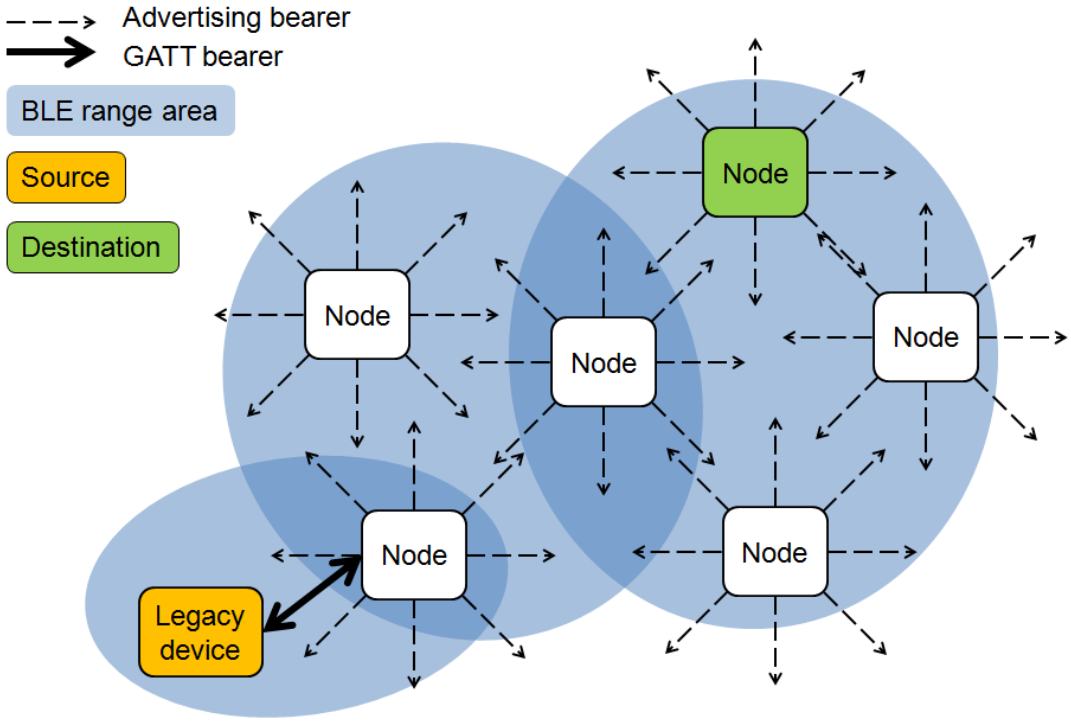
BlueNRG Mesh Android API guide for Mesh over Bluetooth low energy

Introduction

The Mesh over Bluetooth low energy (MoBLE) software is a stack of network protocols for Android®-based handheld and embedded devices (nodes) with Bluetooth low energy on-board. These nodes can be arranged in a Mesh network over Bluetooth low energy to create distributed control systems for applications like smart home systems.

Figure 1. Mesh network over Bluetooth low energy



1 Features of the BLE Mesh library

- Create a new network or load existing networks
- Start network processing
- Enumerate unprovisioned and provisioned devices
- Add unprovisioned nodes to network
- Send custom vendor model commands
- Send generic model commands
- Send lighting model commands
- Send group commands
- Save and restore network configurations
- Configure nodes for additional requirements
- Advise to start receiving callbacks
- Unadvise to stop receiving callbacks
- Stop network processing

2 Include library in Android application project

Follow the procedure below to include the library in the Android application project. The project structure is a standard Android project in which you create a new project and include Bluetooth low energy Mesh library to start developing Mesh applications.

Step 1. Create a new directory for the library under the directory where the Android project is located.

Step 2. Copy library (.aar) file in the new directory.

Step 3. Add these lines in the build.gradle of the app module after the Android tag.

```
repositories {
    flatDir {
        dirs '<path to the MobleLibrary-release.aar>'
    }
}
```

Step 4. Add library dependency in the build.gradle of the app module in the first line of the dependency tag.

```
dependencies  {
            compile(name:'MobileLibrary-release', ext:'aar')
            compile 'com.android.support:support-v4:25.3.1'
            compile 'com.android.support:design:25.3.1'
            compile 'com.android.support:appcompat-v7:25.3.1'
```

Step 5. If present, remove any ndk paths in the local.properties file of the Android project.

Android Studio sometimes appends an ndk path in the local.properties file while building the project; this causes the application to crash.

3 BlueNRG Mesh library guidelines

3.1 Network operation

3.1.1 Initialization API - Mesh network operations

mobleNetwork.createNetwork(...) creates new mesh network

- Address of Android smartphone is specified in parameters. This is the provisioner's address.
- Network and application keys are generated randomly.
- Nodes provisioned to other (existing) mesh networks are not accessible in newly created mesh network.
- Only one mesh network can work with the MoBLE stack at a time.

mobleNetwork.restoreNetwork(...) restores mesh network from its configuration saved in a JSON file.

- Configuration of one mesh network can be loaded from e-mail or synchronized from the cloud to another smartphone .

3.1.2 Allowable Mesh network operations after API initialization

Create/restore

- UserApplication.onCreate()
- MainActivity.onCreate()

Start network

- Once the network is created, call mobleNetwork.Start() to start the network and thus the Android Ble Mesh Network Operations.

3.1.3 Nodes Enumeration API

Devices enumeration takes time

Once the network is started, you get the unprovisioned devices.

- MoBLE stack scans for incoming reports and provides collection of detected devices

Enumeration API

- Start devices enumeration (class mobleNetwork)
- Future<DeviceCollection> enumerateDevices()
  - Collection of devices in range is provided as instance of DeviceCollection
- Check if device with known bdaddr is in range (class DeviceCollection)
  - Device getDevice(bdaddr)
- Get subset of devices in range according filtering rules (class DeviceCollection)
  - Collection<Device> getDevices(int filter, Collection<String> dataBase)
    - BLE_ALL - all Bluetooth LE devices in range
    - BLE_MESH_ALL - all Bluetooth mesh devices
    - BLE_MESH_CONFIGURED - provisioned Bluetooth mesh devices
    - BLE_MESH_UNCONFIGURED - unprovisioned Bluetooth mesh devices
    - BLE_MESH_UNKNOWN - provisioned Bluetooth mesh devices with bdaddr not present in collection provided in argument dataBase
    - BLE_NO_MESH - Bluetooth LE devices but not Bluetooth mesh devices

3.1.4 Provisioning API - Introduction

Class mobleSettings is designed to provision unprovisioned devices within Android device range

- Call mobleNetwork.getSettings() to obtain instance of mobleSettings class

mobleSettings.provision(...) starts provisioning process

- Provisioning process is asynchronous
- Application shall implement interface mobleSettings.capabilitiesListener to confirm that proper Unprovisioned device is chosen for provisioning
- Application may implement interface mobleSettings.provisionerStateChanged to track current step of provisioning process
- Application may implement interface mobleSettings.onProvisionComplete to track completion of provisioning process: success or failure

mobleSettings.cancel() stops provisioning process

- If interface mobleSettings.onProvisionComplete is implemented then it is called back with failure or success status

