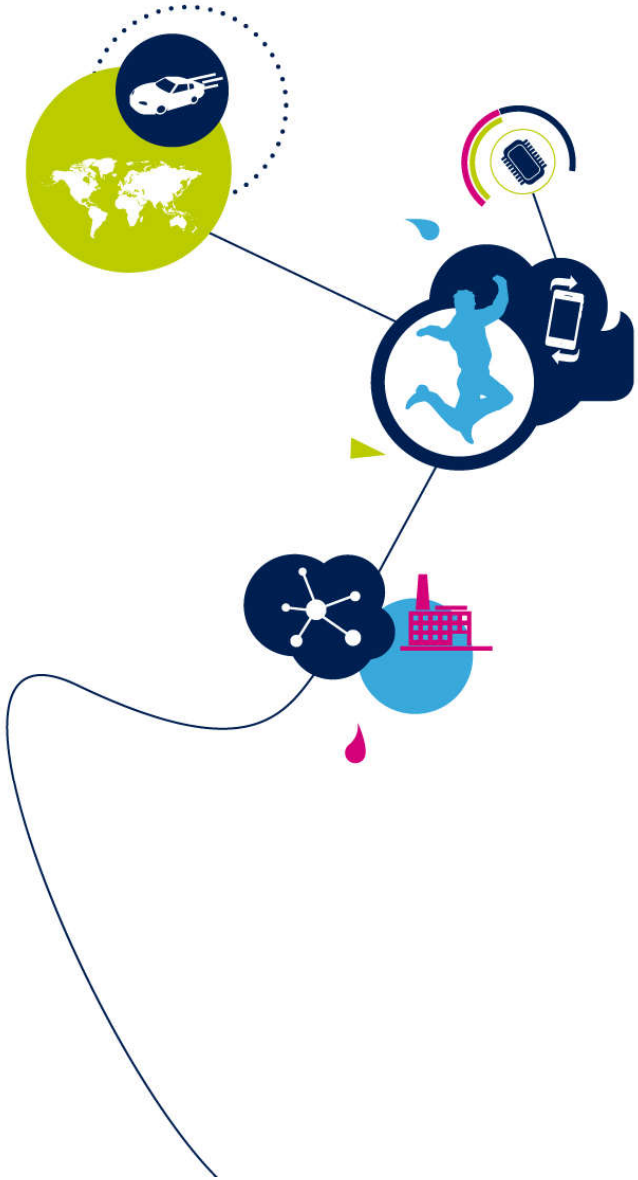
Bluetooth Low Energy Mesh Android App



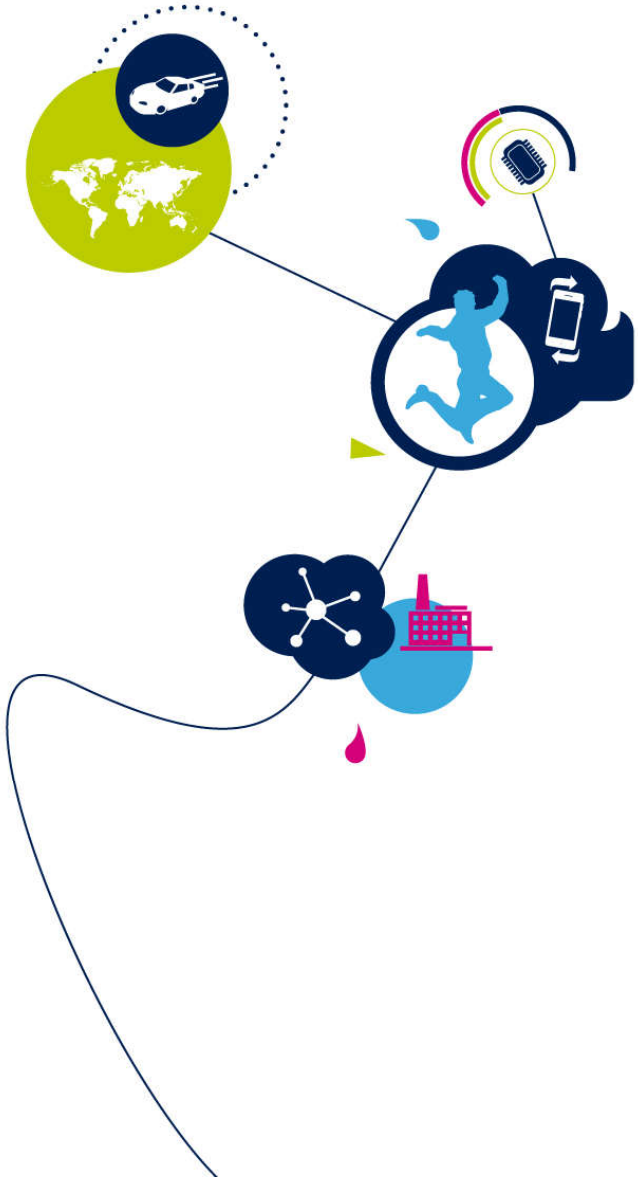
life.augmented

低功耗蓝牙Mesh
安卓App

Saurabh RAWAT
saurabh.rawat@st.com

Kamaldeep BANSAL
kamaldeep.bansal@st.com

SRA, Noida
Date: 29th August 2018



life.augmented

BLE* Mesh Android App

- Bluetooth Low Energy Mesh Introduction
- BLE Mesh over Android Framework
- App Walkthrough
- API Walkthrough and Hands on Session

life.augmented

*BLE: Bluetooth Low Energy

BLE* Mesh 安卓 App

- 低功耗蓝牙Mesh 介绍
- 基于安卓架构的BLE Mesh
- App使用流程
- API 使用流程以及实战任务

*BLE: Bluetooth Low Energy

life.augmented

Video & Demo

- https://www.youtube.com/watch?v=5EKoEcSnXP0

视频 & 示例

- https://www.youtube.com/watch?v=5EKoEcSnXP0

Bluetooth Mesh official announcement

July 18, 2017 - Bluetooth SIG Announces Mesh Networking Capability

Brings proven, global interoperability and the mature, trusted ecosystem of Bluetooth technology to industrial-grade device networks

industrial-grade solution

Bluetooth mesh uniquely meets the reliability, scalability and security requirements of building and factory automation markets that demand true industrial-grade solutions.

- **Reliability:** Enables inherently self-healing networks with no single points of failure
- **Scalability:** Supports thousands of nodes with industrial-level performance
- **Security:** Provides industrial-grade security for protection against all known attacks

proven, global interoperability

Only Bluetooth mesh delivers the proven multi-vendor interoperability that enables markets to flourish and assures that products from different vendors across the globe work together.

- **A full-stack solution:** A unique full stack approach that defines everything from the low-level radio to the high-level application layer, ensuring all levels of the technology are fully specified
- **An interop-centric spec:** Comprehensive interoperability testing conducted prior to specification release, not after
- **Time-tested tools and processes:** A 20-year history of delivering the qualification tools and processes necessary to ensure global, multi-vendor interoperability

mature, trusted technology

Bluetooth®

The value-added capabilities, mature ecosystem and global brand awareness that Bluetooth wireless technology provides enable the creation of much richer solutions with a faster time to market.

- **Value-added services:** A mesh network built on Bluetooth can also provide localized information, asset tracking and way-finding services
- **A mature ecosystem:** The best enabling technology, along with the development and test tools and services needed to shrink your time to market
- **Global brand awareness:** A trusted global brand that stands for simple, secure wireless connectivity

life.augmented

Bluetooth Mesh 官方通告

2017年7月18日 - Bluetooth SIG 通告 Mesh 组网

给工业级设备组网带来可验证的，全球互通和成熟的，可信任的生态系统

industrial-grade solution

Bluetooth mesh uniquely meets the reliability, scalability and security requirements of building and factory automation markets that demand true industrial-grade solutions.

- **Reliability:** Enables inherently self-healing networks with no single points of failure
- **Scalability:** Supports thousands of nodes with industrial-level performance
- **Security:** Provides industrial-grade security for protection against all known attacks

proven, global interoperability

Only Bluetooth mesh delivers the proven multi-vendor interoperability that enables markets to flourish and assures that products from different vendors across the globe work together.

- **A full-stack solution:** A unique full stack approach that defines everything from the low-level radio to the high-level application layer, ensuring all levels of the technology are fully specified
- **An interop-centric spec:** Comprehensive interoperability testing conducted prior to specification release, not after
- **Time-tested tools and processes:** A 20-year history of delivering the qualification tools and processes necessary to ensure global, multi-vendor interoperability

mature, trusted technology

Bluetooth®

The value-added capabilities, mature ecosystem and global brand awareness that Bluetooth wireless technology provides enable the creation of much richer solutions with a faster time to market.

- **Value-added services:** A mesh network built on Bluetooth can also provide localized information, asset tracking and way-finding services
- **A mature ecosystem:** The best enabling technology, along with the development and test tools and services needed to shrink your time to market
- **Global brand awareness:** A trusted global brand that stands for simple, secure wireless connectivity

life.augmented

BlueNRG-Mesh is here

PRESS RELEASES

Bluetooth SIG Announces Mesh Networking Capability

7/18/2017 8:00:00 AM

Bluetooth® companies supporting the launch of Bluetooth mesh networking

3M, AIR CABLE, ARM, BeON home, cortet™ by CEL, CYPRESS, dialog SEMICONDUCTOR, ElectricBench, ERICSSON, FULHAM, Gooee, Gunitech, hellohub, Imagination, muRata INNOVATOR IN ELECTRONICS, QUALCOMM, SILICON LABS, SILVAIR, smart and green, life.augmented, TOSHIBA Leading Innovation, TRLink, WISILICA

life.augmented

BlueNRG

life.augmented

BlueNRG-Mesh 在这里

PRESS RELEASES

Bluetooth SIG Announces Mesh Networking Capability

7/18/2017 8:00:00 AM

Bluetooth® companies supporting the launch of Bluetooth mesh networking

3M, AIR CABLE, ARM, BeON home, cortet™ by CEL, CYPRESS, dialog SEMICONDUCTOR, ElectricBench, ERICSSON, FULHAM, Gooee, Gunitech, hellohub, Imagination, muRata INNOVATOR IN ELECTRONICS, QUALCOMM, SILICON LABS, SILVAIR, smart and green, life.augmented, TOSHIBA Leading Innovation, TRLINK, WISILICA

BlueNRG

life.augmented

Extending Bluetooth Capabilities

PAIRING
one-to-one

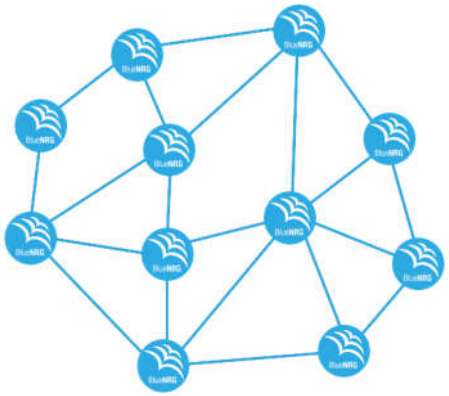
DATA TRANSFER
- Sports & fitness devices
- Health and wellness devices
- Peripherals and accessories

BROADCASTING
one-to-many

LOCALIZED INFORMATION
- Point of interest beacons
- Item finding beacons
- Way finding beacons

NEW

MESH
many-to-many



LARGE DEVICE NETWORKS
- Building automation
- Wireless sensor networks
- Asset tracking

life.augmented

扩展的蓝牙能力

配对
单点到单点

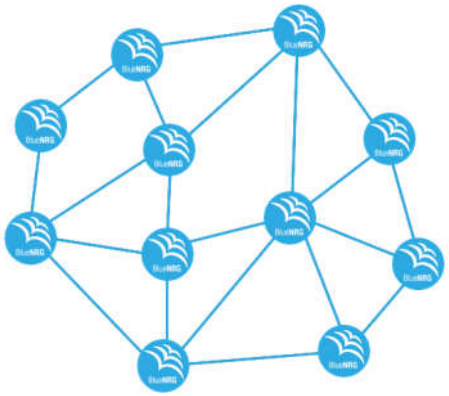
数据传输
- 运动穿戴装置
- 健康装置
- 外设和附件

广播
单点到多点

本地信息
- 单点接入关注的Beacon装置
- 条目搜寻 beacons
- 方式搜寻 beacons

NEW

MESH
多点到多点

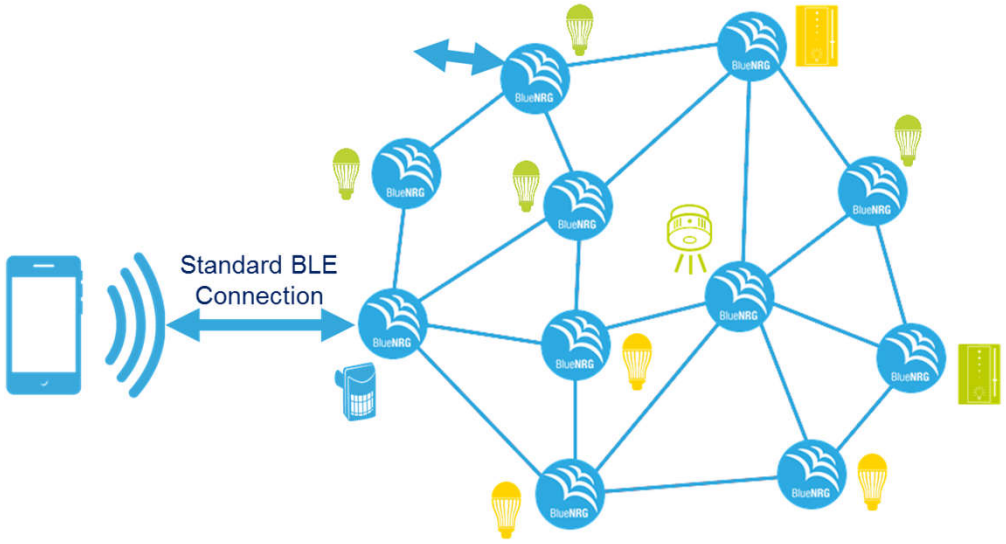


大设备网络
- 搭建自动化
- 无线传感器网络
- 资产追踪

life.augmented

Bluetooth LE Mesh Overview

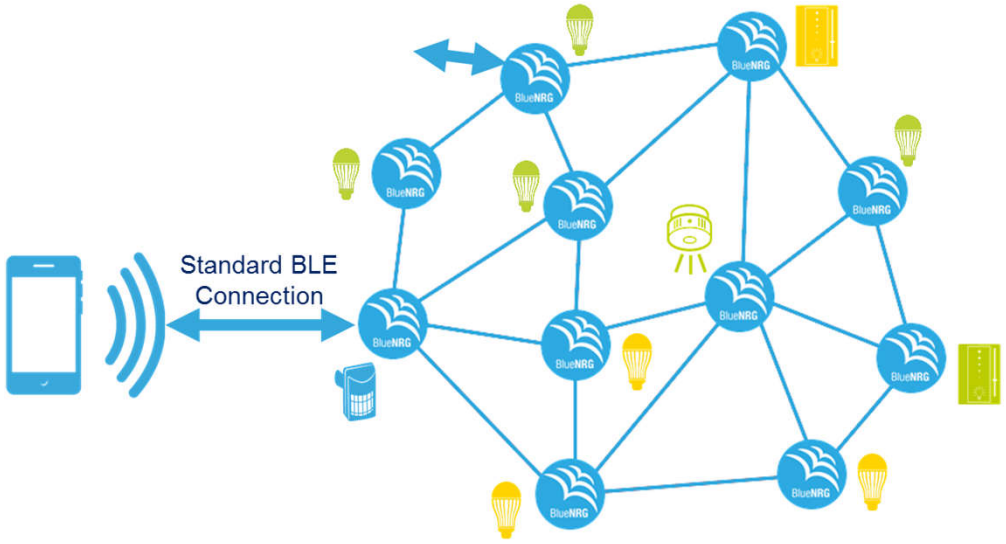
- Mesh created over BLE connected nodes
- Key Features
  - Mesh Profile specifications defined by Bluetooth SIG
  - Data transfers are Flooding based
  - Multicast data transfer (Custom groups defined by end-user)
  - Continue to work even when one or more devices are moved or stop operating
  - Low power consumption
- Security
  - All messages in the network are authenticated
  - Messages are sent with sequence numbers to protect against replay attacks



life.augmented

蓝牙 Mesh 简介

- Mesh 是基于连接BLE的节点
- 关键特性
  - Mesh Profile 规格是由蓝牙SIG 定义的
  - 数据传输时基于Flooding 型
  - 多广播数据传输 (用户定义分组)
  - 即使有一个或多个设备拿走或停止操作，系统仍然可持续工作
  - 低功耗
- 安全
  - 网络内所有的信息都需要认证
  - 防止重传信息发送的时候都带有序列号

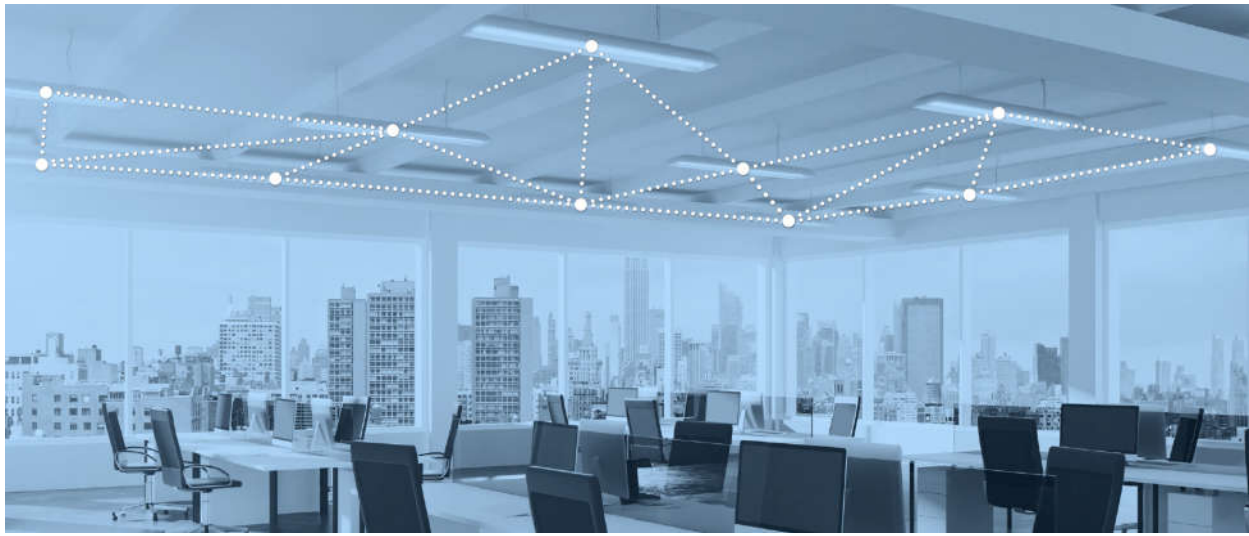


life.augmented

Bluetooth Mesh Applications

The Bluetooth SIG MESH Specification intends to extend the capabilities of Bluetooth Smart chips to answer **more and more complex applications**. The protocol has been developed with the **Smart Lighting industry** in mind.

