge device is not shown.

`MeshService` provides methods to connect to the bridge device and associate devices. When a device is associated it is assigned a device address that can be used to control or query that device using the model classes.

`MeshService` holds a reference to a single instance of each of the model classes. So to use one of the model methods to control or query a device, an application must call a `MeshService` method to obtain a reference to the model class required. Devices can then be controlled by calling that model class's methods via this reference. See section 3.7 for a list of model classes supported in CSRmesh 1.0.

All responses to association, control and query requests as well as status and error messages are delivered asynchronously in messages to the `Activity's Handler` (see section 3.5).

The following is a summary of the steps required to use the library to control a device. Further explanation and code for each of these steps is provided in sections 3.3 onwards.

The following steps need to be performed just once when the application starts:

- Make a connection to `MeshService`.
- When service connection completes call `MeshService.initLe` to connect to a bridge device.

The following steps need to be performed each time devices are associated:

- Get a list of devices to associate by calling `MeshService.setDiscoveryEnabled`.
- Associate reported devices using `MeshService.associateDevice`.
- When a device completes association, save the device id.
- Obtain a reference to the `ConfigModel` object using `MeshService.getConfigModel`.
- Call the `ConfigModel.requestInfo` method to find out the models each device supports.

Once a device is associated, its model functions can be accessed using the following steps. In this example the light model is used to set a light's colour.

- Obtain a reference to the `LightModel` object using `MeshService.getLightModel`.
- Call the `LightModel.setRgb` method to set the light's colour.

3.2. Project Setup

The library file '`meshlibrary.jar`' should be added to the project libs directory, along with the spongy castle libraries that are used for CSRmesh security. These files can be found under `CsrMeshDemo\libs`.

It is also necessary to make changes to the `AndroidManifest.xml` for the project. The following should be added inside the `<application>` element:

```
<service
    android:name="com.csr.mesh.MeshService"
    android:enabled="true"
    android:exported="false" >
</service>
```

The library also needs the following Bluetooth permissions to be specified in the `AndroidManifest.xml`:

```
<uses-permission android:name="android.permission.BLUETOOTH" />
<uses-permission android:name="android.permission.BLUETOOTH_ADMIN" />
```

3.3. Service Connection

The Android application should bind to `MeshService` to access CSRmesh library functions. This is the same as binding to any Android service. The steps are shown briefly below.

Declare the `MeshService` object in the Activity class:

```
private MeshService mService;
```

Bind to the service inside `onCreate`:

```
Intent bindIntent = new Intent(this, MeshService.class);
bindService(bindIntent, mServiceConnection, Context.BIND_AUTO_CREATE);
```

Define the service connection to handle `MeshService` connection events:

```
private ServiceConnection mServiceConnection = new ServiceConnection() {
    public void onServiceConnected(ComponentName className, IBinder rawBinder) {
        mService = ((MeshService.LocalBinder) rawBinder).getService();
    }
    public void onServiceDisconnected(ComponentName classname) {
        mService = null;
    }
};
```

3.4. Connecting to a Bridge Device

Once `MeshService` is connected, the CSRmesh stack needs to be initialised and a connection to a bridge device made. This is achieved using the following method:

```
mService.initLe(Handler meshMessageHandler, BluetoothDevice bridgeDevice);
```

Pass in the `Activity's Handler` (see section 3.5), and the `BluetoothDevice` object representing the bridge device to connect to. Scanning for bridge devices is the same as scanning for any BLE device. This is covered in the Android documentation at https://developer.android.com/guide/topics/connectivity/bluetooth-le.html#find.

CSRmesh bridges are LE only devices, so adverts should be filtered by that device type:

```
if (device.getType() == BluetoothDevice.DEVICE_TYPE_LE) {
    // Process device.
}
```

Adverts should also be filtered by service UUID to make sure the device is a CSRmesh device. The bridge service UUID is `0xFEF1`.

The `meshMessageHandler` will receive `MeshService.MESSAGE_LE_CONNECTED` when the bridge is connected. `MeshService.MESSAGE_LE_DISCONNECTED` is received when the link is disconnected.

3.5. Handling Responses

Since messages from devices that are part of a CSRmesh will take an unknown time to reach the Android device, the `MeshService` handles responses from devices asynchronously. Some messages are handled silently by the `MeshService`, but most of the messages are sent to the `Handler` registered with the `MeshService`. Only a single `Handler` is required; messages can be differentiated based on device address.