3.2 Project Structure

3.2.1 Project folders

gradle (folder and related files) - Gradle automated build system

keys - folder with release keys that is used to sign the release APK

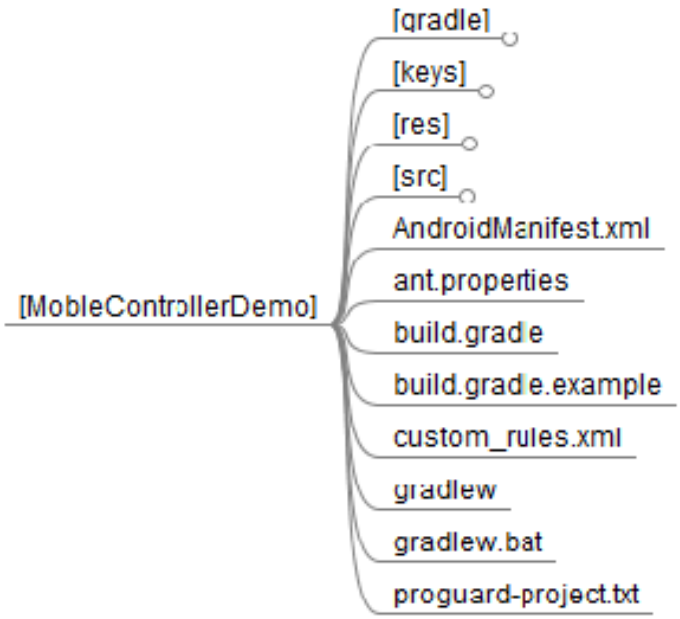
res - graphical and textual resources (images, layouts, menus, strings, etc.) of demo application

src - folder with source code files written in Java language

3.2.2 Temporary compilation folders

build - folder with intermediate Java files and compiled demo application

Figure 2. Android project structure



3.3 Project compilation: Android Studio

Step 1. Run Android Studio

Step 2. Click starting menu item “Open an existing Android Studio project”

Step 3. Select folder of project that need to be compiled:

– MobleControllerDemo

Step 4. Open Gradle tasks view

Step 5. Clean temporary files

– Click Gradle tree item “Tasks → build → clean”

Step 6. Compile (for MobleControllerDemo)
- Check that MobleLibrary is already compiled
- Click Gradle tree item “Tasks → build → assembleRelease” for release compilation
- Click Gradle tree item “Tasks → build → assembleDebug” for debug compilation

Step 7. Run (for MobleControllerDemo)
- Connect Android-based device which will be used for demo application running
- Click menu item “Run → Run ‘MobleControllerDemo’”
- Accept application installation on Android-based device

You need to enable the developer option on Android smartphones to install the app from the ADB.

Note: This method may differ depending on your smartphone (refer to its manual).

3.4 Project Compilation: command line

Step 1. Open command prompt in the folder of project that need to be compiled

Step 2. Clean temporary files
- Execute build script:
- “gradlew.bat clean”

Step 3. Compile (for MobleControllerDemo)
- Check whether the MobleLibrary is already present in the dir path
- Execute build script:
- “gradlew.bat assembleRelease” for release compilation
- “gradlew.bat assembleDebug” for debug compilation

Step 4. Install application (for MobleControllerDemo)
- Open folder with apk-file
- Run adb:
  - “adb.exe install BlueNRG-Mesh-release.apk” or
  - “adb.exe install BlueNRG-Mesh-debug.apk”

Step 5. If application is installed already and adb fails to update it then use key ‘-r’ with adb:
“adb.exe install -r BlueNRG-Mesh-release.apk”

4 APIs

4.1 How to use APIs

Step 1. Create a new mesh network to search for available BLE Mesh devices.

To create a new network, use createNetwork(). Once the network is created, its processing can be started or stopped with the start() and stop() APIs.

Step 2. Get the list of unprovisioned devices using enumerateDevices().

To provision a target device, use provision() API. API provision can be invoked through an instance of mobleSettings.

Step 3. Call provision method from the instance of mobleSettings.

On completion of provisioning, onProvisionComplete callback is called.

Step 4. Once the provisioning is done, use the Sending Configuration Model command for the configuration.

Step 5. Add Subscription and Set Publication.

Step 6. Once the targeted device is provisioned with the Android device, different APIs can be used to send Vendor Model, SIG Model, SetRemoteData, ReadRemoteData, Generic Model commands

RELATED LINKS

5.7 Toggle light for lighting device type on page 10

5.8 Read device type on page 10

6.16 Get settings on page 25

5 Code snippets and examples

5.1 Create Network

Create a new network with createNetwork() API

```
mNetwork = mobleNetwork.createNetwork(address);
```

Note: The default value of the address is 1, which is the ID set for the Android device at the time of createNetwork.

Pass this address to createNetwork to create a new mesh network:

```
mNetwork = mobleNetwork.createNetwork(androidAddress);
```

All subsequent nodes will be assigned a unicast device address starting from 2 as a mobleAddress object using the deviceAddress method.

5.2 Start Network

Start the MoBLE stack with start() API.

```
mobleStatus start() {
    mobleStatus res = mConfiguration.getNetwork().start(this);
    if (res.failed())    {
        return res;
    }
    mConfiguration.getNetwork().advise(mCallback);
    return mobleStatus.SUCCESS;
}
```

5.3 Stop Network

Stop the MoBLE stack with stop() API.

```
mobleStatus stop() {
    if (mConfiguration != null) {
        mConfiguration.getNetwork().unadvise(mCallback);
        return mConfiguration.getNetwork().stop();
    } else {
        trace("Network does not exist");
        return mobleStatus.FALSE;
    }
}
```

5.4 Advise(callback)

Register a callback with advise() API

```
((UserApplication) getApplication()).mConfiguration.getNetwork().advise(mOnDeviceAppearedCall
back);
((UserApplication) getApplication()).mConfiguration.getNetwork().advise(mProxyConnectionEvent
Callback);
((UserApplication) getApplication()).mConfiguration.getNetwork().advise(onDeviceRssiChangedCa
llback);
((UserApplication) getApplication()).mConfiguration.getNetwork().advise(onNetworkStatusChange
d);
```

5.5 Unadvise(callback)

For Deregistering a callback

Deregister a callback with unadvise() API

```
((UserApplication) activity.getApplication()).mConfiguration.getNetwork().unadvise(activity.m
OnDeviceAppearedCallback);
((UserApplication) activity.getApplication()).mConfiguration.getNetwork().unadvise(activity.o
nNetworkStatusChanged);
```

5.6 Provisioning

Invoking of Provisioning() API with an instance of settings

```
mSettings = ((UserApplication)getApplication()).mConfiguration.getNetwork().getSettings();
if (!checkConfiguration() || (null == mAutoAddress)) {
    mProgress.hide();
    makeToast("Network does not exist. Please create network");
    UserApplication.trace("Network does not exists");
}else {
   mSettings.provision(android.content.Context context,
   java.lang.String address,
   mobleAddress moble, int identifyDuration,
   mobleSettings.onProvisionComplete statusListener,
   mobleSettings.capabilitiesListener capabilitiesLstnr,
   mobleSettings.provisionerStateChanged psc,
   int completionDelay,
   CustomProvisioning cpr);

}
```