- Building automation
- Wireless sensor networks
- Asset tracking
- Smart home
- Street lighting
- Industry 4.0
- …

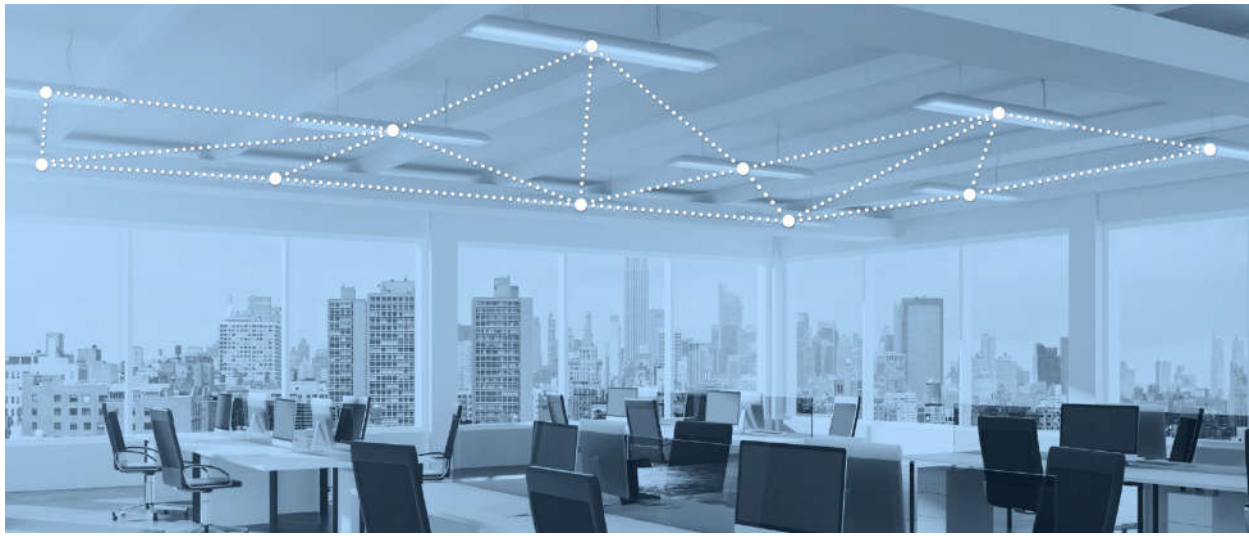


life.augmented

蓝牙 Mesh 应用

蓝牙特别兴趣小组Mesh规范 扩展了Bluetooth Smart复杂的应用场景. 该协议规范同智慧照明工业的理念发展而来。

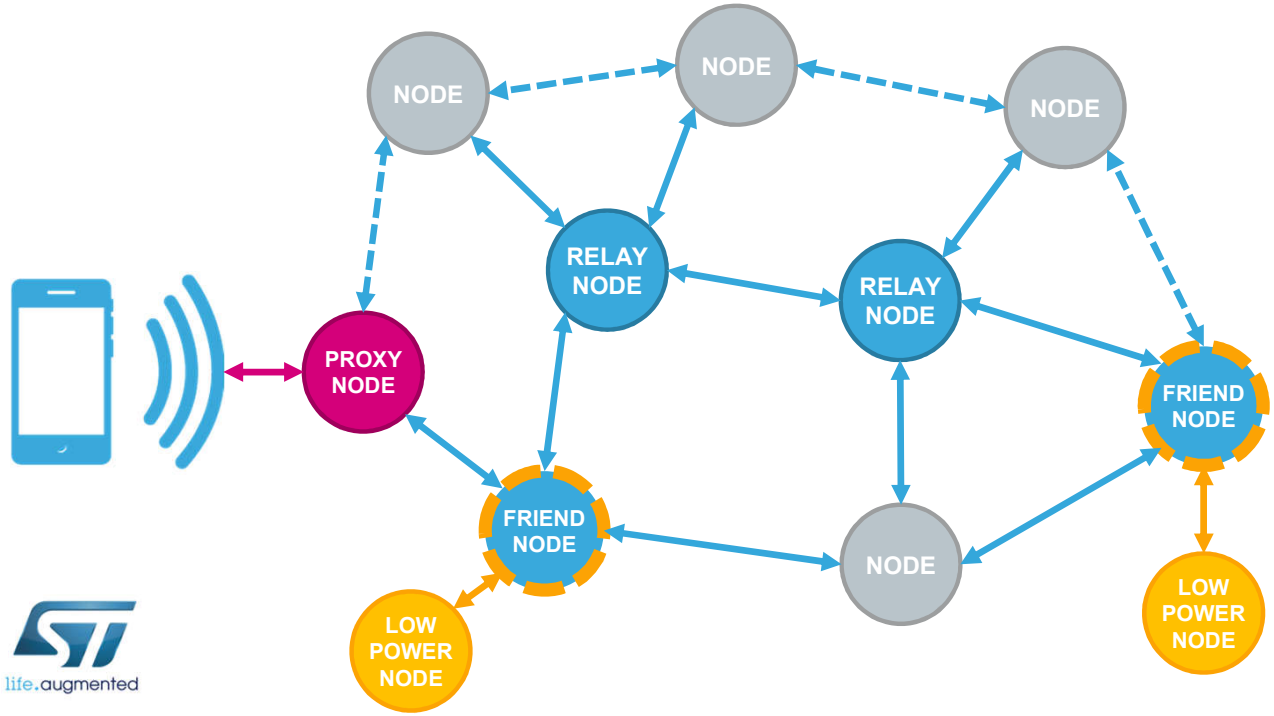
- 搭建自动化
- 无线传感网络
- 设备追踪
- 智能家庭
- 道路照明
- 工业4.0
- …

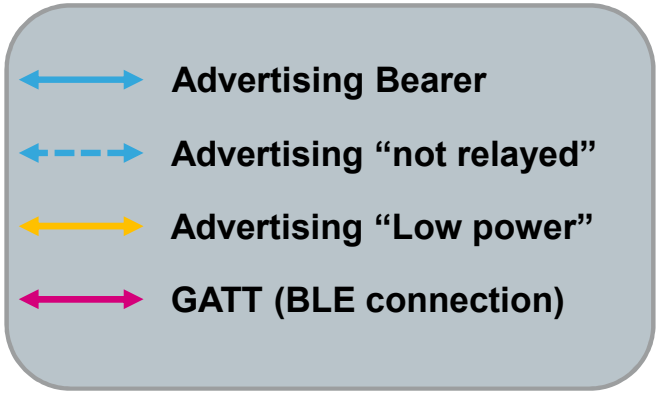


Bluetooth Mesh Topology

Managed Flooding

The Bluetooth Mesh working group chose for mesh network mechanism a **flooding protocol**. Compared to routed protocols, it is **much more simpler** to deploy. To stay efficient, the BLE Mesh take advantage of a **managed flooding network**.

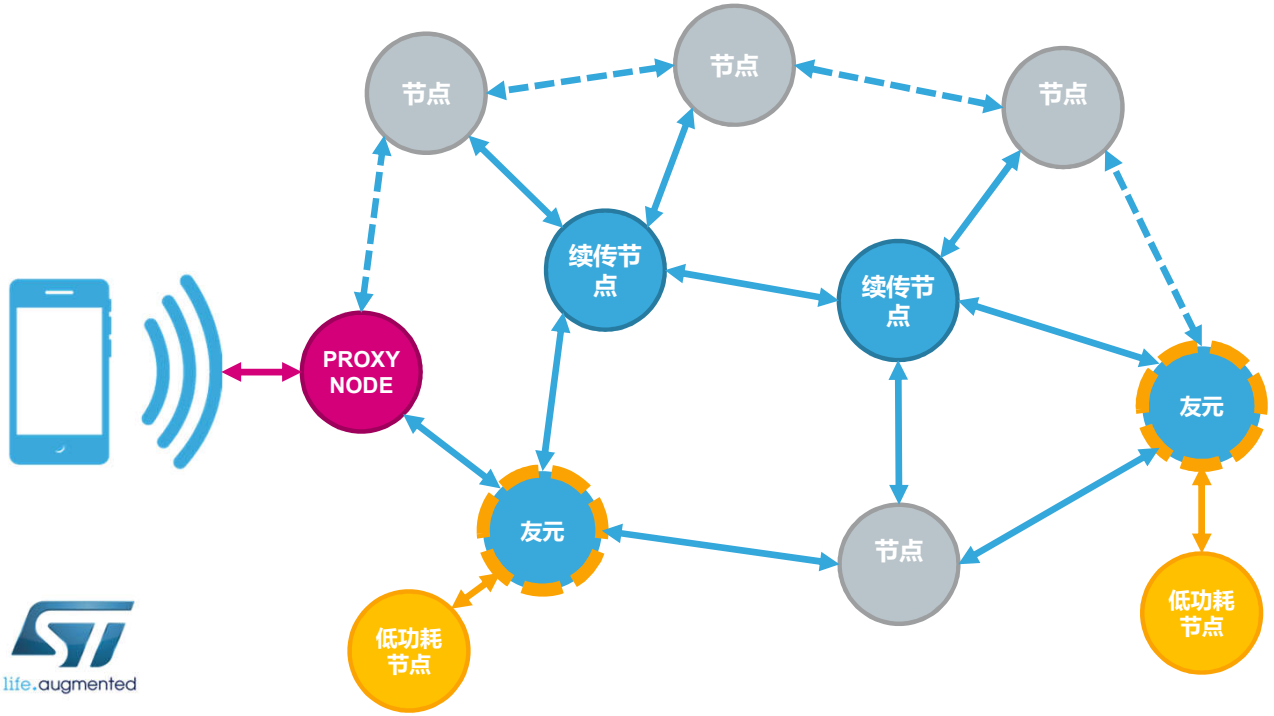




life.augmented

蓝牙Mesh 拓扑结构
管理的泛洪

蓝牙 Mesh 工作组选择的 mesh 是协议是 **flooding 类型**. 对比routed 协议, 它**更加简单** 适用于开发. 为了保持效率，BLE Mesh 吸收了 **可管理 flooding 网络**的优势.



GATT - Generic Attribute Profile

life.augmented

BlueNRG-Mesh

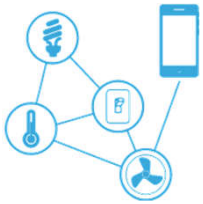
Easily connecting appliances to iOS/Android, out-of-the-box

life.augmented

BlueNRG-Mesh

简易连接到iOS/Android, 拆箱即用

life.augmented



STSW-BNRG-Mesh

- SDK includes: Firmware, Android, iOS
- www.st.com/blemesh

GET SOFTWARE

Part Number	Software Version	Marketing Status	Supplier	
BlueNRG-Mesh for Android		Active	ST	GO TO SITE
BlueNRG-Mesh for iOS		Active	ST	GO TO SITE
STSW-BNRG-Mesh	1.03.000	Active	ST	Get Software

life.augmented

life.augmented

Search

Menu

Home › Embedded Software › Wireless Connectivity Software › STSW-BNRG-Mesh

STSW-BNRG-Mesh ACTIVE

Mesh over Bluetooth Low Energy

Download Databrief

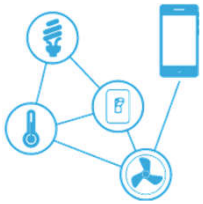
QUICK VIEW | RESOURCES | TOOLS AND SOFTWARE | GET SOFTWARE

BlueNRG-Mesh is a software solution for connecting multiple BLE (Bluetooth low energy) devices in Mesh networks for Internet of Things (IoT) solutions. It enables true two-way communication between Bluetooth-enabled devices in powerful, secure, integrated and range-extending Mesh networks.

The solution is compatible with the ST BlueNRG product family range.

Key Features

- Mesh network with Bluetooth low energy (BLE) nodes enabling communication between a BLE device and a Smartphone
- Control and monitor applications involving short packets and infrequent communication
- Advertising packets used for data communication using managed flooding method
- Multi-hop data transmission
- Network node support up to 32,767 nodes and up to 126 hops
- Multiple communication scenario
  - Smartphone to node communication with unicast addressing
  - Smartphone to node communication with multicast (Group) addressing
  - Smartphone to node communication with broadcast addressing
  - Node to node communication
- Secure communication
  - Devices added to a network are provisioned using proven security algorithms using 256-bit elliptic curves
  - All messages in the network are encrypted with AES-128 CCM mode
  - Privacy through obfuscation



STSW-BNRG-Mesh

- SDK 包含: 固件, Android, iOS
- www.st.com/blemesh

GET SOFTWARE

Part Number	Software Version	Marketing Status	Supplier	
BlueNRG-Mesh for Android		Active	ST	GO TO SITE
BlueNRG-Mesh for iOS		Active	ST	GO TO SITE
STSW-BNRG-Mesh	1.03.000	Active	ST	Get Software

life.augmented

life.augmented

Menu

Search

Home › Embedded Software › Wireless Connectivity Software › STSW-BNRG-Mesh

STSW-BNRG-Mesh ACTIVE

Mesh over Bluetooth Low Energy

Download Databrief

QUICK VIEW | RESOURCES | TOOLS AND SOFTWARE | GET SOFTWARE

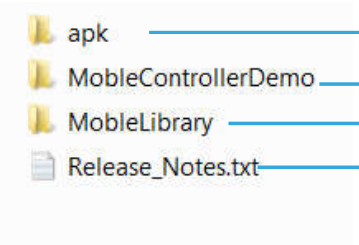
BlueNRG-Mesh is a software solution for connecting multiple BLE (Bluetooth low energy) devices in Mesh networks for Internet of Things (IoT) solutions. It enables true two-way communication between Bluetooth-enabled devices in powerful, secure, integrated and range-extending Mesh networks.

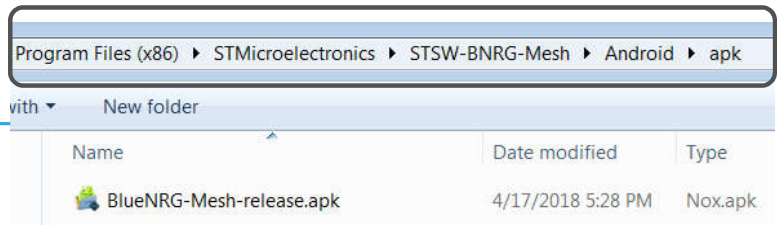
The solution is compatible with the ST BlueNRG product family range.