The messages that the Handler will receive are all defined in `MeshService` as constants that begin with "`MESSAGE_`". Messages often have extra data associated that can be retrieved via the message object using the correct key. The keys for this data are defined in `MeshService` as constants and are documented in the Javadoc. They begin with "`EXTRA_".`

For example when a new lamp has been associated the `Handler` will receive `MeshService.MESSAGE_DEVICE_ID`, with data indexed by the key `MeshService.EXTRA_DEVICE_ID`. This contains the MCP layer device id (address) of the newly associated lamp. The following `handleMessage` method implementation in the `Handler` would extract the device id:

```
public void handleMessage(Message msg) {
    switch(msg.what) {
        case MeshService.MESSAGE_DEVICE_ID:
            int deviceId = msg.getData().getInt(MeshService.EXTRA_DEVICE_ID);
            break;
    }
}
```

An important message that should be handled is `MESSAGE_TIMEOUT.` This is received in cases where no response was received from a device for a sent command. This message contains extra data with the key `EXTRA_EXPECTED_MESSAGE`. This contains the message that would have been received in case of success. E.g. If no response is received after sending the command to get firmware version, this would contain `MESSAGE_FIRMWARE_VERSION`.

3.6. Security and Associating Devices

All control messages sent to devices are encrypted using the network key. The network key is generated from a passphrase that can be set using the `MeshService` method `setNetworkPassPhrase`. This should be called before associating any devices.

Devices that wish to be associated will advertise their UUID. To discover devices that are advertising to be associated use the `setDiscoveryEnabled` method:

```
mService.setDiscoveryEnabled(true);
```

After discovery is enabled, the `Activity Handler` will receive `MESSAGE_DEVICE_UUID` for each device that has been discovered. This message contains a `ParcelUUID` object containing the UUID of the device, a 31-bit hash of this UUID, and the RSSI value. These values can be retrieved as follows:

```
ParcelUuid uuid = msg.getData().getParcelable(MeshService.EXTRA_UUID);
int uuidHash = msg.getData().getInt(MeshService.EXTRA_UUIDHASH_31);
int rssi = msg.getData().getInt(MeshService.EXTRA_RSSI);
```

To start association for a discovered device, use the `associateDevice` method, passing in the UUID hash received in the previous step:

```
mService.associateDevice(uuidHash, 0);
```

The second parameter is the authorisation code which is optional and can be set to zero as it is here. Other association functions also exist to associate with different parameters such as short code. See the Javadoc for `MeshService` for details.

Once association completes the `Activity Handler` will receive `MESSAGE_DEVICE_ASSOCIATED` for each associated device.

3.7. Controlling CSRmesh Devices via Model Classes

The app can request services from CSRmesh via the model classes. A reference to a model can be obtained via the `MeshService` object by calling the appropriate method. The models defined in this release, and the `MeshService` methods to access them are shown in Table 5.

Model	Access method	Functions
Attention Model	`getAttentionModel`	Tell a device to get user attention, e.g. flash light.
Bearer Model	`getBearerModel`	Enable or disable device relay function.
Config Model	`getConfigModel`	Get device configuration (including supported models), remove device association.
Light Model	`getLightModel`	Controlling and querying lights, e.g. setting and getting light colour.
Group Model	`getGroupModel`	Group management.
Power Model	`getPowerModel`	Set and get device power state.
Firmware Model	`getFirmwareModel`	Get device firmware version.
Ping Model	`getPingModel`	Ping command to help optimise network configuration.

Table 5: Available models in CSRmesh 1.0

An example of a function call to turn on the light with device id 0x8001 is:

```
mService.getPowerModel().setPowerState(PowerState.ON, 0x8001);
```

The available methods for each model are documented via the Javadoc for that class. The Javadoc also defines the messages that will be returned asynchronously in response to the command (if any). These should be handled as described in section 3.5.

3.8. Querying Model Support

Before using a model class to send a command to a device, the `ConfigModel requestInfo` method should be used to find out which models the device supports. Response is `MESSAGE_CONFIG_DEVICE_INFO` with data `EXTRA_DEVICE_INFO_TYPE` and `EXTRA_DEVICE_INFORMATION`. `EXTRA_DEVICE_INFO_TYPE` defines what is contained in `EXTRA_DEVICE_INFORMATION`. The possible values are shown in table 6.

EXTRA_DEVICE_INFO_TYPE	Meaning
0x00	Octets 0 to 7 of the DeviceUUID.
0x01	Octets 8 to 15 of the DeviceUUID.
0x02	Octets 0 to 7 of the model support bitmap.
0x03	Octets 8 to 15 of the model support bitmap.