Callback will be called while provisioning is under process.

```
private final mobleSettings.onProvisionComplete mProvisionCallback = new mobleSettings.onProv
isionComplete() {
    @Override    public void onCompleted(byte status) {
        mProvisioningInProgress = false;
        if (status == mobleProvisioningStatus.SUCCESS) {
            provisioningStep++;
            mProvisionerStateChanged.onStateChanged(provisioningStep + 1, "");
            ((UserApplication) getApplication()).mConfiguration.getNetwork().advise(mDeviceDe
terminationCallback);
            mCookies = ((UserApplication) getApplication()).mConfiguration.getNetwork().getAp
plication().readRemoteData(mAutoDevice.getAddress(), Nucleo.APPLI_CMD_DEVICE_TYPE, 1, true);
        } else if (status == mobleProvisioningStatus.CANCEL) {
        } else {
            makeToast("Unable to configure " + mAutoAddress + ". Please check if device power
ed and try again");
            UserApplication.trace("Unable to configure " + mAutoAddress);
            updateRequest(false);
        }
    }
};
```

Callback will be called whenever new capabilities are found.

```
public final mobleSettings.capabilitiesListener mCapabilitiesLstnr = new
mobleSettings.capabilitiesListener()
{
@Override  public void onCapabilitiesReceived(mobleSettings.Identifier identifier, Byte eleme
ntsNumber)
{      //User code        mIdentifyDialog.createDialog(identifier);    }
};
```

Callback will be called whenever provisioner state changes

```
private final mobleSettings.provisionerStateChanged mProvisionerStateChanged = new mobleSetti
ngs.provisionerStateChanged() {
    @Override    public void onStateChanged(final int state, final String label) {
        provisioningStep = state;        runOnUiThread(new Runnable() {
            @Override            public void run() {
```

```
                mProgress.setProgress(state + 1, "Provisioning: \n" + label);
            }
        });
    }
};
```

RELATED LINKS

5.7 Toggle light for lighting device type

Setting data on remote device by using setRemoteData() API

```
mobleNetwork network = ((UserApplication) activity.getApplication()).mConfiguration.getNetwor
k();
network.getApplication().setRemoteData(mobleAddress.deviceAddress(addr), Nucleo.APPLI_CMD_LED
_CONTROL, 1,
new byte[]{Nucleo.APPLI_CMD_LED_TOGGLE}, true);
```

RELATED LINKS

4.1 How to use APIs on page 7

5.8 Read device type

Read data from remote device with readRemoteData() API.

```
network.getApplication().readRemoteData(mobleAddress.deviceAddress(addr), Nucleo.APPLI_CMD_DE
VICE, 1,
new byte[]{Nucleo.APPLI_CMD_DEVICE_BMESH_LIBVER}, true);
```

Note: Device Types are configured in the Device Firmware

RELATED LINKS

4.1 How to use APIs on page 7

5.9 Device RSSI changed

This is used to display the RSSI values of the Mesh nodes. The application signals RSSI change events.
Advise to onDeviceRssiChanged() callback

```
((UserApplication) getApplication()).mConfiguration.getNetwork().advise(onDeviceRssiChangedCa
llback);
```

onDeviceRssiChanged() callback implementation

```
private final defaultAppCallback onDeviceRssiChangedCallback =
        new defaultAppCallback(){
    @Override
    public void onDeviceRssiChanged(String bt_addr, int mRssi) {
        super.onDeviceRssiChanged(bt_addr, mRssi);
        Message msg = Message.obtain();
        msg.obj = new Object[] {bt_addr, mRssi};
        msg.what = MSG_UPDATE_RSSI;
        mHandler.sendMessage(msg);
    }
};
```

5.10 Send config model messages

This API gets the instance of the config model and sends a corresponding config model message (refer to Section B Config model commands for the list of config model messages supported).

5.11 Send generic model messages

It gets the instance of the generic model and sends a corresponding generic model message (refer to Section C Generic model commands for the list of generic model messages supported).

5.12 Send lighting model messages

It gets the instance of the lighting model and sends a corresponding lighting model message (refer to Section D Lighting model commands for the list of lighting model messages supported).

5.13 Configuration database sharing

Configuration database sharing can be done using the standard JSON way. You can also use other methods (plain text, sharepreference, etc.).

5.14 Troubleshooting

Creating/Loading configuration file

- Creating a file requires WRITE_EXTERNAL_STORAGE permission in manifest. If you receive a security exception during file writes, ensure the application has the necessary permissions.
- To load or save the configuration file, the file streams require application context. Ensure that the application is registered (the class extending Application class) in the manifest.

Exception: Unable to invoke virtual method

- Usually occurs while invoking an API with a new application reference. Do not make a new reference. Use getApplication() and type cast it to the child class.

Exception: Activity is not assignable to android.app.Activity

- The class should extend the Application class. Application cannot make UI changes; it requires an Activity.
- The child class should be registered by extending Application class in the manifest file.

Exception: Thread not able to update UI

- This exception occurs when a non UI thread tries to update the UI component of the activity. Use the handler to manage UI updates.

Exception: Security: coarse or fine location permission required.

- Permission for location is only required for Android 6 (SDK version 23) and above. To solve this issue, set the application targetSdkVersion to 22 or below (21 is recommended) in the build.gradle app module.

Exception: Illegal argument

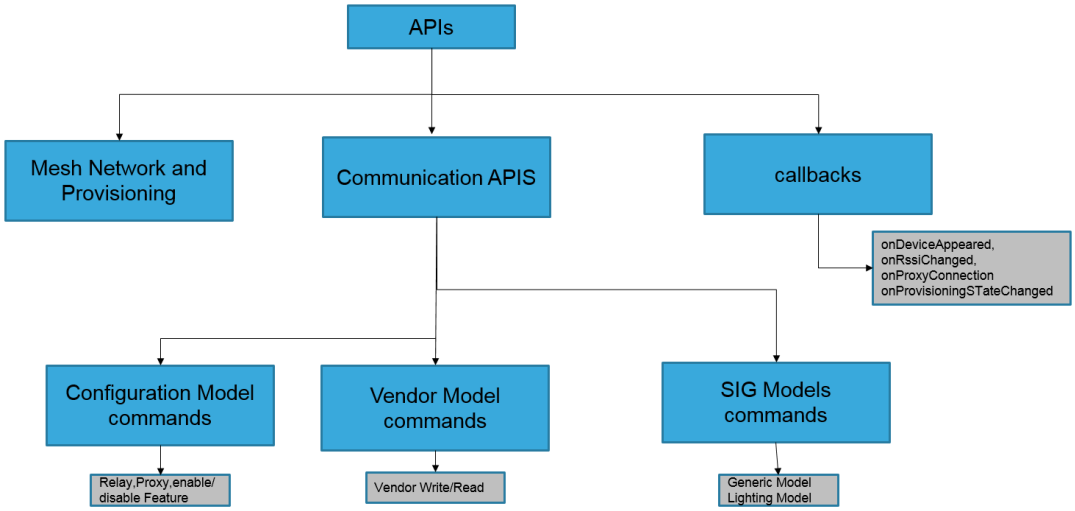
- The app may sometimes crash due to an old configuration of a device which need to be deleted. Go to Settings → Application → Select the app → Clear Cache & Data → uninstall, then reinstall the app.