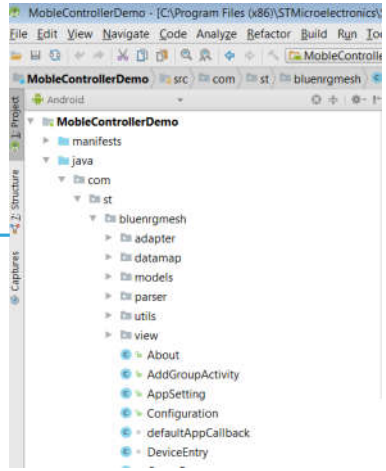
Key Features

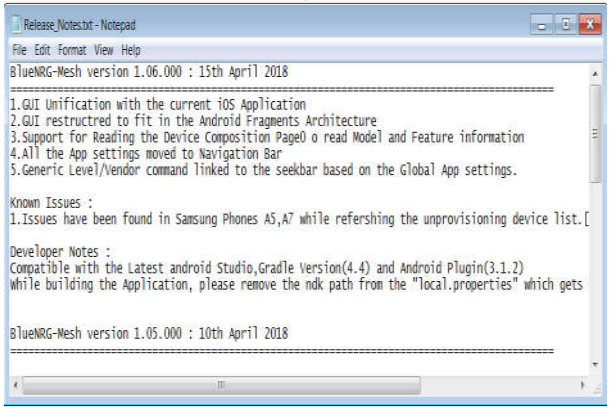
- Mesh network with Bluetooth low energy (BLE) nodes enabling communication between a BLE device and a Smartphone
- Control and monitor applications involving short packets and infrequent communication
- Advertising packets used for data communication using managed flooding method
- Multi-hop data transmission
- Network node support up to 32,767 nodes and up to 126 hops
- Multiple communication scenario
  - Smartphone to node communication with unicast addressing
  - Smartphone to node communication with multicast (Group) addressing
  - Smartphone to node communication with broadcast addressing
  - Node to node communication
- Secure communication
  - Devices added to a network are provisioned using proven security algorithms using 256-bit elliptic curves
  - All messages in the network are encrypted with AES-128 CCM mode
  - Privacy through obfuscation

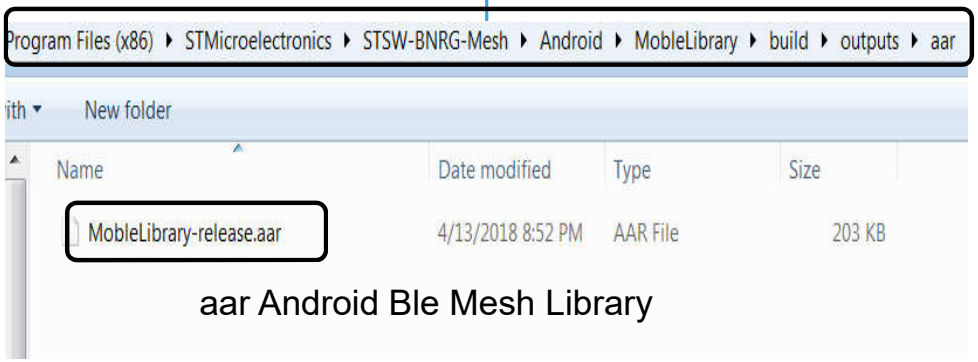
BLE Mesh Android App and SDK







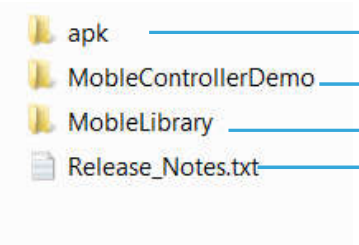


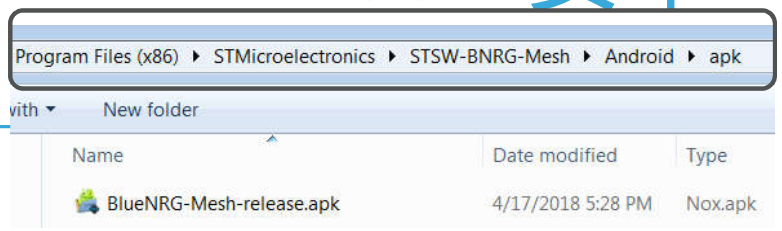


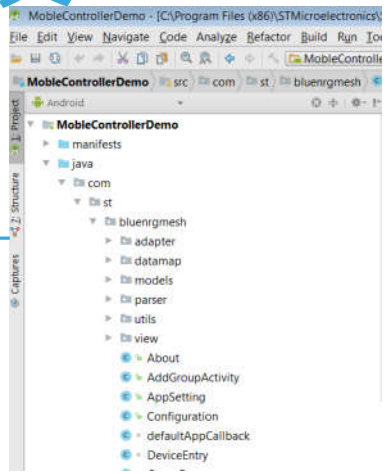
aar Android Ble Mesh Library

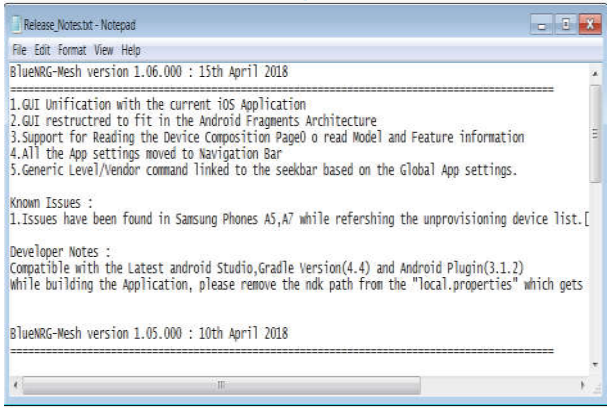
life.augmented

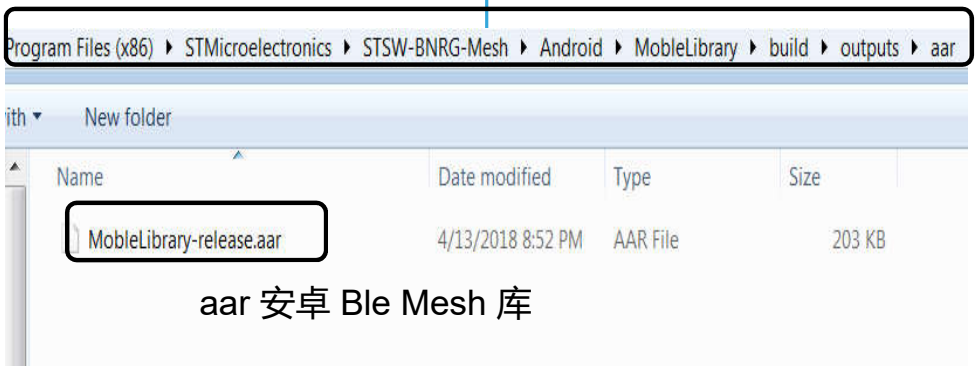
BLE Mesh 安卓 App 以及 SDK











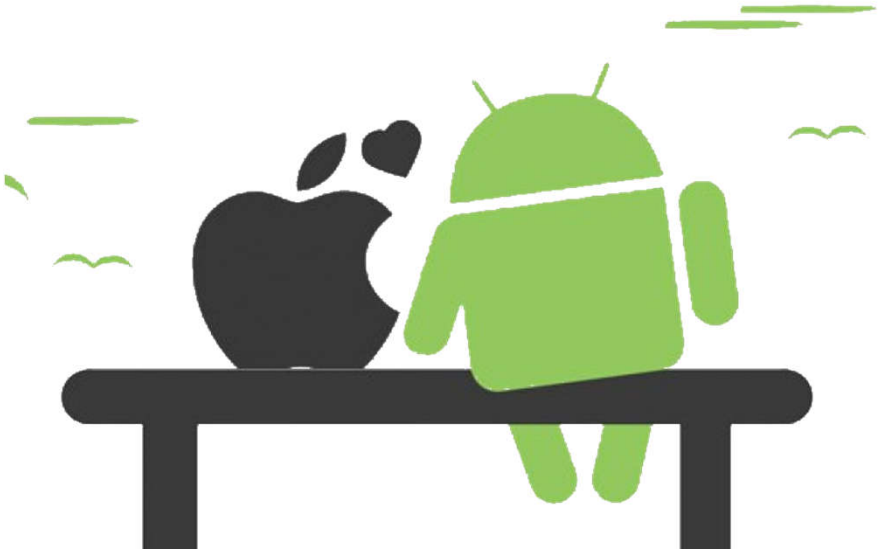
aar 安卓 Ble Mesh 库

life.augmented

BlueNRG-Mesh On Store

Download on the App Store

GET IT ON Google play

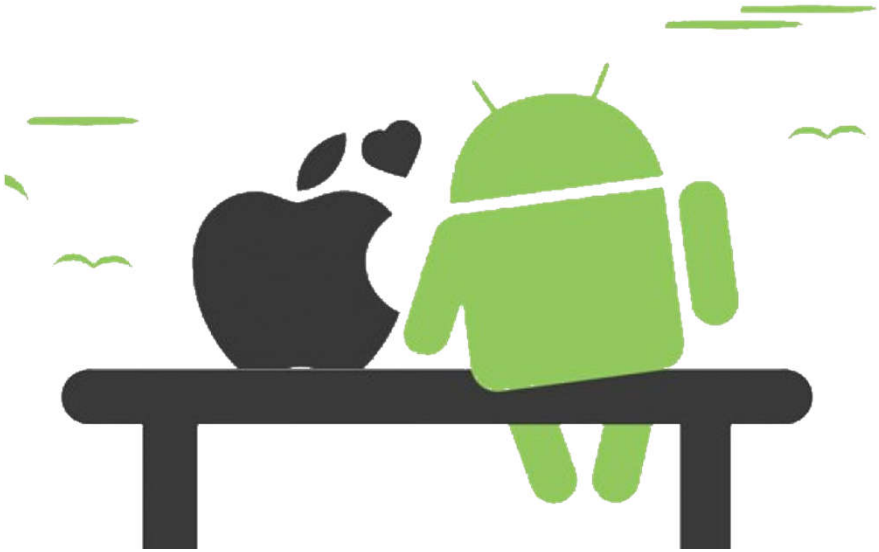


life.augmented

BlueNRG-Mesh 商店

Download on the App Store

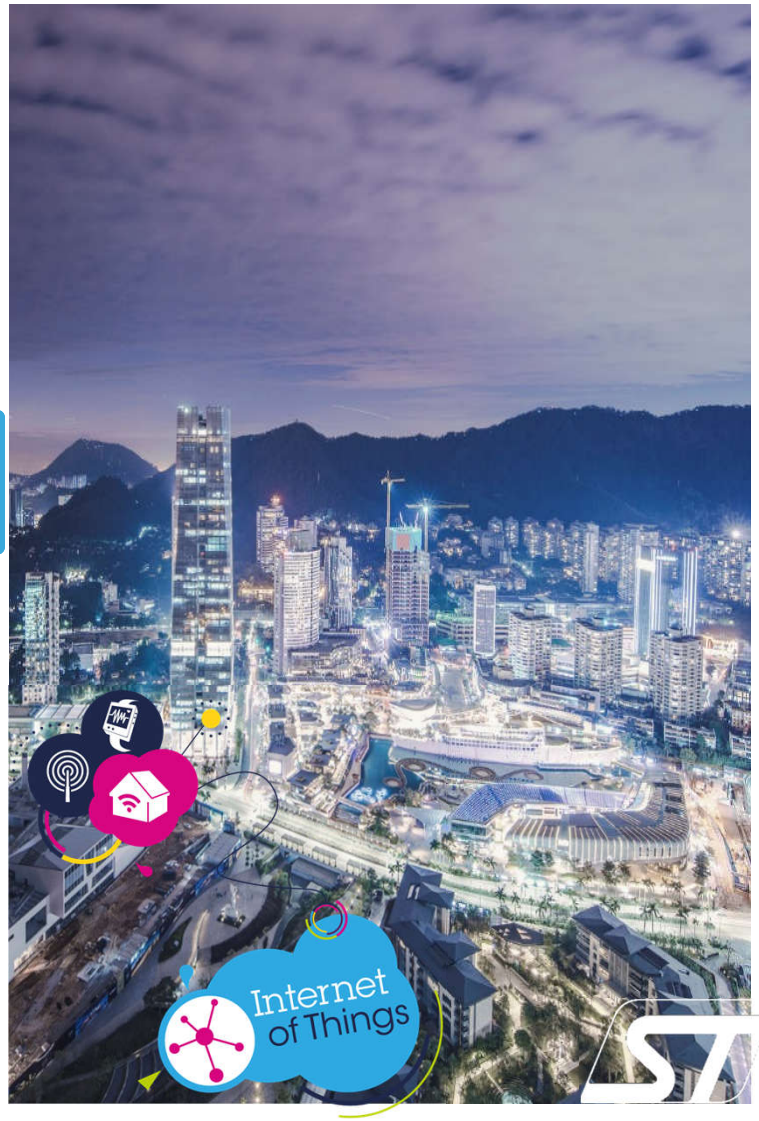
GET IT ON Google play



life.augmented

BLE Mesh Android App

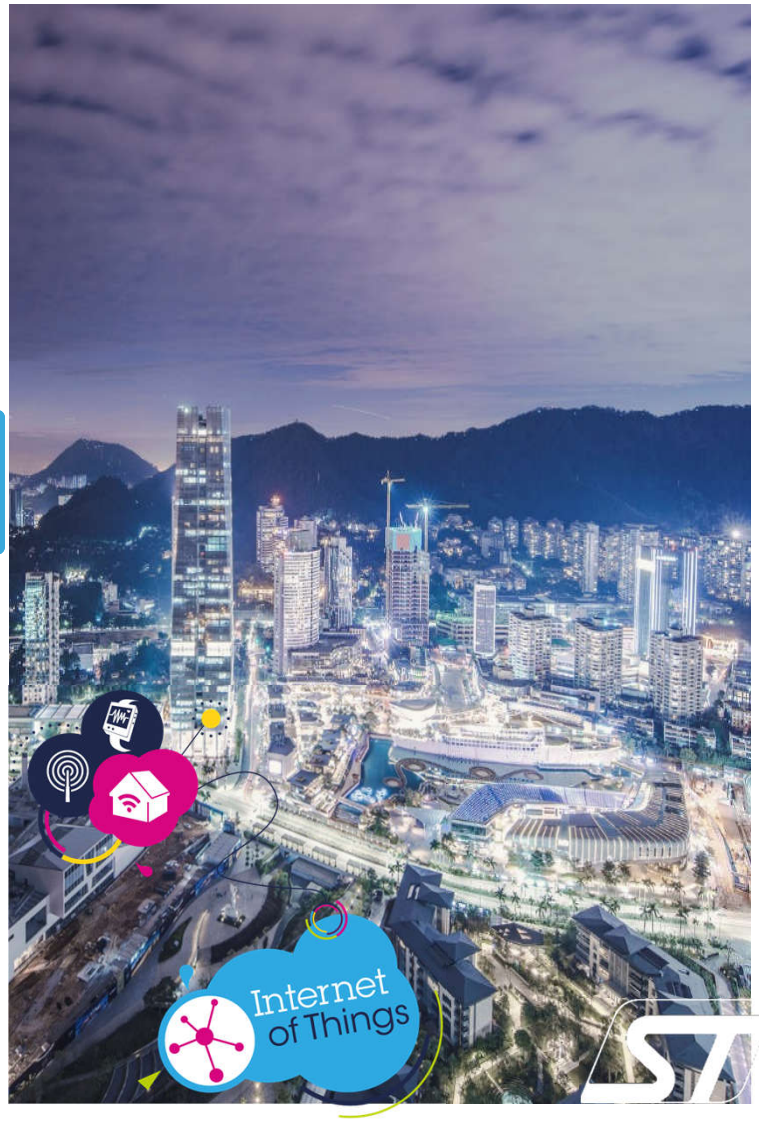
- Bluetooth Low Energy Mesh Introduction
- BLE Mesh over Android Framework
- App Walkthrough
- API Walkthrough and Hands on Session



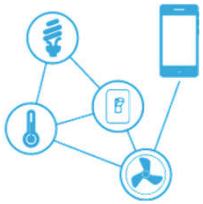
life.augmented

BLE Mesh 安卓 App

- Bluetooth Low Energy Mesh 介绍
- 基于安卓框架的BLE Mesh
- App 流程
- API 使用流程以及实战任务



life.augmented



BlueNRG-MESH SDK

for Embedded, Android and iOS

BlueNRG-Mesh SDK

Applications	LED Light Demo	
Middleware	ST BlueNRG-Mesh Library	ST BlueNRG Core Stack
Drivers	Hardware Abstraction Layer API	Boards Support Packages
HW Components	STM32 + BlueNRG-MS	BlueNRG-1 or BlueNRG-2