Table 6: Values of EXTRA_DEVICE_INFO_TYPE

The `MeshService getMcpModelMask` function can be used to get the mask for a particular model to compare against the model support bitmap. An example follows.

```
int deviceId = msg.getData().getInt(MeshService.EXTRA_DEVICE_ID);
long bitmap = msg.getData().getLong(MeshService.EXTRA_DEVICE_INFORMATION);
// Look up the info type received in the DeviceInfo enum.
ConfigModel.DeviceInfo infoType = ConfigModel.DeviceInfo.values()
    [msg.getData().getByte(MeshService.EXTRA_DEVICE_INFO_TYPE)];
// Only interested in low part of model bitmap
if (infoType == DeviceInfo.MODEL_LOW &&
   (bitmap & MeshService.getMcpModelMask(LightModel.MODEL_NUMBER)) ==
        MeshService.getMcpModelMask(LightModel.MODEL_NUMBER)) {

      // This device supports light model.
}
```

Appendix A Encoding Authorisation code

There are two methods for encoding the Authorization Code and Device UUID for a device

- QR-Code
- Short Code

A.1 QR-Code

The QR-Code shall contain a URL that has the following format:

https://www.example.com/path?UUID=uuuu&AC=cccc

For example, the Device UUID of 2A3A0074-FD2B-4C7C-B61B-5A4EBBC5B2C8, with the Authorisation Code 0x11A43F1FCD4987BE that is being provided at https://mesh.example.com/add would encode to the string:

```
https://mesh.example.com/add?UUID=2A3A0074FD2B4C7CB61B5A4EBBC5B2C8&AC=11A43F1FCD4987BE
```

This would result in the following QR-Code:

Figure A.1: Example QR Code

A.2 ShortCode

The Short Code is composed of three fields: the ShortDeviceUUID, the Authorization Code, and a CheckSum.

ShortDeviceUUID	Authorization Code	CheckSum

The ShortDeviceUUID is the lowest 31 bits of SHA256 hash of the Device UUID.

The CheckSum is a 5-bit value. Given that the shortDeviceUUID is 31 bits, the Authorisation Code is 64 bits, and the CheckSum is 5 bits, the resultant ShortCode contains 100 bits of information.

The CheckSum shall be chosen such that the sum of each 5-bit value within the ShortCode shall result in the value of zero modulo 32.

This ShortCode is output 5-bits at a time by encoding each 5-bit value with the following characters in order “B b C D d F f G g H h K L M m N n P p R r T t U V W X 4 5 6 8 9” for values 0 to 31.

So B represents 0, b represents 1 and so on.

The text string should be split into five groups of four characters separated by a ‘-’ character to provide easier readability of the blocks of characters.

For example, the DeviceUUID of 2A3A0074-FD2B-4C7C-B61B-5A4EBBC5B2C8, with the Authorization Code 0x11A43F1FCD4987BE would encode to the string:

```
"NDhd-pbbD-Hb9D-9frR-b68T"
```

Document References

Document	Reference
Bluetooth Core Specification Version 4.1	https://www.bluetooth.org/Technical/Specifications/adopted.htm
Battery Service Specification Version 1.0	https://www.bluetooth.org/Technical/Specifications/adopted.htm
Device Information Service Specification Version 1.1	https://www.bluetooth.org/Technical/Specifications/adopted.htm
Service Characteristics And Descriptions	http://developer.bluetooth.org/gatt/characteristics/Pages/default.aspx
GATT Database Generator	CS-219225-UG
CSR µEnergy xIDE User Guide	CS-212742-UG
CSRmesh 1.0 Light Application Note	CS-318309-AN
CSRmesh 1.0 Switch Application Note	CS-318310-AN
CSRmesh 1.0 Bridge Application Note	CS-318311-AN
Installing the CSR Driver for the Profile Demonstrator Application	CS-235358-UG
CSRmesh Release Notes	https://www.csrsupport.com/CSRmesh
CSRmesh Android Application	https://www.csrsupport.com/CSRmesh

Terms and Definitions

API	Application Programmer's Interface
BLE	Bluetooth Low Energy (now known as Bluetooth Smart)
BlueCore®	Group term for CSR's range of Bluetooth wireless technology chips
Bluetooth®	Set of technologies providing audio and data transfer over short-range radio connections
CS	Configuration Store
CSR	Cambridge Silicon Radio
CSRmesh™	A CSR protocol that enables peer-to-peerlike networking of Bluetooth Smart devices
EEPROM	Electrically Erasable Programmable Read Only Memory
GAP	Generic Access Profile
GATT	Generic Attribute Profile
I^{2}C	Inter-Integrated Circuit
IOT	Internet of Things
IRK	Identity Resolving Key
LED	Light Emitting Diode
LM	Link Manager
MTL	Message Transport Layer
NVM	Non Volatile Memory
OTA	Over the Air
PIO	Programmable Input Output
PWM	Pulse Width Modulation
Tx	Transmit
UUID	Universally Unique Identifier