6 BLE Mesh Android APIs

BLE Mesh APIs are broadly classified in:

- Mesh and networking APIs to start, stop,create and resume a network
- Communication APIs to send vendor and SIG model commands
- Callbacks to get asynchronous events from the Mesh network (for example, unprovisioned device appeared, response from the Mesh nodes, etc.)

Figure 3. BLE Mesh API overview



6.1 Create network

Figure 4. Create network (1 of 3)

public static mobleNetwork createNetwork(mobleAddress address) Creates new network with given address of Provisioner	
address	local address of Smartphone. Should not be equal to any element address or any other provisioner's address
returns	reference to the mobleNetwork object
Usage	private mobleNetwork mNetwork; mNetwork = mobleNetwork.createNetwork(address);

Figure 5. Create network (2 of 3)

public static mobleNetwork createNetwork(mobleAddress address, java.lang.String netKey, java.lang.String appKey) Creates new network with given address of Provisioner and NetWork Key and Application Key in string format(length :16 Bytes) e.g. “1431ea1afeb05224ab892a0217ccab38” .	
address	local address of Smartphone
netKey	Network Key. It has to be unique for each Network.Size : 16 Bytes
appKey	Application Key. It has to be unique for each Network.Size : 16bytes
returns	reference to the mobleNetwork object
Usage	private mobleNetwork mNetwork; mNetwork = mobleNetwork.createNetwork(address, “1431ea1afeb05224ab892a0217ccab38”,” 6da9698c95f500e4edce3bb47f92754f ”);

Figure 6. Create network (3 of 3)

public static mobleNetwork createNetwork(mobleAddress address, byte[] netKey, byte[] appKey) Creates new network with given address of Provisioner ,NetWork Key and Application Key in byte format(length :16 Bytes) e.g. 1431ea1afeb05224ab892a0217ccab38.	
address	local address of Smartphone
netKey	Unique Network Key.
appKey	Unique Application Key.
returns	reference to the mobleNetwork object
Usage	private mobleNetwork mNetwork; mNetwork = mobleNetwork.createNetwork(address, new 1431ea1afeb05224ab892a0217ccab38”.toByteArray(),” 6da9698c95f500e4edce3bb47f92754f.toByteArray());

6.2 Start network

Figure 7. Start network

public mobleStatus start(android.content.Context context) Starts functioning of Android BLE Mesh stack.	
Context	Object of User Application
returns	status of operation.
Usage	private mobleNetwork mNetwork; mNetwork.start() ;
Prerequisite	A BLE Mesh network must already be created using createNetwork() API.

6.3 Stop network

Figure 8. Stop network

public mobleStatus stop(android.content.Context context) Stops functioning of Android BLE Mesh stack.	
Context	Object of User Application
returns	status of operation.
usage	private mobleNetwork mNetwork; mNetwork.stop() ;
Prerequisite	A BLE Mesh network must already be created using createNetwork() API.

6.4 Restore network

Figure 9. Restore network

public static mobleNetwork restoreNetwork(mobleAddress address, String netKey , String appKey, String meshdata) throws IOException Restores the previously formed Mesh Network using the Provisioner Address,Network Key,Application Key and Mesh JSON data.	
Address	local address of Smartphone
netKey	Unique Network Key
appKey	Unique Application Key
returns	reference to the mobleNetwork object
Meshdata	JSON String for the Mesh Data

6.5 Device enumeration

Figure 10. Device enumeration

public Future<DeviceCollection> enumerateDevices() Initiates devices enumeration.Enumeration may require some time therefore result is provided as Future object.	
return	Furure object with collection of enumerated devices
Usage	futureED = ((UserApplication) getApplication()).mConfiguration.getNetwork().enumerateDevices();

6.6 Provision

```
public boolean provision(android.content.Context context,
                                java.lang.String address,
                                mobleAddress moble, int identifyDuration,
                                mobleSettings.onProvisionComplete statusListener,
                                mobleSettings.capabilitiesListener capabilitiesLstnr,
                                  mobleSettings, provisionerStateChanged psc,
```

```
                                    int completionDelay,
                                    CustomProvisioning cpr)
```

Table 1. Provision

Parameter	Description
Context	Object of User Application
returns	Status of the provisioning operation
address	MAC address of the Target Device (AA:BB:CC:DD:EE:FF)
mobleAddress	MobleAddress address of the Target Device (1,2,3,4...). It should not be equal to the Provisioner's address or any previously allocated Device Address.
Identifyduration	Duration in seconds up to which the Target Device can identify itself (by blinking, for example) - e.g. 10 secs.
statusListener	Callback when the provisioning is complete.
capabilitiesLstnr	Callback when the node capabilities are received.
psc	Callback when the provisioner state has been changed. It can be used to update the progess on the GUI
completionDelay	Duration after which the configuration starts after re-connection to the proxy service
cpr	Custom provisioning instance, currently kept null. (Not required)
returns	Status of provisioning
Usage	private mobleSettings mSettings; mSettings = ((UserApplication) getApplication ()).mConfiguration.getNetwork().getSettings(); mSettings.provision(MainActivity.this, address, mobleAddress 10, statusListener, capabilitiesLstnr, psc, completionDelay,//10 seconds cpr);

6.7 Add subscription/group

```
public mobleStatus addGroup(android.content.Context context,
                                    mobleAddress address,
                                    mobleAddress mElementaddress,
                                    mobleAddress group,
ConfigurationModelClient.ConfigModelSubscriptionStatusCallback listener)
```

Table 2. Add subscription/group

Parameter	Description
Context	Object of User Application
returns	Status of the operation

Parameter	Description
address	Moble address of the node or element
mElementaddress	Moble address of the element
group	Moble address of the group
Listener	Callback of the subscription command
Usage of Listener	app.mConfiguration.getNetwork().getSettings().addGroup (MainActivity.this, Nodeaddress, elementAddress, groupAddress mSubscriptionListener);
Interface	`ConfigurationModelClient.ConfigModel SubscriptionStatusCallback`

Usage of Interface	public final ConfigurationModelClient.ConfigModelSubscriptionStatusCallback
	mSubscriptionListener = new
	ConfigurationModelClient.ConfigModelSubscriptionStatusCallback() {
	@Override
	public void onModelSubscriptionStatus(boolean timeout,
	ApplicationParameters.Status status,
	ApplicationParameters.Address address,
	ApplicationParameters.Address subAddress,
	ApplicationParameters.GenericModelID model) {
	//User code, status is true on Success, Failure on Timeout
	}
	);

6.8 Set publication

```
public boolean setPublicationAddress(android.content.Context context,
mobleAddress nodeAddress,
mobleAddress elementAddress,
int publishAddress,
 ConfigurationModelClient.ConfigModelPublicationStatusCallback callback)
```

Table 3. Set publication

Parameter	Description
Context	Object of User Application
nodeAddress	Moble Address of the node
mElementaddress	Moble Address of the element
publishAddress	Moble Address for the publication (Address can be of any provisioned node or any group)
callback	Callback of the Publication Command
returns	Status of the operation

Parameter	Description
Usage	app.mConfiguration.getNetwork().getSettings(). setPublicationAddress(context, nodeAddress, elementAddress, publishAddress, callback)

Figure 11. Reliable response for publication status

Usage	app.mConfiguration.getNetwork().getSettings(). setPublicationAddress(context, nodeAddress, elementAddress, publishAddress, callback)

6.9 Generic model commands

It is used to send messages for the generic model.