Android SDK

ST BlueNRG-Mesh Android App	
Android BlueNRG-Mesh Core Stack	
Android Application Framework	
Android BT4.x Stack	
Linux Kernel	
Bluetooth Chip	

iOS SDK

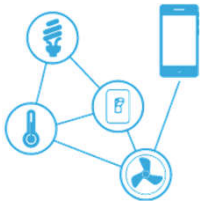
ST BlueNRG-Mesh iOS App	
iOS BlueNRG-Mesh Core Stack	
iOS Application Framework	
iOS BT4.X Stack	
XNU Kernel	
Bluetooth Chip	

Available on SoC and network processor

life.augmented

BlueNRG-Mesh SDK

嵌入式， Android 和 iOS



BlueNRG-Mesh SDK

应用	LED Light Demo	
中间层	ST BlueNRG-Mesh Library	ST BlueNRG Core Stack
驱动	Hardware Abstraction Layer API	Boards Support Packages
硬件器件	STM32 + BlueNRG-MS	BlueNRG-1 or BlueNRG-2

Android SDK

ST BlueNRG-Mesh Android App	
Android BlueNRG-Mesh Core Stack	
Android Application Framework	
Android BT4.x Stack	
Linux Kernel	
Bluetooth Chip	

iOS SDK

ST BlueNRG-Mesh iOS App	
iOS BlueNRG-Mesh Core Stack	
iOS Application Framework	
iOS BT4.X Stack	
XNU Kernel	
Bluetooth Chip	

SoC和网络处理器皆可提供

life.augmented

BLE Mesh Over Android Framework

- Android SDK for BLE Mesh provides easy to use APIs for creating and controlling the Bluetooth Mesh Network
- The APIs are the abstraction on top of the Android Bluetooth APIs , therefore developers need not know the Android Bluetooth APIs.
- Sample Application is provided which developers can modify or write their own App from scratch.
- Real Android Smartphones are required to develop the applications. Emulators cannot be used.

https://developer.android.com/studio/

基于安卓架构的BLE Mesh

- BLE Mesh的安卓SDK提供了简单易用的API用于创建和控制蓝牙Mesh网络
- APIs是安卓蓝牙APIs的顶层的抽象，因此开发者不需要了解安卓蓝牙APIs。
- 开发者在提供的参考应用程序上修改或开发他们自己的App。
- 开关应用需要真正的安卓手机。仿真器不能使用。

https://developer.android.com/studio/

Key Concepts(1/3)

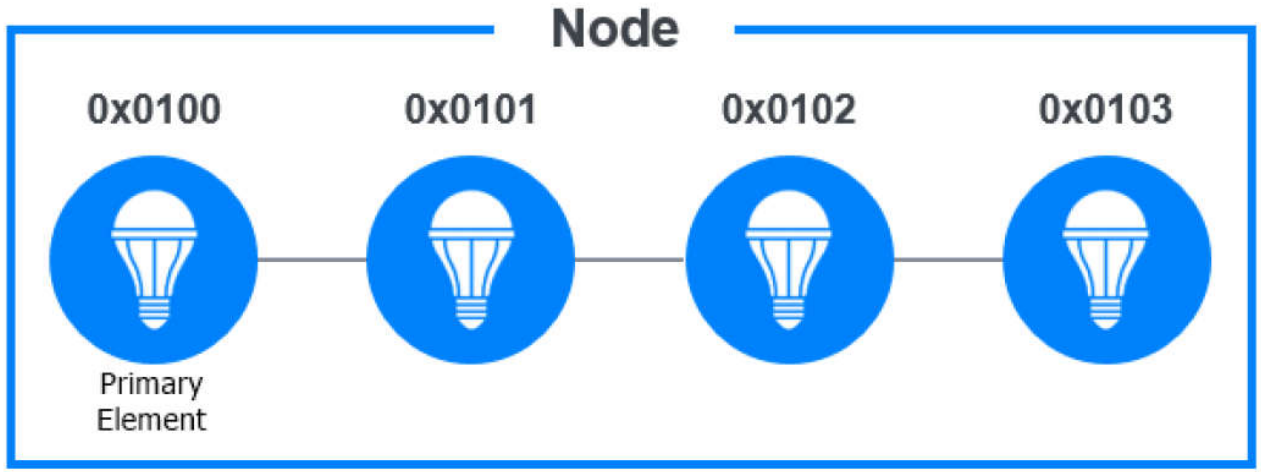
- Provisioning and Configuration :
  - Provisioning : It is a security procedure which is carried out manually by a network administrator (Mesh owner) using smart phone or a tablet .It equips the Unprovisioned devices with a series of security keys and other essential network credentials(unicast address, IVI index, etc) . All the Unprovisioned entities are called devices and all the provisioned entities are called Nodes in a mesh network. Once provisioning of a device is done it is called a Node.
  - Configuration : Once provisioned, a node can be configured in many ways . It is a process of providing a desired subscription or a publication address to it which is most appropriate to the type of product. For example a Mesh switch can be configured to control a group of lights subscribed to it. It is generally performed with the same smartphone with which the provisioning is done. Smartphone can provide a default configuration to a node at the time of provisioning.

关键概念 (1/3)

- 启动配置和设置:
  - 启动配置：这是一个安全的步骤，由网络管理员（mesh主人）使用智能手机或平板电脑手动执行。它给未被配置的设备提供了一系列安全密钥和其他基础的网络证书（单播地址，初始向量索引，等等）。在网络中所有未被配置的设备被称作实体，所有配置过的设备称作节点。一旦一个设备被配置完成，它就被称作节点。
  - 设置：一旦启动配置完成，一个节点可以使用多种方式进行设置。这是一个提供给最合适的产品类型所需要的订阅或发布地址的过程。例如，一个mesh开关可以被设置去控制一组订阅它的灯。这通常是跟启动配置完成的相同的智能手机一起操作的。智能手机可以在启动配置的同时给节点提供默认设置。

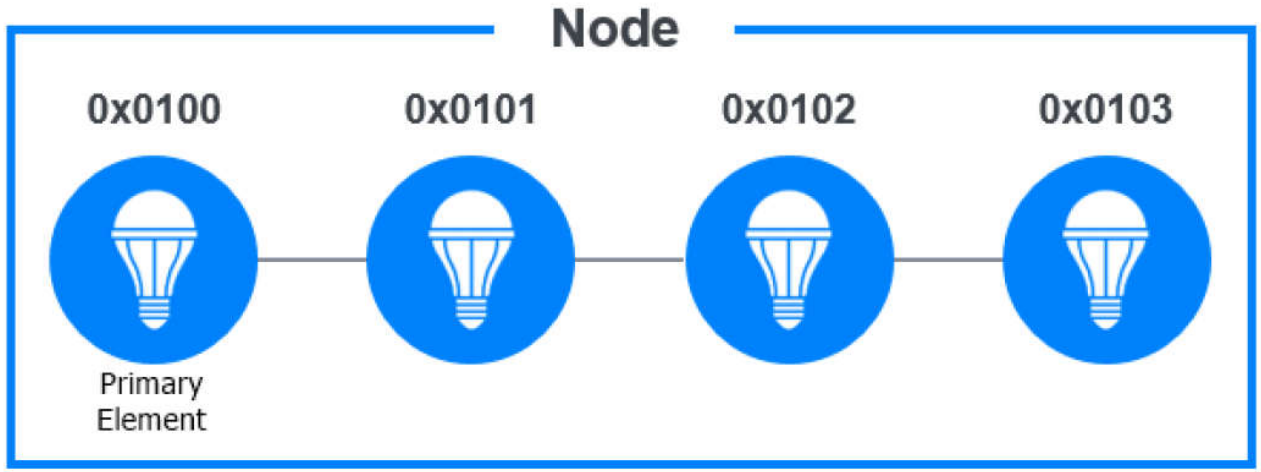
Key Concepts(2/3)

- Nodes and Elements
  - Nodes : Nodes are devices which have been provisioned and are members of a specific mesh network identified by one of the keys provided to it by the provisioning process
  - Elements : Some nodes types are more complicated than others and may consist of multiple parts, each individually addressable but typically housed inside the same physical unit .Each parts is an addressable units . One of the elements is designated the primary element which contains the configuration data which is applicable to the whole node .



关键概念(2/3)

- 节点和要素
  - 节点：节点是一些被配置过设备，并且是一个特定mesh网络的成员，这个网络是由启动配置过程中提供的密钥之一进行认定的。
  - 要素：一些节点类型是相对更复杂的并且包含多种部分，每部分都可以独立寻址，但是被典型地封装在相同的物理单元。每一部分都是可以独立寻址的单元。要素之一被指定为主要要素，包含了整个节点可应用的配置数据。



Key Concepts(3/3)

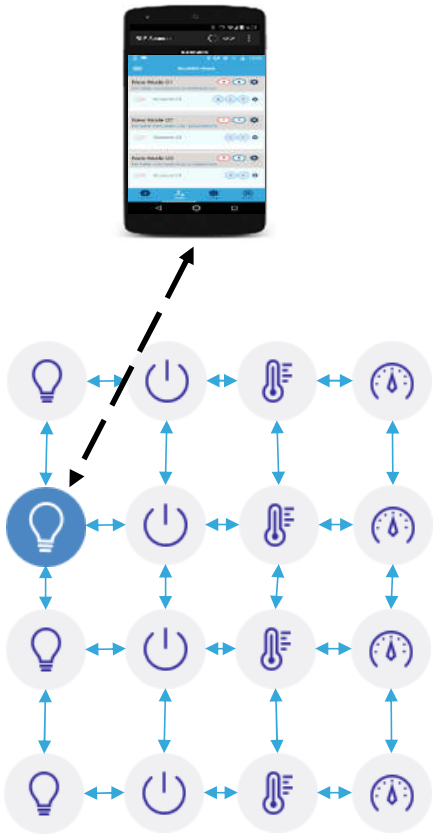
Unassigned	0000000000000000	No address assigned (typically used when not publishing or subscribing)
Unicast	0xxxxxxxxxxxxxx	Address of a mesh node in a network (max unicast Address = 32,767/network) 16-bit unicast addresses
Group	11xxxxxxxxxxxxx	Identify collections or sets of devices(typically used for publish and subscribe) (max group Address = 16,384/network) 16-bit addresses
Virtual	10xxxxxxxxxxxxx	Identify collections or sets of devices : 16-bit value which maps to 128–bit UUID* (14 bits contains Hash of 128 bit UUID which the virtual address represent)
Broadcast	1111111111111111	Every node in a network

UUID* : Universal Unique Identifier

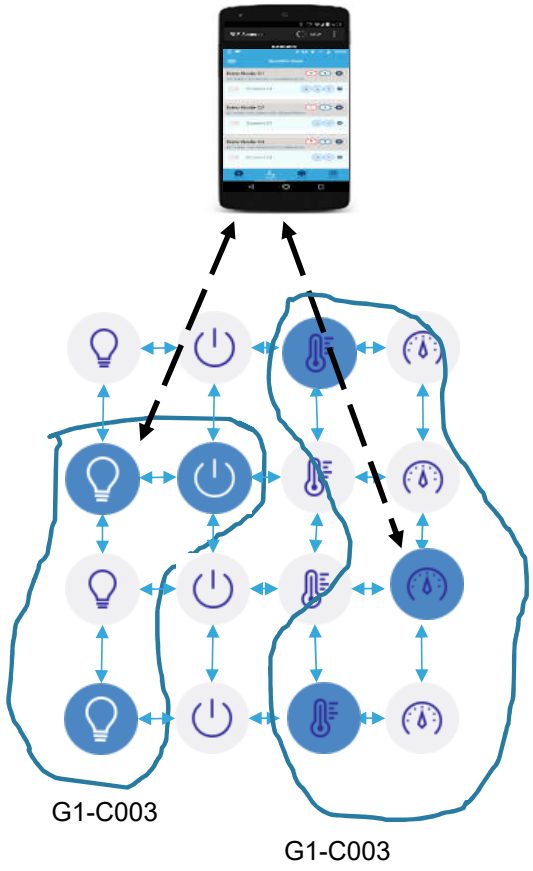
Mesh 地址

未分配	0000000000000000	没有分配地址(无发布或订阅时被特定使用)
单一的	0xxxxxxxxxxxxxxx	网络里的一个mesh节点的地址
虚拟的	10xxxxxxxxxxxxxx	一个或一组设备的虚拟地址
分组的	11xxxxxxxxxxxxxx	一组mesh节点的地址 (发布或订阅时被特定使用)
全体的	1111111111111111	网络中的每一个节点

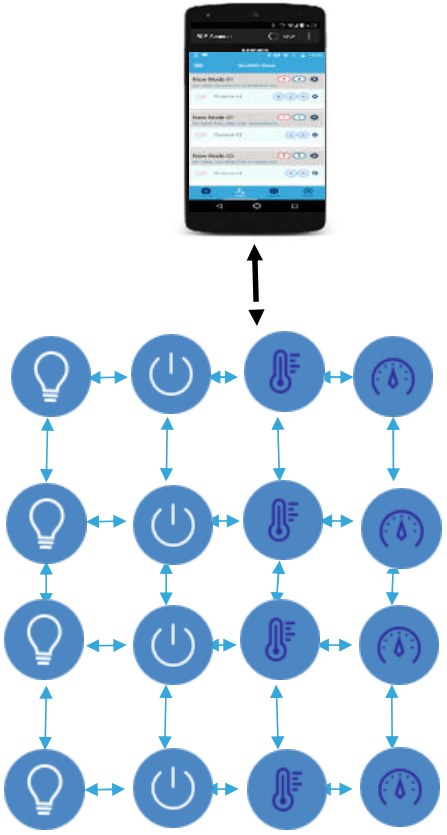
Addresses in a Mesh Network



Unicast Addressing

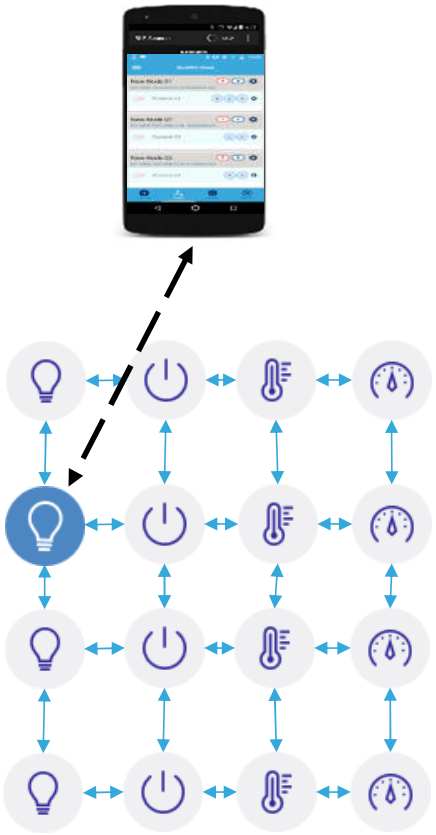


Group Addressing

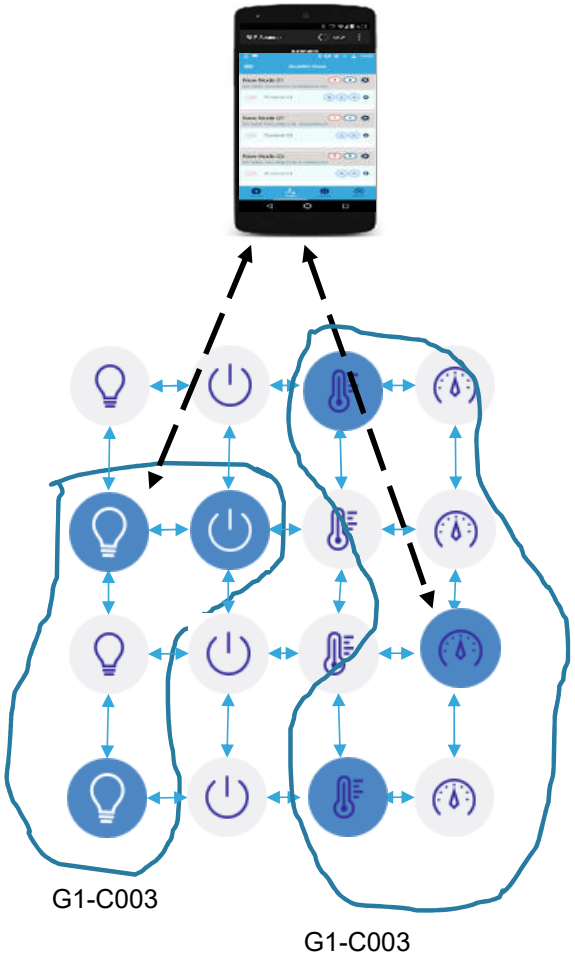


Broadcast Addressing (0xFFFF)