Figure 12. GenericOnOff - set

public boolean setGenericOnOff(boolean reliable, ApplicationParameters.Address address, ApplicationParameters.OnOff state, ApplicationParameters.TID tid, ApplicationParameters.Time transitionTime, ApplicationParameters.Delay delay, GenericOnOffModelClient.GenericOnOffStatusCallback callback)	
reliable	Whether a response is required or Not
address	Address of the Node or Element
state	OnOFF State – ENABLED(1) or DISABLED(0)
tid	Transaction id
transitionTime	Transition Time
delay	Delay after which the effect will take place on the Target
callback	Callback of the Message Response
returns	Status of the operation
Usage	UserApplication app = (UserApplication) context.getApplicationContext(); ((UserApplication) ((MainActivity) context).getApplication()).mConfiguration.getNetwork().getOnOffModel().setGenericOnOff(true , Address state, tid, transitionTime, delay, Utils.mOnOffCallback);

Figure 13. GenericonOff - get

public boolean getGenericOnOff(ApplicationParameters.Address address, GenericOnOffModelClient.GenericOnOffStatusCallback callback)	
Context	Object of User Application
address	Moble Address of the Node
callback	Callback of the Publication
returns	Status of the operation

Usage	((UserApplication) ((MainActivity)
	context).getApplication()).mConfiguration.getNetwork().getGenericOnOff
	(ApplicationParameters.Address address,
	GenericOnOffStatusCallback callback) {
	…
	}

Figure 14. Reliable response for GenericOnOffModel

Interface	public static interface GenericOnOffModelClient.GenericOnOffStatusCallback

Usage	public static final GenericOnOffModelClient.GenericOnOffStatusCallback mOnOffCallback = new GenericOnOffModelClient.GenericOnOffStatusCallback() {
	@Override
	public void onOnOffStatus(boolean timeout,
	ApplicationParameters.OnOff state,
	ApplicationParameters.OnOff targetState,
	ApplicationParameters.Time remainingTime, ApplicationParameters.Address nodeAddress) {
	if (timeout) {
	UserApplication.trace("Generic OnOff Timeout");
	} else {
	}
	}
	};

Figure 15. Generic Level set

public boolean setGenericLevel(boolean reliable, ApplicationParameters.Address address, ApplicationParameters.Level level, ApplicationParameters.TID tid, ApplicationParameters.Time transitionTime, ApplicationParameters.Delay delay, GenericLevelStatusCallback callback)	
reliable	Whether a response is required or Not
address	status of operation.
level	Level Value
tid	Transaction Id
transitionTime	Transition Time
delay	delay Time
returns	Status of the operation
Usage	mGenericLevelModel = ((UserApplication) ((MainActivity) context).getApplication()).mConfiguration.getNetwork().getLevelModel(); mGenericLevelModel.setGenericLevel(true, elementAddress, level, new ApplicationParameters.TID(2), null,//transition time null,//delay mLevelCallback);

Figure 16. Generic Level get

public boolean getGenericLevel(ApplicationParameters.Address address, GenericLevelStatusCallback callback)	
address	Moble Address of the Node or Element
callback	Callback of the Publication
returns	status of operation.

Usage	mGenericLevelModel = ((UserApplication) ((MainActivity) context).getApplication()).mConfiguration.getNetwork().getLevelModel();
	
	mGenericLevelModel.getGenericLevel(adress, mLevelCallback);

Figure 17. Reliable response for GenericModel Level

Interface	public interface GenericLevelStatusCallback

Usage	private final GenericLevelModelClient.GenericLevelStatusCallback mLevelCallback = new GenericLevelModelClient.GenericLevelStatusCallback() {
	@Override
	public void onLevelStatus(boolean timeout,
	ApplicationParameters.Level level,
	ApplicationParameters.Level targetLevel,
	ApplicationParameters.Time remainingTime) {
	if (timeout) {
	UserApplication.trace("Generic Level Timeout");
	} else {
	mDimming = level.getValue();
	}
	}
	};

6.10 Lighting model commands

It is used to send messages for the lighting model.

Figure 18. Lighting model – LightLightness Set (1 of 2)

public boolean setLightnessLevel(boolean reliable, ApplicationParameters.Address address, ApplicationParameters.Lightness lightness, ApplicationParameters.TID tid, ApplicationParameters.Delay delay, LightLightnessModelClient.LightingLightnessStatusCallback callback)	
Reliable	Whether a response is required or Not
Address	Target address(device/Element)
Lightness	Light Lightness value (0- 0xFFFF)
Tid	Transaction ID(unique transaction Id number)
Delay	Delay value
callback	Reliable response

Figure 19. Lighting model – LightLightness Set (2 of 2)

Usage	mLightingLightnessModel = network.getLightnessModel();
	
	mLightingLightnessModel. setLightnessLevel(true, TEST_M_ADDRESS, lightness, tid, delay, mLightnessStatusCallback);

Figure 20. Lighting model – LightLightness Get

public boolean getLightnessLevel(ApplicationParameters.Address address, LightLightnessModelClient.LightingLightnessStatusCallback callback	
address	Moble Address of the Node
callback	Reliable response of the command
Return type	Succes or Failure
Usage	mLightingLightnessModel.getLightnessLevel(TEST_M_ADDRESS, mLightnessStatusCallback);

Figure 21. Reliable response for LightingLightness level

public static interface	LightLightnessModelClient.LightingLightnessStatusCallback

Usage	private final LightLightnessModelClient.LightingLightnessStatusCallback mLightnessStatusCallback = new LightLightnessModelClient.LightingLightnessStatusCallback() {
	@Override
	public void onLightnessStatus(boolean timeout, ApplicationParameters.Lightness lightness, ApplicationParameters.Lightness lightness1, ApplicationParameters.Time time) {
	
	if (timeout) {
	UserApplication.trace("Lighting Lightness Timeout");
	} else {
	UserApplication.trace("Lighting Lightness status = SUCCESS ");
	}
	}
	};

Figure 22. Lighting Model – Light CTL Set (1 of 2)

public boolean setLightCTL(boolean reliable, ApplicationParameters.Address address, ApplicationParameters.Lightness lightness, ApplicationParameters.Temperature temperature, ApplicationParameters.TemperatureDeltaUV deltaUV, ApplicationParameters.TID tid, ApplicationParameters.Delay delay, LightCTLModelClient.LightCTLStatusCallback callback)	
Reliable	Whether a response is required or Not
Address	Target address
Lightness	Light Lightness Value
Temperature	Temperature value
deltaUV	deltaUV value
Tid	Trasaction Id
Delay	Delay
Callback	Response for the command

Figure 23. Lighting Model – Light CTL Set (2 of 2)

public static interface	LightLightnessModelClient.LightingLightnessStatusCallback

Usage	mLightCTLModelClient = network.getLightnessCTLModel();
	
	mLightCTLModelClient.setLightCTL(true,TEST_M_ADDRESS,CTL_lightness,CTL_temperature,CTL_temperatureDeltaUV,CTL_tid,CTL_transitionTime,CTL_del, mLightCTLStatusCallback);

Figure 24. Lighting Model – Light CTL Get

public boolean getLightCTL(ApplicationParameters.Address address, LightCTLModelClient.LightCTLStatusCallback callback)	
Returns	status of operation.
address	Moble Address of the Node
callback	Callback of the command

Usage	mLightCTLModelClient = network.getLightnessCTLModel();
	
	mLightCTLModelClient.getLightCTL(TEST_M_ADDRESS, mLightCTLStatusCallback);

Figure 25. Reliable response for LightingCTL

public static interface	public static interface LightCTLModelClient.LightCTLStatusCallback

Usage	private final LightCTLModelClient.LightCTLStatusCallback mLightCTLStatusCallback = new LightCTLModelClient.LightCTLStatusCallback() {
	@Override
	public void onLightCTLStatus(boolean timeout,
	ApplicationParameters.Lightness presentCTLLightness,
	ApplicationParameters.Temperature presentCTLtemperature,
	ApplicationParameters.Lightness targetCTLLightness,
	ApplicationParameters.Temperature targetCTLtemperature,
	ApplicationParameters.Time remainingTime) {
	
	if (timeout) {
	UserApplication.trace("Lighting Lightness CTL Timeout");
	} else {
	
	UserApplication.trace("Lighting Lightness CTL status = SUCCESS ");
	}
	}
	};

```
public boolean setLightCTL(boolean reliable,
ApplicationParameters.Address address,
 ApplicationParameters.Lightness lightness,
 ApplicationParameters.Temperature temperature, ApplicationParameters.TemperatureDeltaUV delt
aUV,
 ApplicationParameters.TID tid,
 ApplicationParameters.Delay delay,
LightCTLModelClient.LightCTLStatusCallback callback)
```

Table 4. Lighting Model – Light CTL Set

Parameter	Description
Reliable	Whether a response is required or not
Address	Target address
Lightness	Light Lightness value
Temperature	Temperature value
deltaUV	deltaUV value
Tid	Transaction ID
Delay	Delay
Callback	Response for the command

Parameter	Description
public static interface	`LightLightnessModelClient.LightingLightnessStatusCallback`
Usage	mLightCTLModelClient = network.getLightnessCTLModel(); mLightCTLModelClient.setLightCTL(true,TEST_M_ADDRESS,CTL_lightness,CTL_temperature,CTL_temperatureDeltaUV,CTL_tid,CTL_transitionTime,CTL_del, mLightCTLStatusCallback);

Figure 26. Lighting Model – Light CTL Temperature Get

public boolean getLightCTLTemperature(ApplicationParameters.Address address, LightCTLModelClient.LightCTLTemperatureStatusCallback callback)	
returns	status of operation.
address	Moble Address of the Node
callback	Callback of the Publication

Usage	mLightCTLModelClient = network.getLightnessCTLModel();
	
	mLightCTLModelClient.getLightCTLTemperature(TEST_M_ADDRESS, mLightCTLTemperatureStatusCallback);

Figure 27. Reliable response for LightingCTLTemperature

public static interface	LightCTLModelClient.LightCTLTemperatureStatusCallback

Usage	mLightCTLModelClient = network.getLightnessCTLModel();
	
	private final LightCTLModelClient.LightCTLTemperatureStatusCallback mLightCTLTemperatureStatusCallback = new LightCTLModelClient.LightCTLTemperatureStatusCallback() {
	@Override
	public void onLightCTLTemperatureStatus(boolean timeout,
	ApplicationParameters.Temperature presentCTLtemperature,
	ApplicationParameters.TemperatureDeltaUV presentCTLDeltaUV,
	ApplicationParameters.Temperature targetCTLtemperature,
	ApplicationParameters.TemperatureDeltaUV targetCTLDeltaUV,
	ApplicationParameters.Time remainingTime){
	
	if (timeout) {
	//failure
	} else {
	
	//success
	}
	}
	};

6.11 Vendor model commands

It is used to send messages for the vendor model.

Figure 28. Vendor Model – ReadRemoteData

java.lang.Object readRemoteData(mobleAddress peer, int opcode, int count, byte[] data, boolean with_response)	
Peer	Target Address
opcode	Opcode to be sent
Count	Count (redundant)
Data	Byte data to be sent
Boolean_with_response	Reliable or unreliable(always True)
Usage	mobleNetwork network = ((UserApplication) ((MainActivity) context).getApplication()).mConfiguration.getNetwork(); network.getApplication().readRemoteData(mobleAddress.deviceAddress(addr), Nucleo.APPLI_CMD_DEVICE, 1, new byte[]{Nucleo.APPLI_CMD_DEVICE_BMESH_LIBVER}, true);

Figure 29. Vendor Model – SetRemoteData

java.lang.Object setRemoteData(mobleAddress peer, int opcode, int count, byte[] data, boolean with_response)	
Peer	Target Address
Opcode	Opcode to be sent
Count	Count (redundant)
Data	Byte data to be sent
boolean_with_response	Reliable or unreliable
Usage	network.getApplication().setRemoteData(addre, Nucleo.APPLI_CMD_LED_CONTROL, 1, new byte[]{Nucleo.APPLI_CMD_LED_ON}, ((MainActivity) context).rel_unrel);

Figure 30. Reliable response for Vendor Model

java.lang.Object readRemoteData(mobleAddress peer, int opcode, int count, byte[] data, boolean with_response)	
Peer	Target Address
opcode	Opcode to be sent
Count	Count (redundant)
Data	Byte data to be sent
Boolean_with_response	Reliable or unreliable(always True)
Usage	mobleNetwork network = ((UserApplication) ((MainActivity) context).getApplication()).mConfiguration.getNetwork(); network.getApplication().readRemoteData(mobleAddress.deviceAddress(addr), Nucleo.APPLI_CMD_DEVICE, 1, new byte[]{Nucleo.APPLI_CMD_DEVICE_BMESH_LIBVER}, true);

6.12 Moble(Mesh) Addresses

6.12.1 Device address

```
public static mobleAddress deviceAddress(int address)
```

Construct a new mobleAddress object for device address.