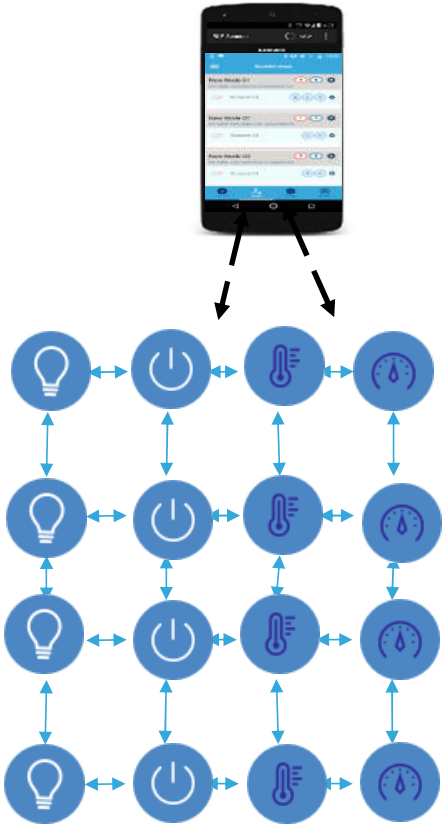
Mesh 网络中的地址



单播地址



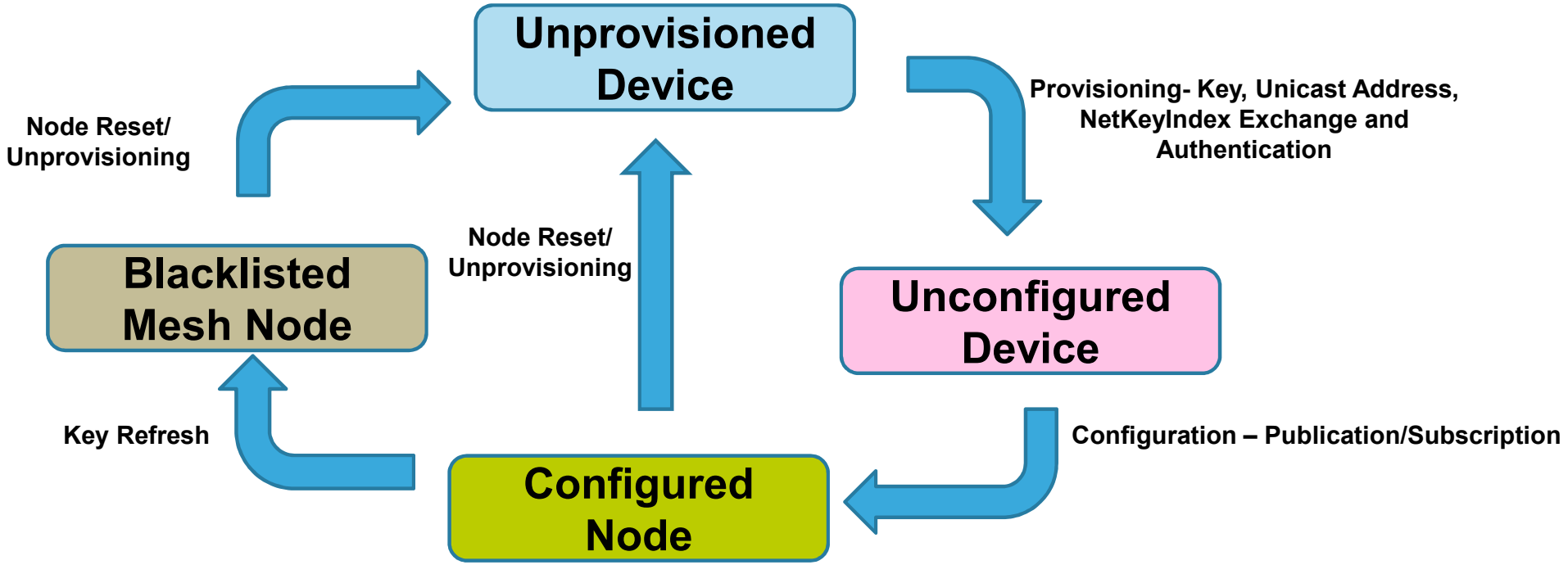
组播地址



全体广播地址
(0xFFFF)

life.augmented

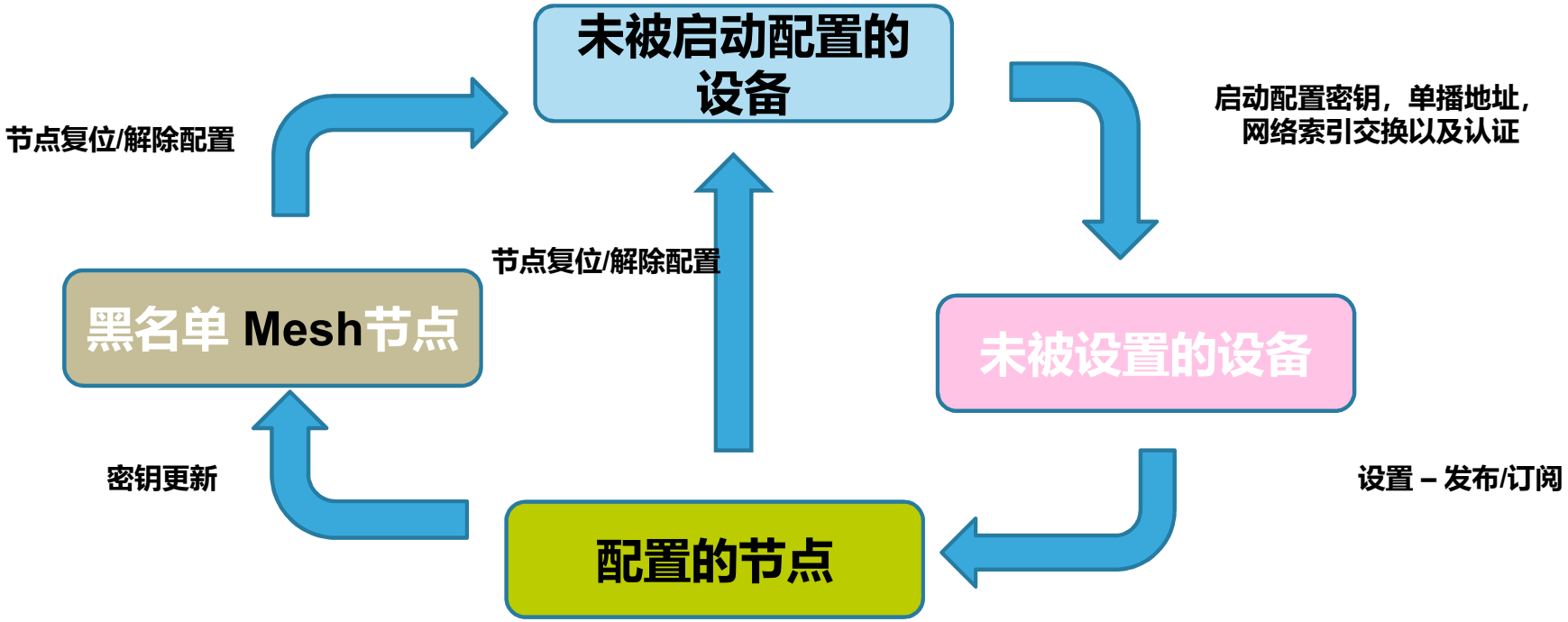
Provisioning/Configuration State Machine



Bluetooth Mesh Device State Lifecycle

life.augmented

启动配置/配置状态机

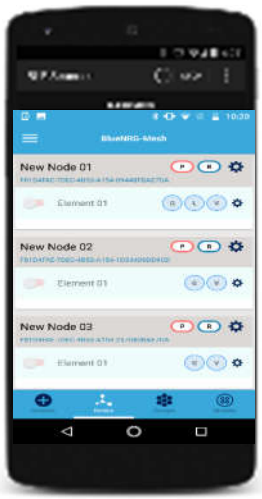


蓝牙 Mesh 设备状态生命周期

Provisioning Process(0-70%)

Provisioner

New Device



Scan for Unprovisioned Device

Advertises Unprovisioned Device Beacons and un-provisioning service

10% = Invite for Provisioning

20% = Provisioning Capabilities(OOB*, Algorithm Types)

30% = Provisioning start PDU

40% = Provisioning Public Key(**ECDH compliant)

Calculate ECDH

Calculate ECDH

50% Provisioning Public Key(ECDH compliant)

Session Key calculation

Session Key calculation

60% Provisioning Data(unicast address, Network Key,NetKeyIndex)

70% Provisioning Completed(0x08 **PDU Type)

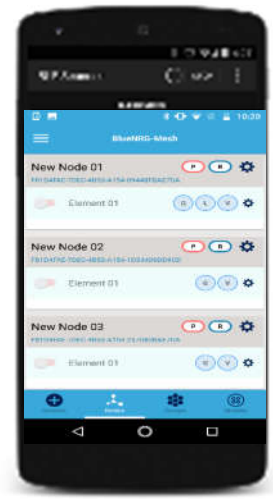
Disconnection

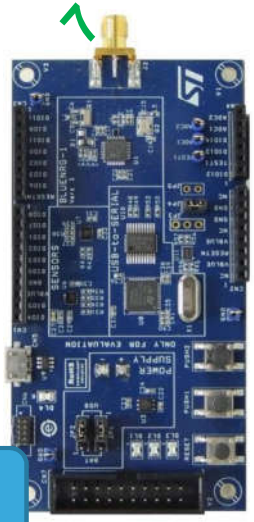
Disconnection

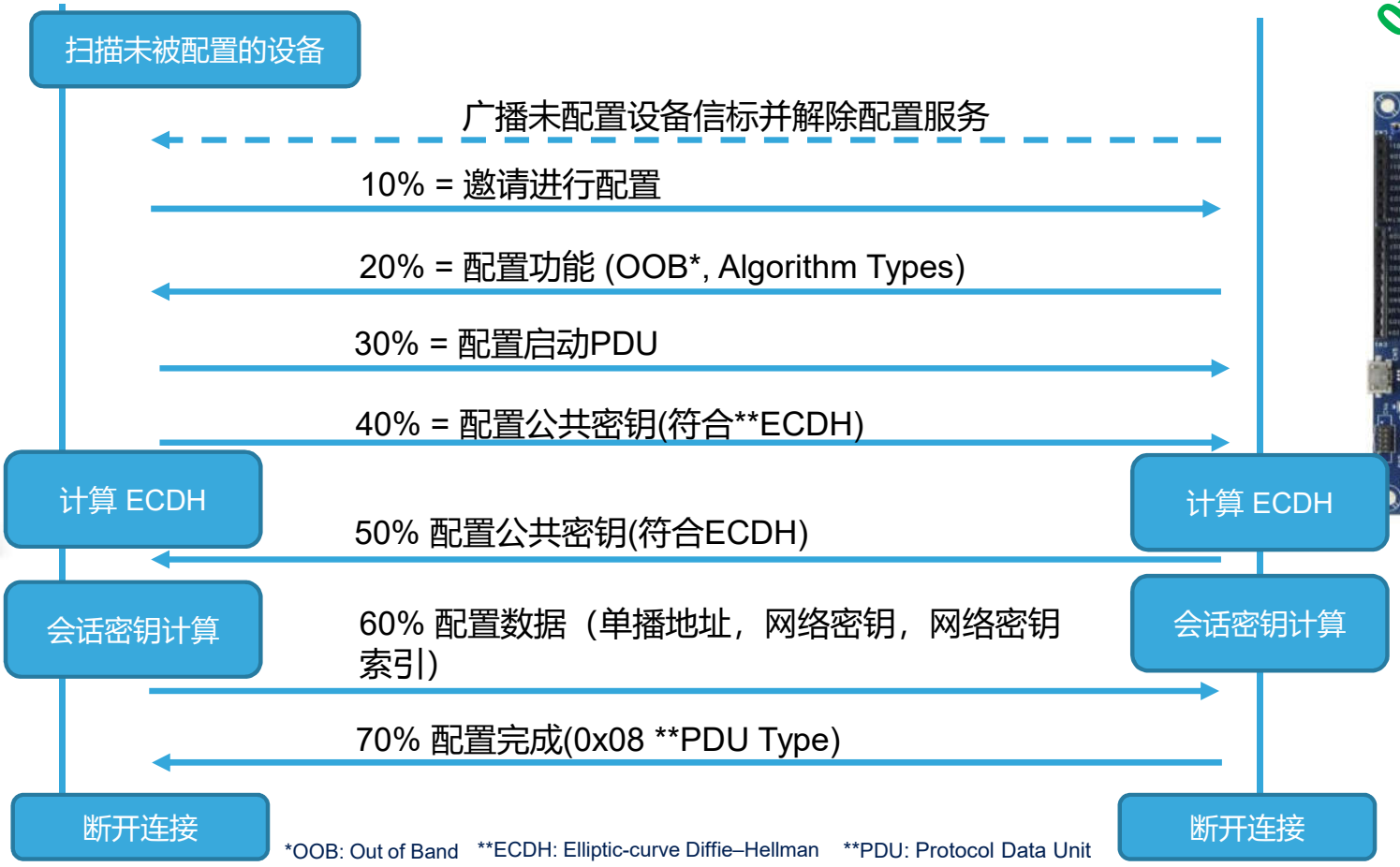
*OOB: Out of Band **ECDH: Elliptic-curve Diffie–Hellman **PDU: Protocol Data Unit

life.augmented

启动配置过程（0-70%）



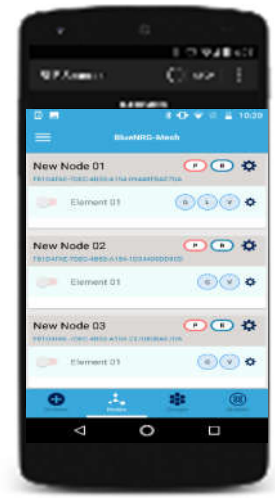




*OOB: Out of Band **ECDH: Elliptic-curve Diffie–Hellman **PDU: Protocol Data Unit

life.augmented

Configuration Process (70-100%)



Provisioner

Provisioned Device

0x1828

Advertises Node Identity or Network Id in the Proxy Service

Scan for Proxy Service

80% = Share App Key(Random AppKey)

Status = Success

90% = What is Device type ?(Optional)

Device Type answer(Optional)

100% Default Subscribe and Publications to Elements

Subscription and Publication Status

GATT bearer

ADV bearer

Configuration completed

Configuration completed

Advertises Secure Network Beacons and Proxy Service Data

Node (Provisioned+Configured)

life.augmented

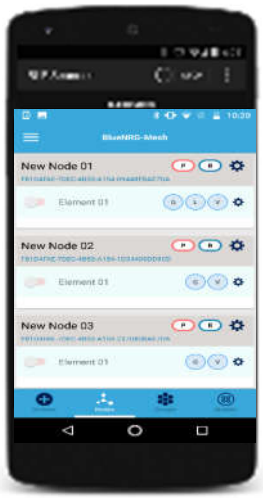
设置过程 (70-100%)

配置器

在代理服务中广播节点身份或网络身份

启动配置过的设备

扫描代理服务



80% = 分享App密钥（随机App密钥）

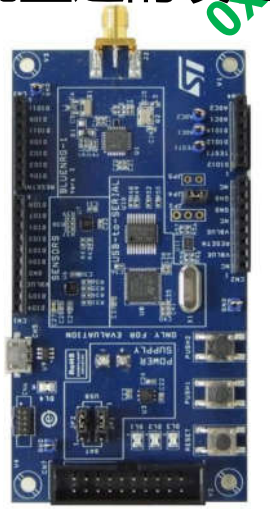
状态 = 成功

90% = 设备类型是什么?(可操作的)

设备类型回复(可操作的)