Returns

- mobleAddress → Created Object.

Parameters

- address → int: device network address.

Prerequisites

- configuration → Configuration: an instance of configuration must be initialized.

6.12.2 Group address

```
public static mobleAddress groupAddress(int address)
```

Construct a new mobleAddress object for a group address.

Returns

- mobleAddress → Created Object.

Parameters

- address → int: group network address.

Prerequisites

- configuration → Configuration: an instance of configuration must be initialized.

6.13 Advise/unadvise callbacks

API used to register and unregister the asynchronous events in a Mesh network.

Figure 31. Register/unregister callbacks

void advise(mobleLayerApplication application) void unadvise(mobleLayerApplication application)	
Calback	Application Callbacks
Usage	Register : ((UserApplication) getApplication()).mConfiguration.getNetwork().advise(<device-Callbacks>); Unregister : ((UserApplication) getApplication()).mConfiguration.getNetwork().unadvise(<device-Callbacks>);

6.14 Enumerate devices

Figure 32. Enumerate devices

public Future<DeviceCollection> enumerateDevices() Initiates devices enumeration.Enumeration may require some time therefore result is provided as Future object.	
return	Furure object with collection of enumerated devices
Usage	futureED = ((UserApplication) getApplication()).mConfiguration.getNetwork().enumerateDevices();

6.15 Get devices

```
Collection<Devices> getDevices(int filter, Collection<String> database)
```

Returns

- Collection → Returns collection of devices.

Parameters

- Filterint → bit mask from flags.
- database → Collection<String> database of mac addresses of BLE devices.

Prerequisites

- configuration → Configuration: an instance of configuration must be initialized.
- mobleNetwork → mobleNetwork: an instance of network must be created.

6.16 Get settings

```
mobleSettings getSettings()
```

Get settings object for device provisioning.

Returns

- mobleSettings → Returns reference to the object.

Prerequisites

- configuration → Configuration: an instance of configuration must be initialized.
- mobleNetwork → mobleNetwork: an instance of network must be created.

RELATED LINKS

4.1 How to use APIs on page 7

7 Callbacks

7.1 onProvisionComplete

Figure 33. OnProvisionComplete

public static interface mobleSettings.onProvisionComplete Called if provisioning process is completed.	
Status	Success or Failure of the provisioning process

Usage	// Listener for requests to MoBLE Settings. Handles autoConfigurecompleted
	public final mobleSettings.onProvisionComplete mProvisionCallback = new mobleSettings.onProvisionComplete() {
	@Override
	public void onCompleted(byte status) {
	//do the configuration Step
	}
	};

7.2 onProxyConnectionEvent

Figure 34. onProxyConnectionEvent

public void onProxyConnectionEvent(boolean process, String proxyAddress) Called when a proxy connection is made	
Process	Proxy Connection Process
proxyAddress	Mac address of the proxy Device

Usage	public final defaultAppCallback mProxyConnectionEventCallback = new defaultAppCallback() {
	@Override
	public void onProxyConnectionEvent(boolean process, String proxyAddress) {
	mProxyAddress = proxyAddress;
	//User code
	}
	};

7.3 onDeviceAppeared

Figure 35. onDeviceAppeared

void onDeviceAppeared(java.lang.String bt_addr) Called if detected unconfigured MOBLE device.	
Bt_addr	Bluetooth Address of the Device

Usage	public defaultAppCallback mOnDeviceAppearedCallback = new defaultAppCallback() {
	@Override
	public void onDeviceAppeared(String bt_addr) {
	//mac address of the Mesh Device
	}
	};

7.4 onDeviceRSSIChanged

Figure 36. **onDeviceRSSIChanged**

void onDeviceAppeared(java.lang.String bt_addr) Called if detected unconfigured MOBLE device.	
Bt_addr	Bluetooth Address of the Device

Usage	public defaultAppCallback mOnDeviceAppearedCallback = new defaultAppCallback() {
	@Override
	public void onDeviceAppeared(String bt_addr) {
	//mac address of the Mesh Device
	
	}
	};

7.5 onProvisionStatechanged

Figure 37. **onProvisionStatechanged**

public static interface mobleSettings.provisionerStateChanged Called if provisioner state has been changed.	
State	Provisioning Current State
Label	Label of the State

Usage	public final mobleSettings.provisionerStateChanged mProvisionerStateChanged = new mobleSettings.provisionerStateChanged() {
	@Override
	public void onStateChanged(final int state, final String label) {
	;
	}
	};

8 Device configuration

Mesh Network configuration is stored in the JSON file which can be shared via e-mail or cloud.

Its a way of storing the complete network configuration (like Network Key, Application Key, Device Keys, Groups formed), publication and subscription address and model information.

8.1 Mesh configuration database

Figure 38. Mesh network configuration database: JSON (1 of 2)

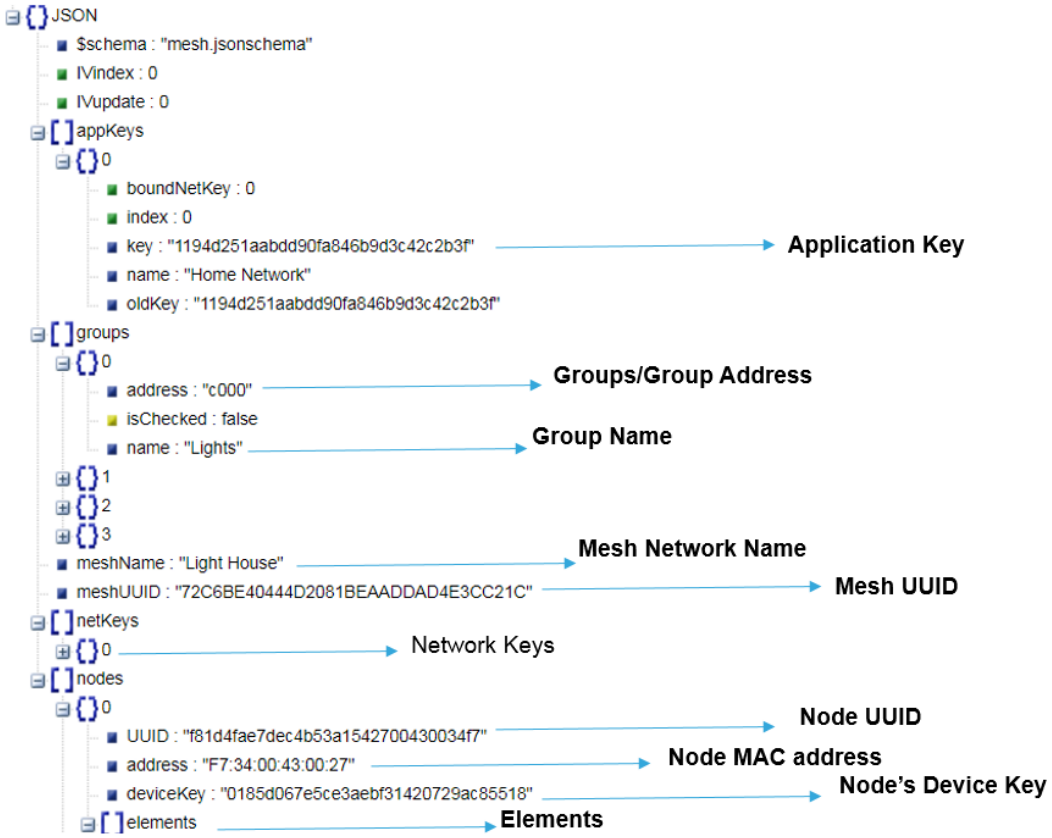
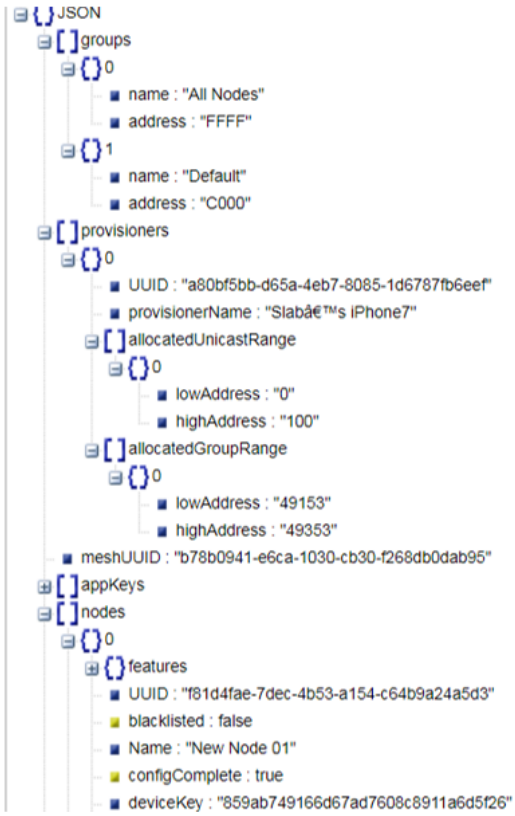


Figure 39. **Mesh network configuration database: JSON (2 of 2)**



A Licensing and other information

Developer-friendly license terms

The initial BlueNRG-Mesh is built over Motorola’s Mesh Over Bluetooth Low Energy (MoBLE) technology.

The present solution involving both the Mesh library and applications is developed and maintained solely by STMicroelectronics.

B Config model commands

- Config Model: Config AppKey Add
- Config Model: Config AppKey Delete
- Config Model: Config AppKey Get
- Config Model: Config AppKey List
- Config Model: Config AppKey Status
- Config Model: Config AppKey Update
- Config Model: Config Beacon Get
- Config Model: Config Beacon Set
- Config Model: Config Composition Data Get
- Config Model: Config Model Publication Set
- Config Model: Config Default TTL Get
- Config Model: Config Default TTL Set
- Config Model: Config Default TTL Status
- Config Model: Config Friend Get
- Config Model: Config Friend Set
- Config Model : Config Friend Status
- Config Model: Config GATT Proxy Get
- Config Model: Config GATT Proxy Set
- Config Model: Config GATT Proxy Status
- Config Model: Config Model App Bind
- Config Model: Config Model App Status
- Config Model: Config Model App Unbind
- Config Model : Config Model Publication Get
- Config Model: Config Model Publication Status
- Config Model: Config Model Publication Virtual Address Set
- Config Model: Config Model Subscription Add
- Config Model: Config Model Subscription Delete
- Config Model: Config Model Subscription Delete All
- Config Model: Config Model Subscription Overwrite
- Config Model: Config Model Subscription Status
- Config Model: Config Model Subscription Virtual Address Add
- Config Model: Config Model Subscription Virtual Address Delete
- Config Model: Config Model Subscription Virtual Address Overwrite
- Config Model: Config NetKey Add
- Config Model: Config NetKey Delete
- Config Model: Config NetKey Get
- Config Model: Config NetKey List
- Config Model: Config NetKey Status
- Config Model: Config NetKey Update
- Config Model: Config Network Transmit Get
- Config Model: Config Network Transmit Set
- Config Model: Config Network Transmit Status
- Config Model: Config Node Identity Get
- Config Model: Config Node Identity Set
- Config Model: Config Node Identity Status
- Config Model: Config Node Reset
- Config Model: Config Node Reset Status

- Config Model: Config Relay Get
- Config Model: Config Relay Set
- Config Model: Config Relay Status
- Config Model: Config SIG Model App Get
- Config Model: Config SIG Model App List
- Config Model: Config SIG Model Subscription Get
- Config Model: Config SIG Model Subscription List
- Config Model: Config Vendor Model App Get
- Config Model: Config Vendor Model App List
- Config Model: Config Vendor Model Subscription Get
- Config Model: Config Vendor Model Subscription List

C Generic model commands

- Generic OnOff Model: Generic OnOff Get
- Generic OnOff Model: Generic OnOff Set
- Generic OnOff Model: Generic Level Get
- Generic OnOff Model: Generic Level Set

D Lighting model commands

- Light Lightness Model: Light Lightness Get
- Light Lightness Model: Light Lightness Set
- Light Lightness Model: Light Lightness Set Unacknowledged
- Light CTL Model: Light CTL Get
- Light CTL Model: Light CTL Set
- Light CTL Model: Light CTL Set Unacknowledged
- Light CTL Model: Light CTL Temperature Get
- Light CTL Model : Light CTL Temperature Set
- Light CTL Model: Light CTL Temperature Set Unacknowledged

Revision history

Table 5. **Document revision history**

Date	Version	Changes
01-Feb-2018	1	Initial release.
20-Sep-2018	2	Updated Section 1 Features of the BLE Mesh library, Section 3.1.1 Initialization API - Mesh network operations, Section 3.1.2 Allowable Mesh network operations after API initialization, Section 4.1 How to use APIs, Section 5.9 Device RSSI changed, Section 6 BLE Mesh Android APIs, Section 6.1 Create network, Section 6.2 Start network, Section 6.3 Stop network, Section 6.4 Restore network, Section 6.13 Advise/unadvise callbacks, Section 6.14 Enumerate devices, Section 7.1 onProvisionComplete, Section 7.2 onProxyConnectionEvent, Section 7.3 onDeviceAppeared, Section 7.4 onDeviceRSSIChanged, Section 7.5 onProvisionStatechanged, Section 8 Device configuration and Section 8.1 Mesh configuration database. Added Section 5.10 Send config model messages, Section 5.11 Send generic model messages, Section 5.12 Send lighting model messages, Section 5.13 Configuration database sharing, Section 6.5 Device enumeration, Section 6.6 Provision, Section 6.7 Add subscription/group, Section 6.8 Set publication, Section 6.9 Generic model commands, Section 6.10 Lighting model commands, Section 6.11 Vendor model commands, Section 6.12.2 Group address, Section B Config model commands, Section C Generic model commands and Section D Lighting model commands.

Contents

List of figures

List of tables