100% 对元素的默认订阅和广播

订阅和广播状态



GATT bearer

ADV bearer

设置完成

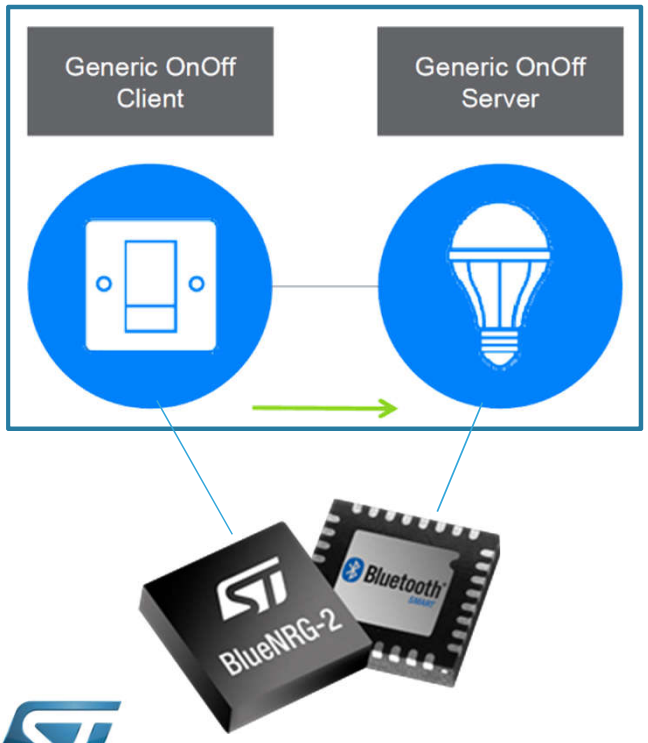
广播安全网络信标和代理服务数据

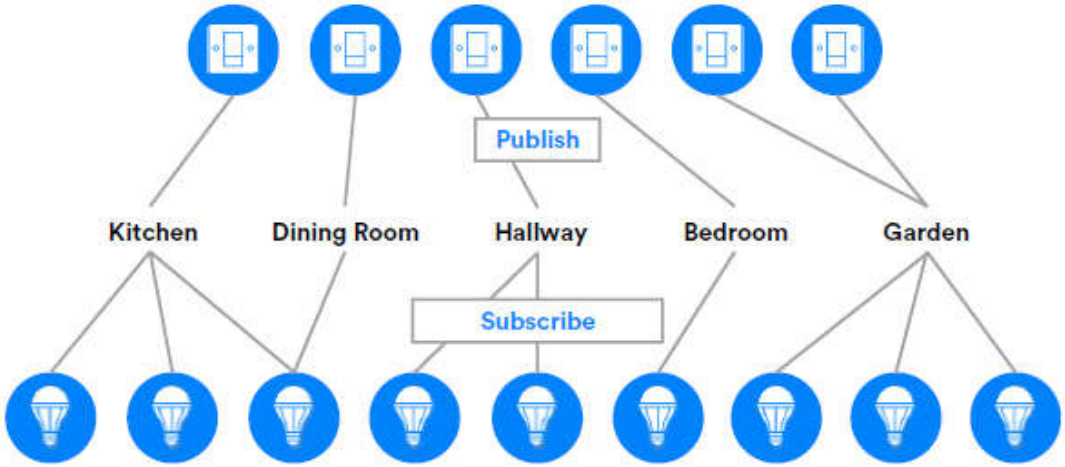
设置完成

节点
(配置完成+设置完成)

life.augmented

Mesh Messaging Model
Publish and Subscribe

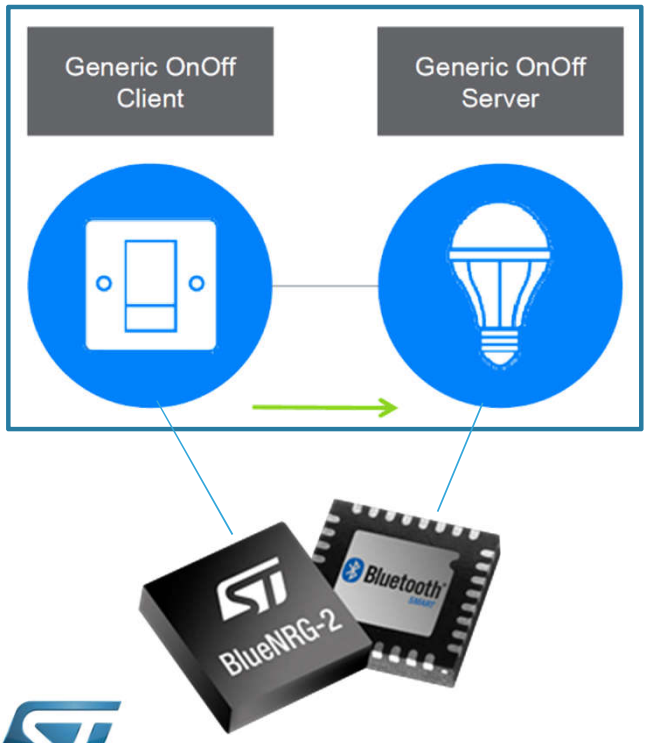


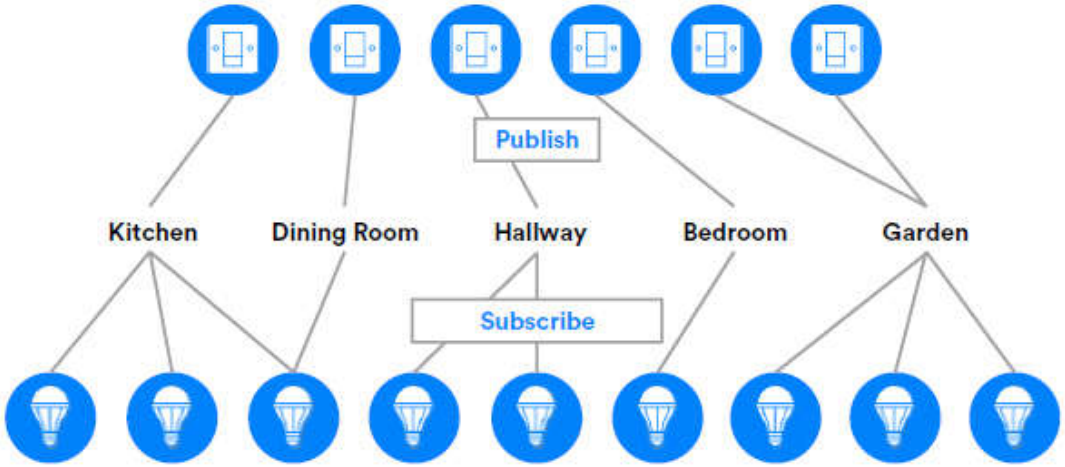


Example: **client device** (switch) can post messages and **server device** (light bulb) can be notified about new command arrival.

life.augmented

Mesh 消息模型
发布和订阅





示例: **客户端设备** (开关) 可以投送消息并且服务器设备（照明灯泡）会被通知新命令的到达

life.augmented

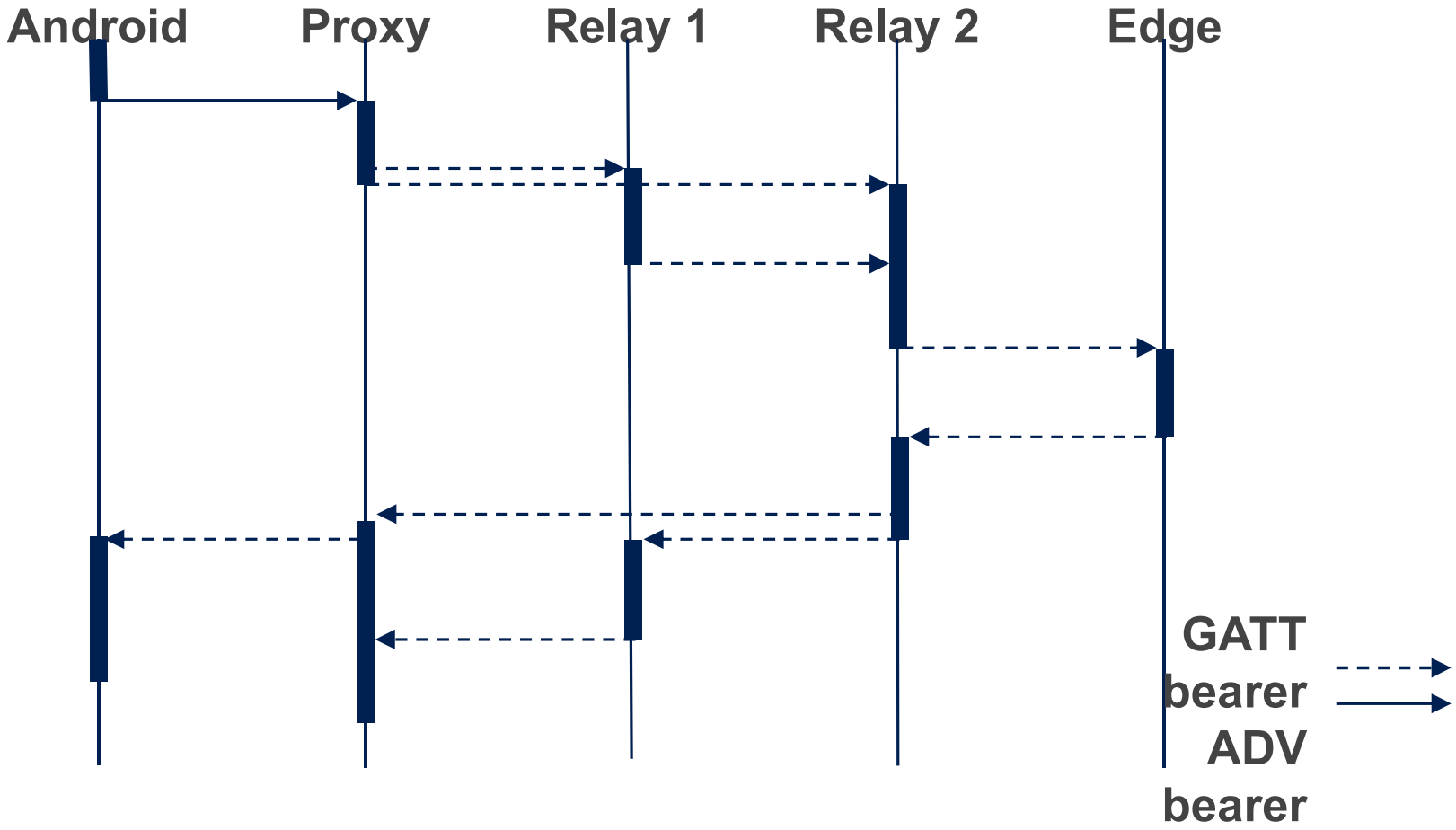
Publish & Subscribe

- Publish
  - Model state is published to a single DST
  - can be a unicast / virtual / group address
- State is published
  - periodically
  - when reliable message is received
  - when state has changed on it's own
  - when state was changing
- Subscribe
  - Nodes subscribe to DST addresses
  - a node can subscribe to multiple addresses
- Plug and Play
  - matching publish and subscribe addresses is the way that mesh network is configured
  - allows replacement of devices without having to reconfigure all other nodes

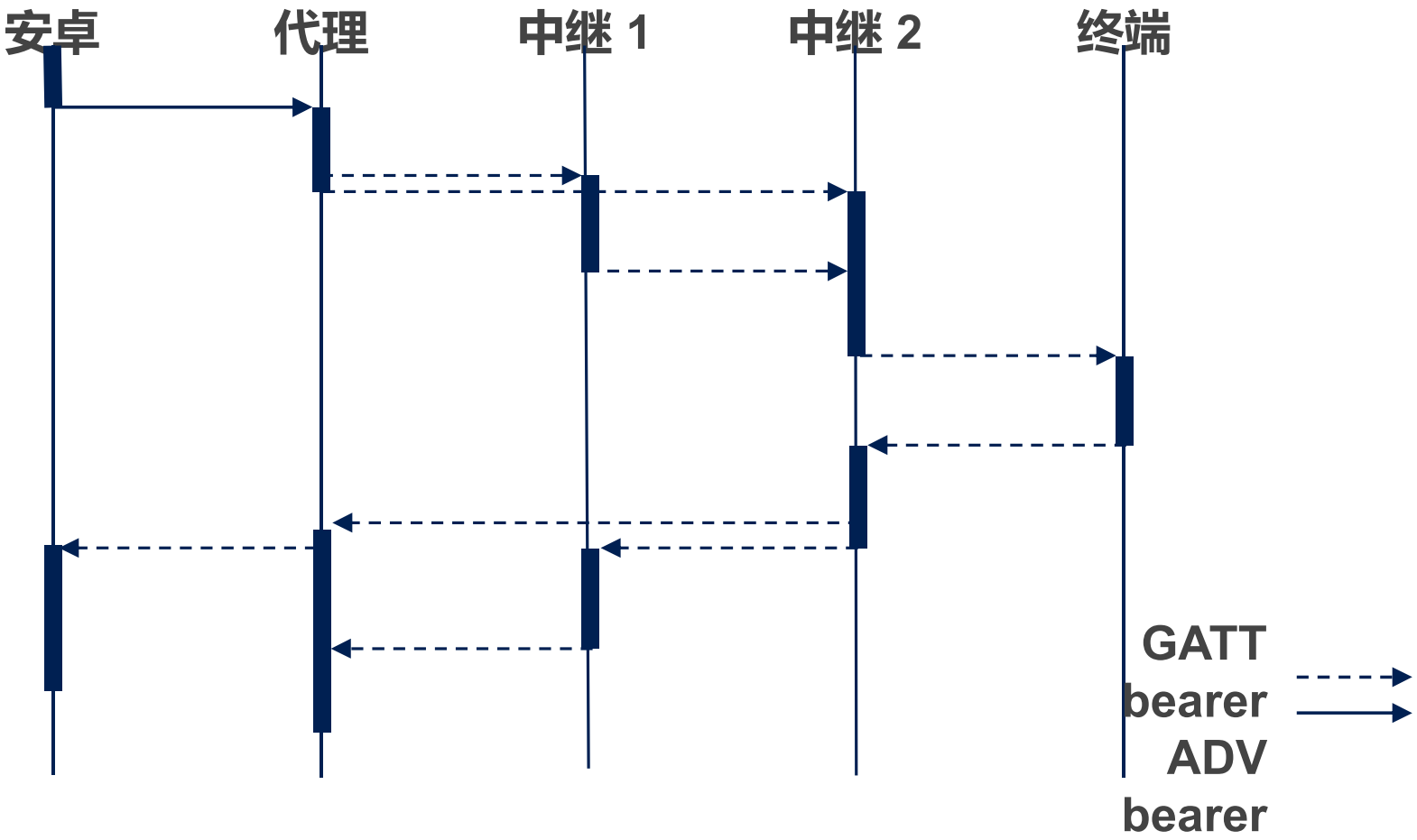
发布 & 订阅

- 发布
  - 模型状态被发布到一个单一的目标地址
  - 可以被单播、虚拟广播、组播
- 发布的状态
  - 周期性地
  - 当收到可靠消息时
  - 当状态自己发生变化后
  - 当状态正在改变时
- 订阅
  - 节点订阅到目标地址
  - 一个节点能够订阅到多个地址
- 即插即用
  - mesh网络是通过匹配发布和订阅地址的途径被配置的
  - 允许不经重新配置其他节点即可替换设备

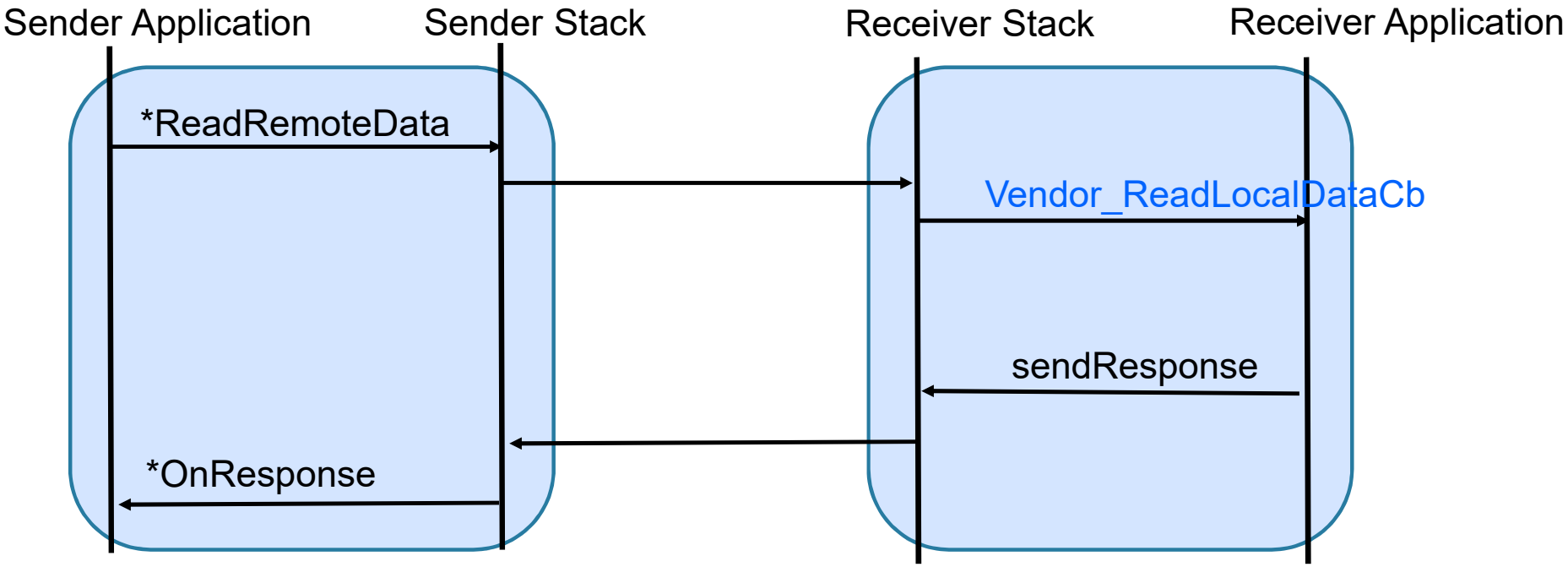
Android-Based MoBLE Stack Communication in Mesh Network



Mesh网络中基于安卓的手机协议栈通讯



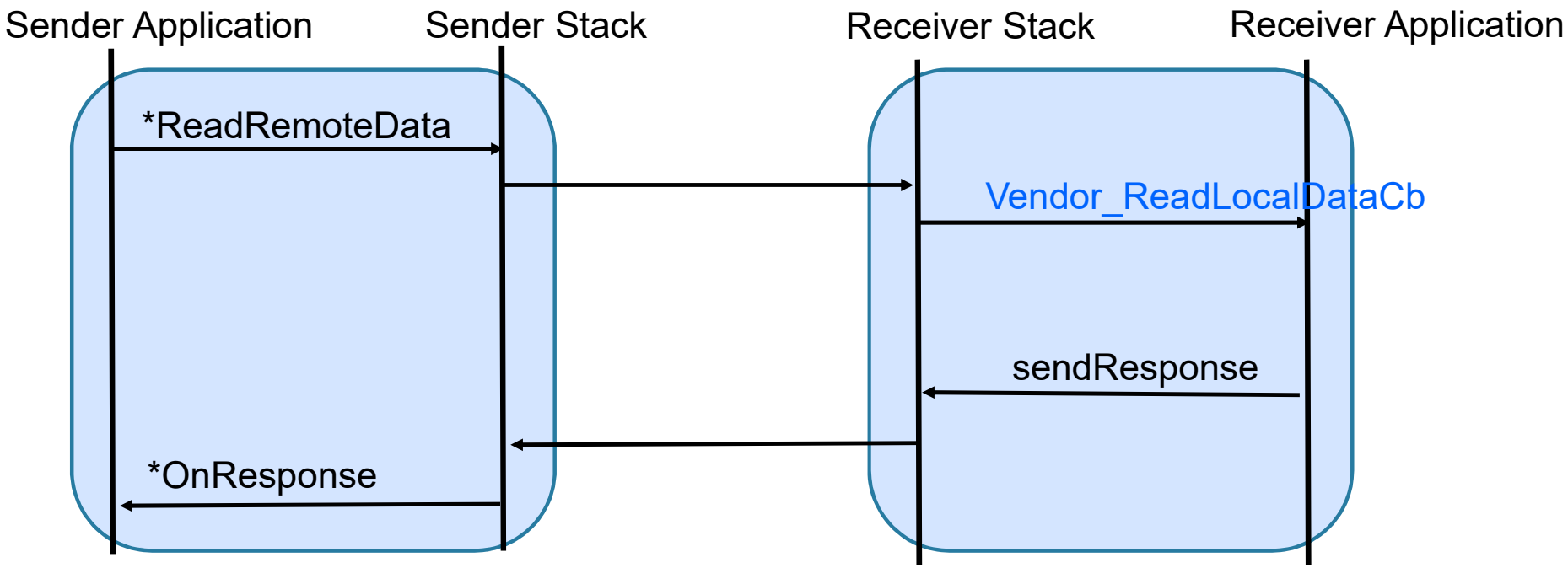
readRemote() : Read Remote Data-Reliable(sent to Proxy Node)



*Use case : Provisioner wants to read the Sensor Data
Bydefault : reliable
Sender = Android Phone/ BLE Device
Receiver = BLE Device

life.augmented

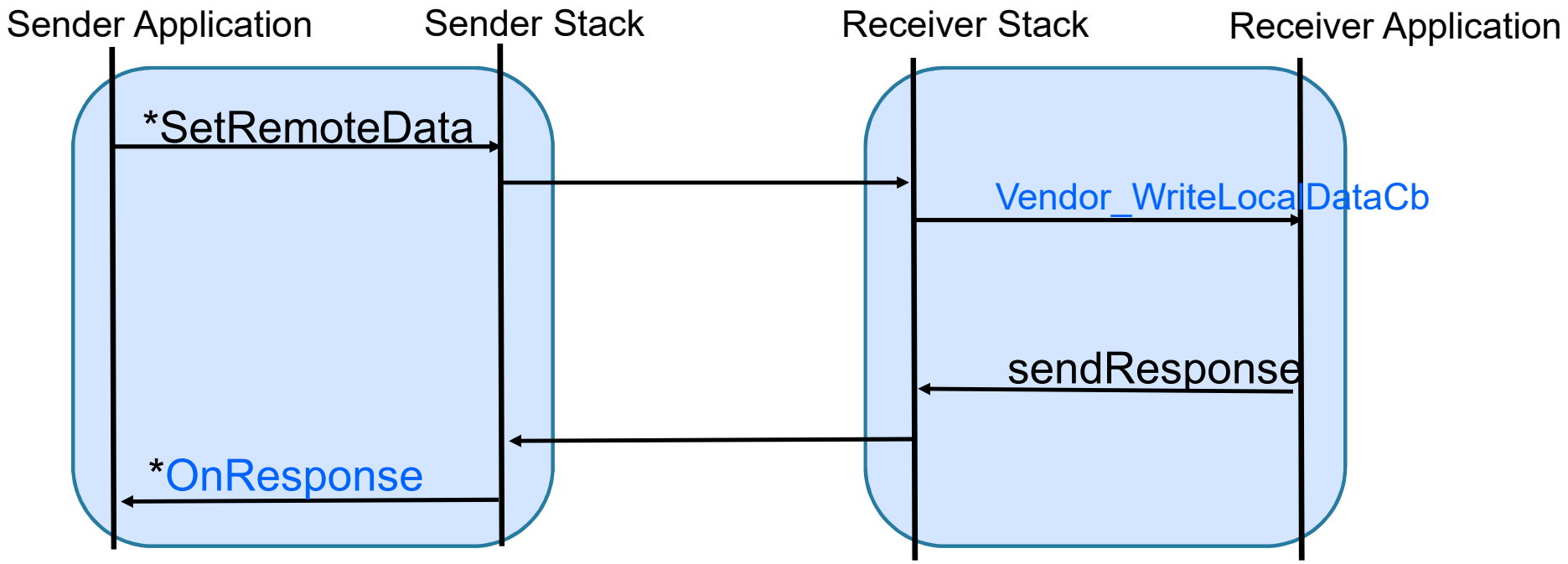
readRemote()：读远程可靠数据（发送到代理节点）



*使用案例：配置器想读取传感器数据
默认：可靠的
发送设备 = 安卓手机/BLE设备
接收设备 = BLE 设备

life.augmented

writeRemote : WriteRemote Data(Reliable) (sent to Proxy Node)



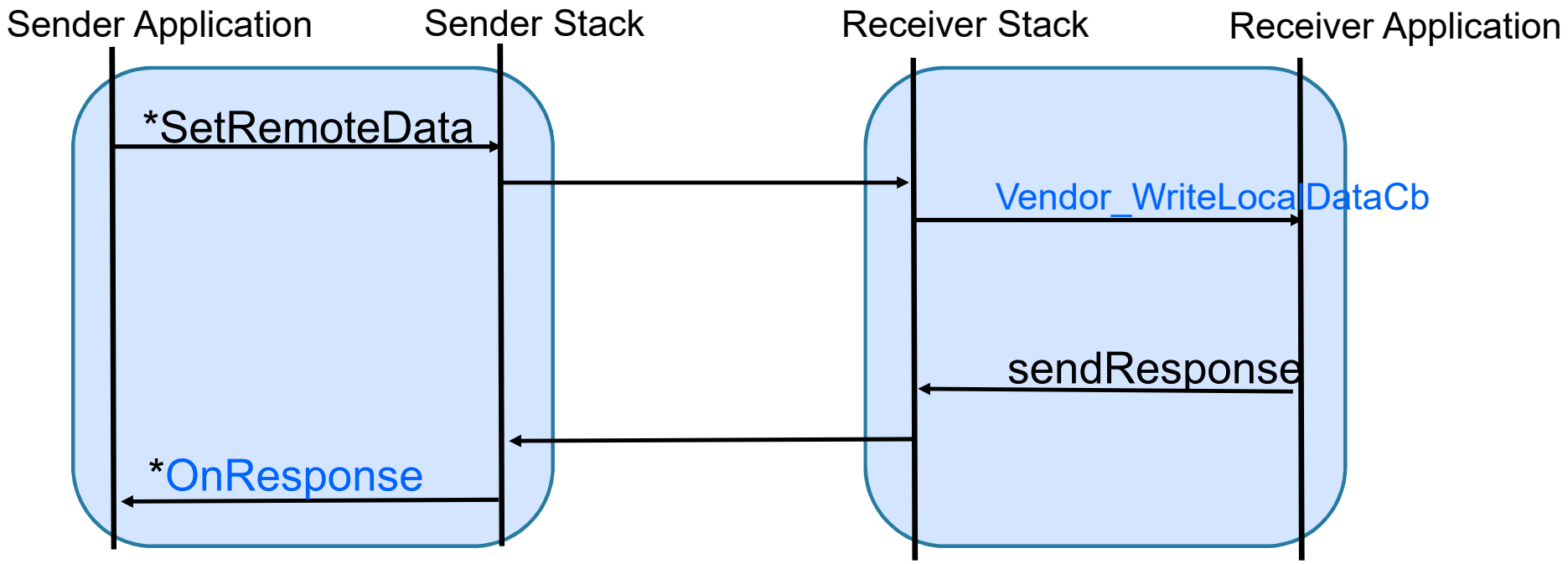
*Use case : Provisioner wants to send data the Sensor Data and get the response

Sender = Android Phone/ BLE Device

Receiver = BLE Device

life.augmented

writeRemote：写远程可靠数据（发送到代理节点）



*使用案例: 配置器想发送数据到传感器并获取响应
发送设备 = 安卓手机 / BLE 设备
接收设备 = BLE 设备

life.augmented

Building and Running the Application

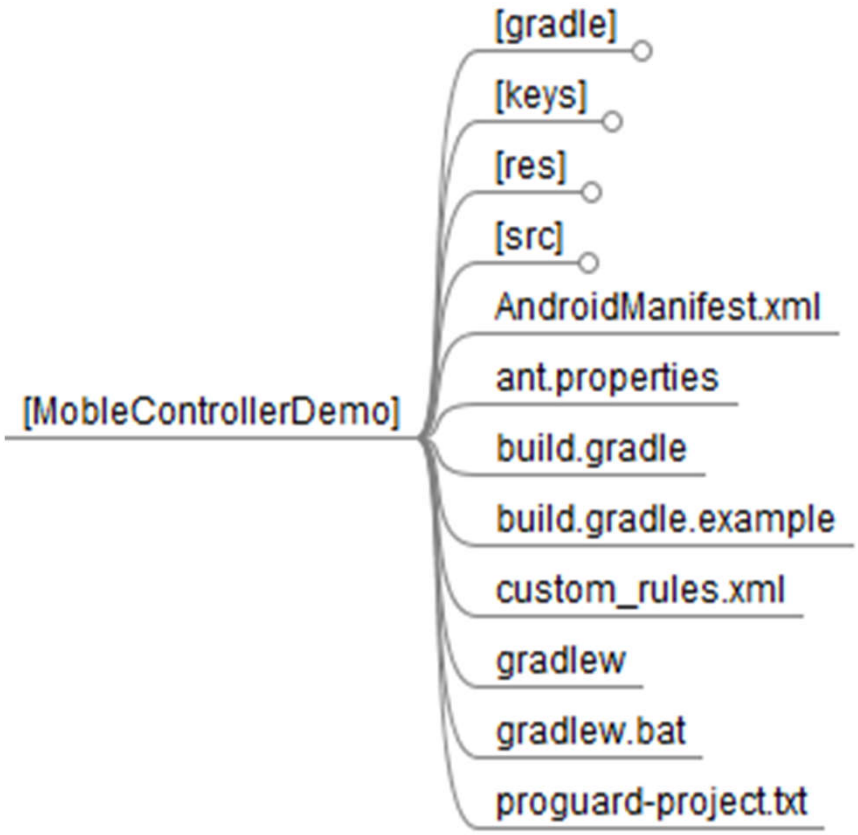
life.augmented

创建和运行应用程

life.augmented

MobleControllerDemo : Detailed Project Structure

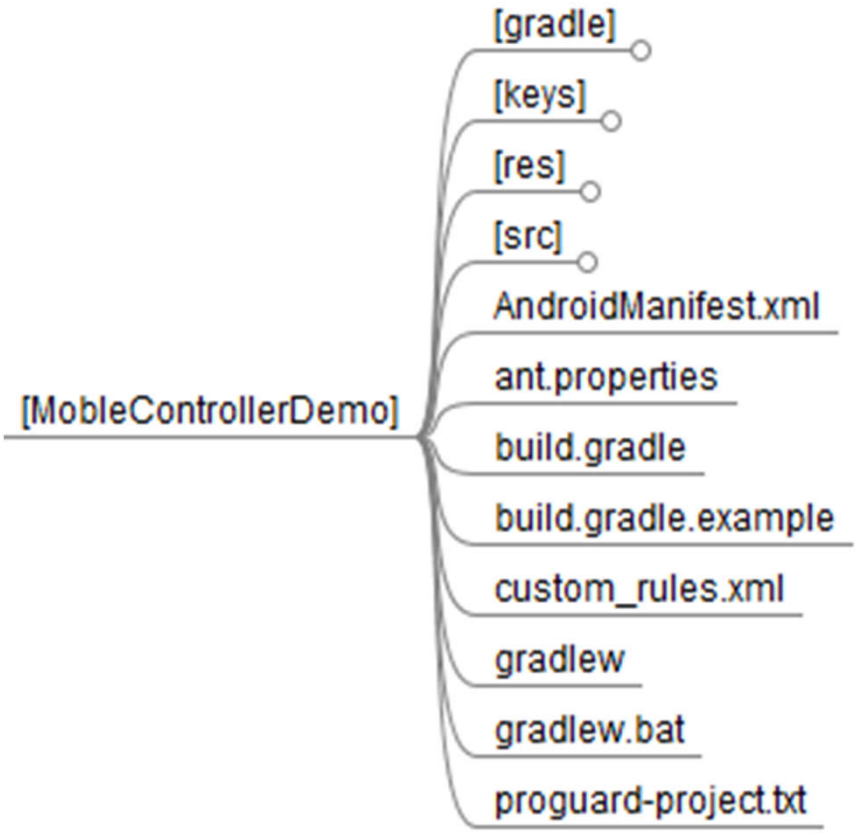
- Project folders
  - “gradle” (folder and related files) - Gradle automated build system
  - “keys” - Folder with release keys that is used to sign the release APK
  - “res” - graphical and textual resources (images, layouts, menus, strings, etc.) of demo application
  - “src” - folder with source code files written on Java language
- Temporary folders
  - “build” - folder with intermediate Java files and compiled demo application



life.augmented

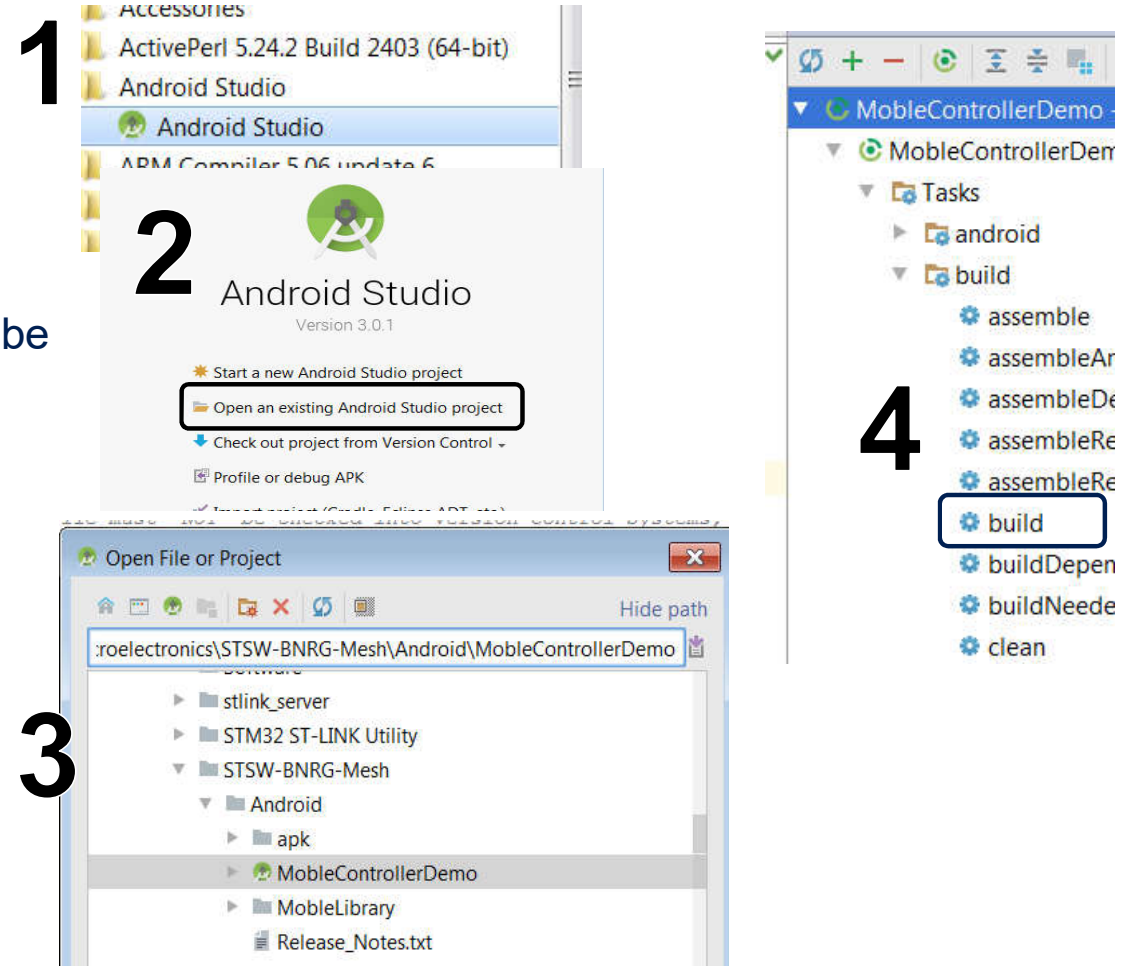
遥控示例：详细的工程结构

- 工程文件夹
  - “gradle”（文件夹和相关文件）- gradle自动创建系统
  - “keys” – 带有释放密钥的文件夹，释放密钥用于签署释放APK
  - “res” – 示例应用程序的图形和文本资源（图片，走线，菜单，线条，等等）
  - “src” – 包含JAVA源码的文件夹
- 临时文件夹
  - “build” – 包含中间JAVA文件和编译过的示例应用的文件夹



Project Compilation : Android Studio

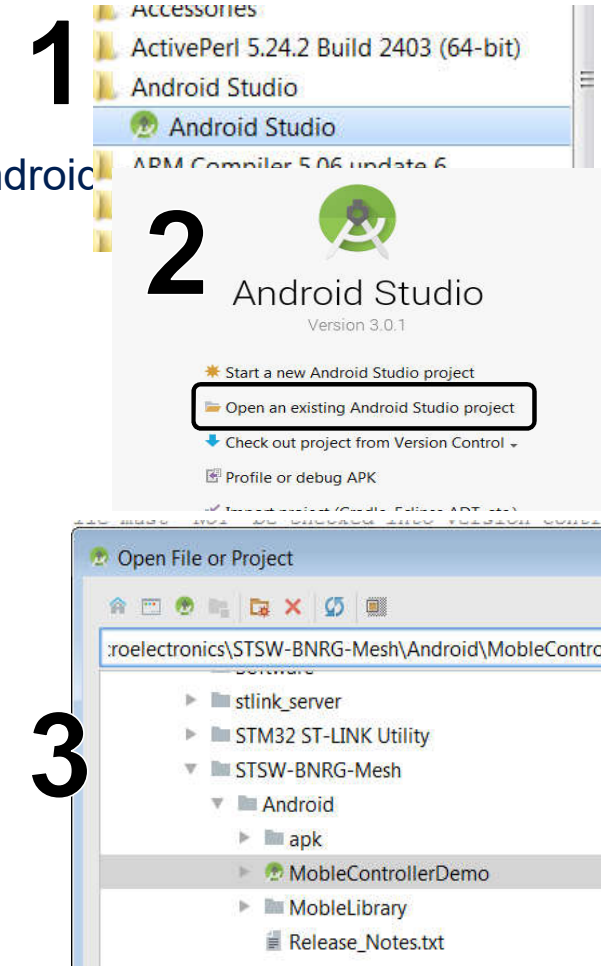
- Run Android Studio
- Click starting menu item “Open an existing Android Studio project”
- Select folder of project that need to be compiled:
  - MobleControllerDemo
- Open Gradle tasks view
- Build the project
  - Clicek Gradle tree item “Tasks->build”

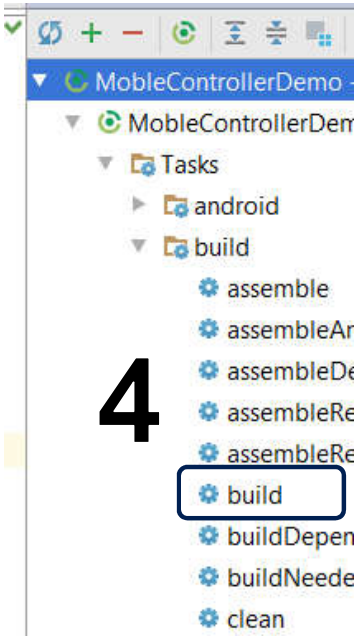


life.augmented

工程编辑：安卓工作室

- 运行 Android Studio
- 点击开始菜单 “Open an existing Androic Studio project”
- 选择需要编译的工程文件夹：
  - MobleControllerDemo
- 打开Gradle 任务查看
- 编译工程
  - Clicek Gradle tree item “Tasks->build”





life.augmented

Compilation Output

Android project	MobleControllerDemo
Output folder	src\android\MobleControllerDemo\build\outputs\apk
Filename	BlueNRG-Mesh-releaseUnsigned-unsigned.apk BlueNRG-Mesh-debug.apk BlueNRG-Mesh-release.apk

1. The file can be directly copied to the Android Filesystem and can be launched by clicking it.
2. The app can be uninstalled from the command line using command :
   - adb* uninstall com.st.bluenrgmesh
3. The app can also be installed using the adb command :
   - adb* install <apk-name>

life.augmented

*adb(android debug bridge) tool is found in the sdk\platform-tools

编辑输出

安卓工程	遥控示例
输出文件夹	src\android\MobleControllerDemo\build\outputs\apk
文件名	BlueNRG-Mesh-releaseUnsigned-unsigned.apk BlueNRG-Mesh-debug.apk BlueNRG-Mesh-release.apk

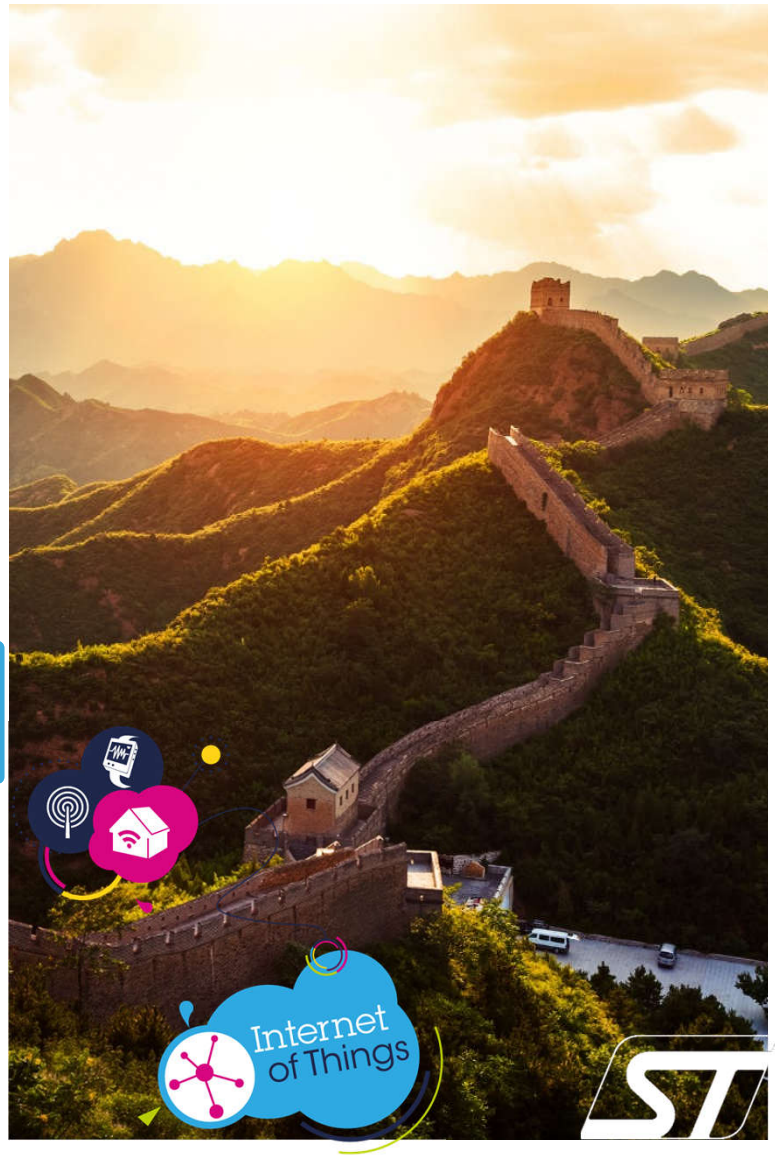
1. 文件可以被直接复制到安卓文件系统并且能够点击运行。
2. App能够通过使用命令行卸载：
   - adb* uninstall com.st.bluenrgmesh
3. App也能够使用adb命令进行安装：
   adb* install <apk-name>

life.augmented

*adb(android debug bridge) tool is found in the sdk\platform-tools

BLE Mesh Android App

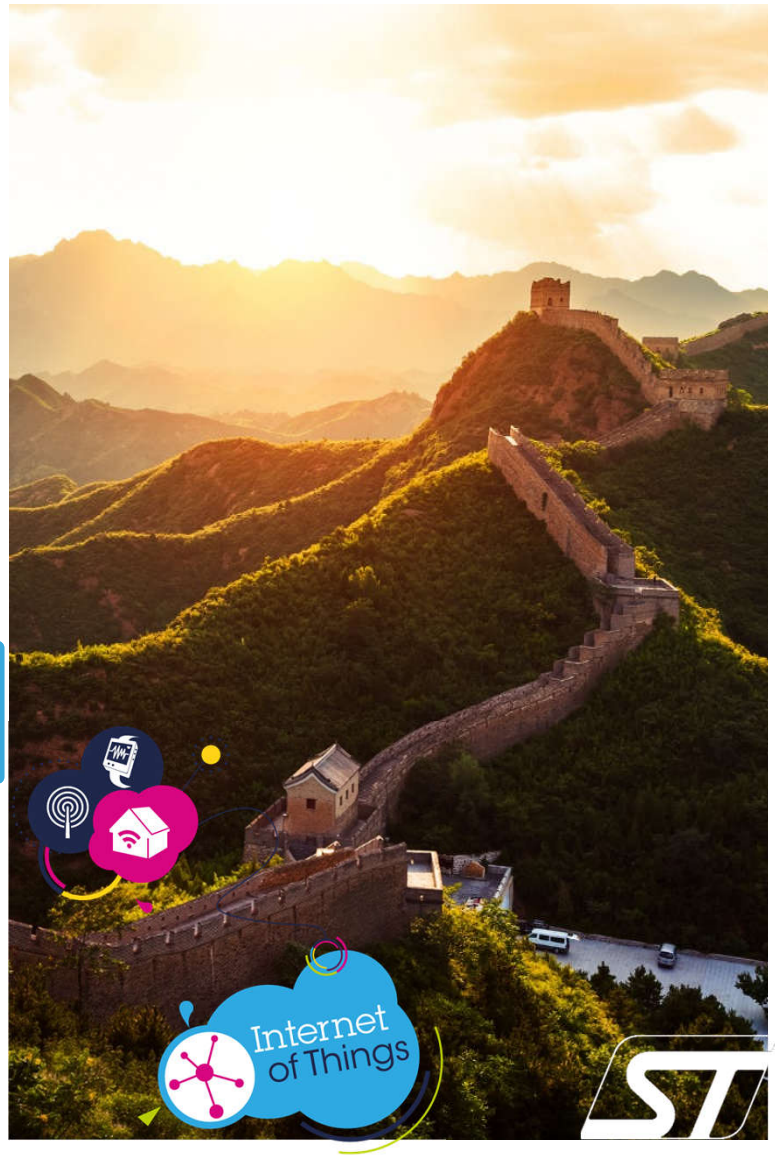
- Bluetooth Low Energy Mesh Introduction
- BLE Mesh over Android Framework
- App Walkthrough
- API Walkthrough and Hands on Session



life.augmented

BLE Mesh Android App

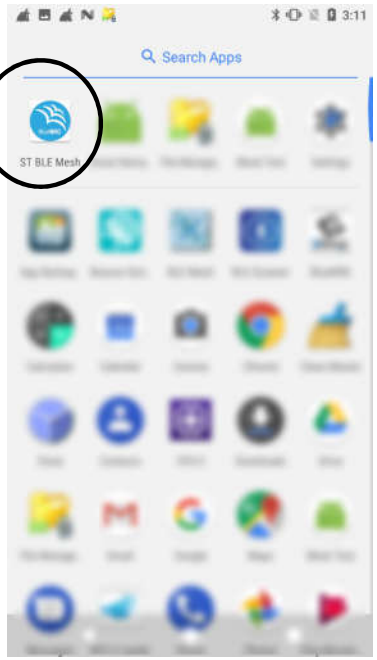
- Bluetooth Low Energy Mesh 介绍
- 基于安卓框架的BLE Mesh
- App 使用流程
- API 使用流程以及实战任务



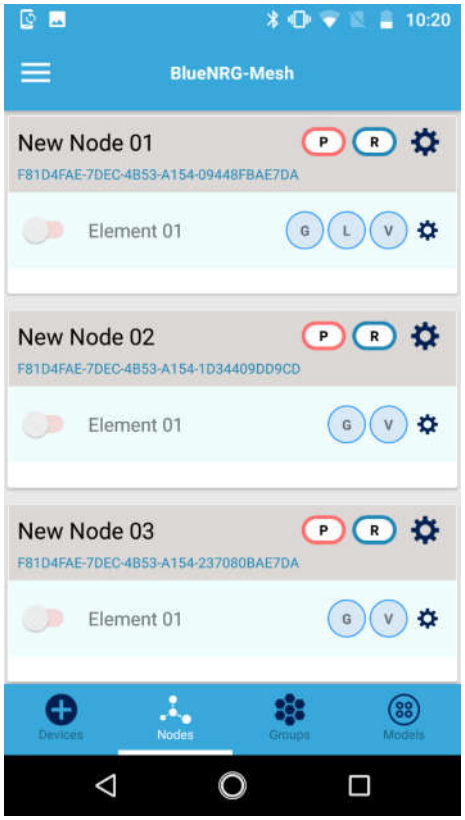
life.augmented

Android App Walk Through : App Launch and default View

- Launch The app



Application opens in the “Nodes” View showing the provisioned devices

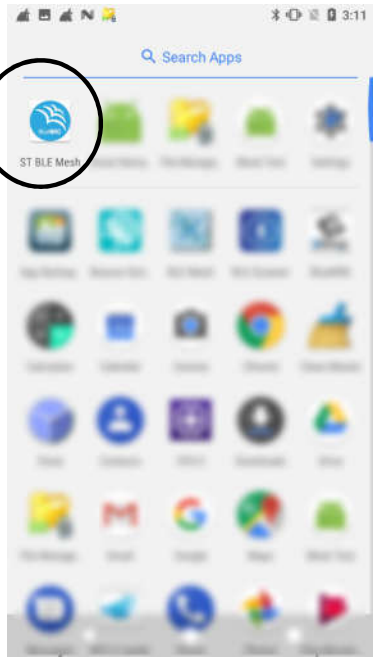


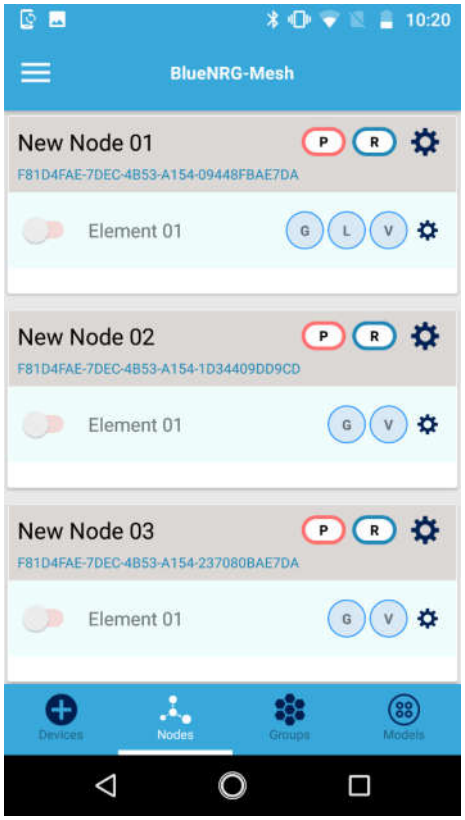
life.augmented

安卓 App 流程: 启动App以及默认视图

- Launch The app

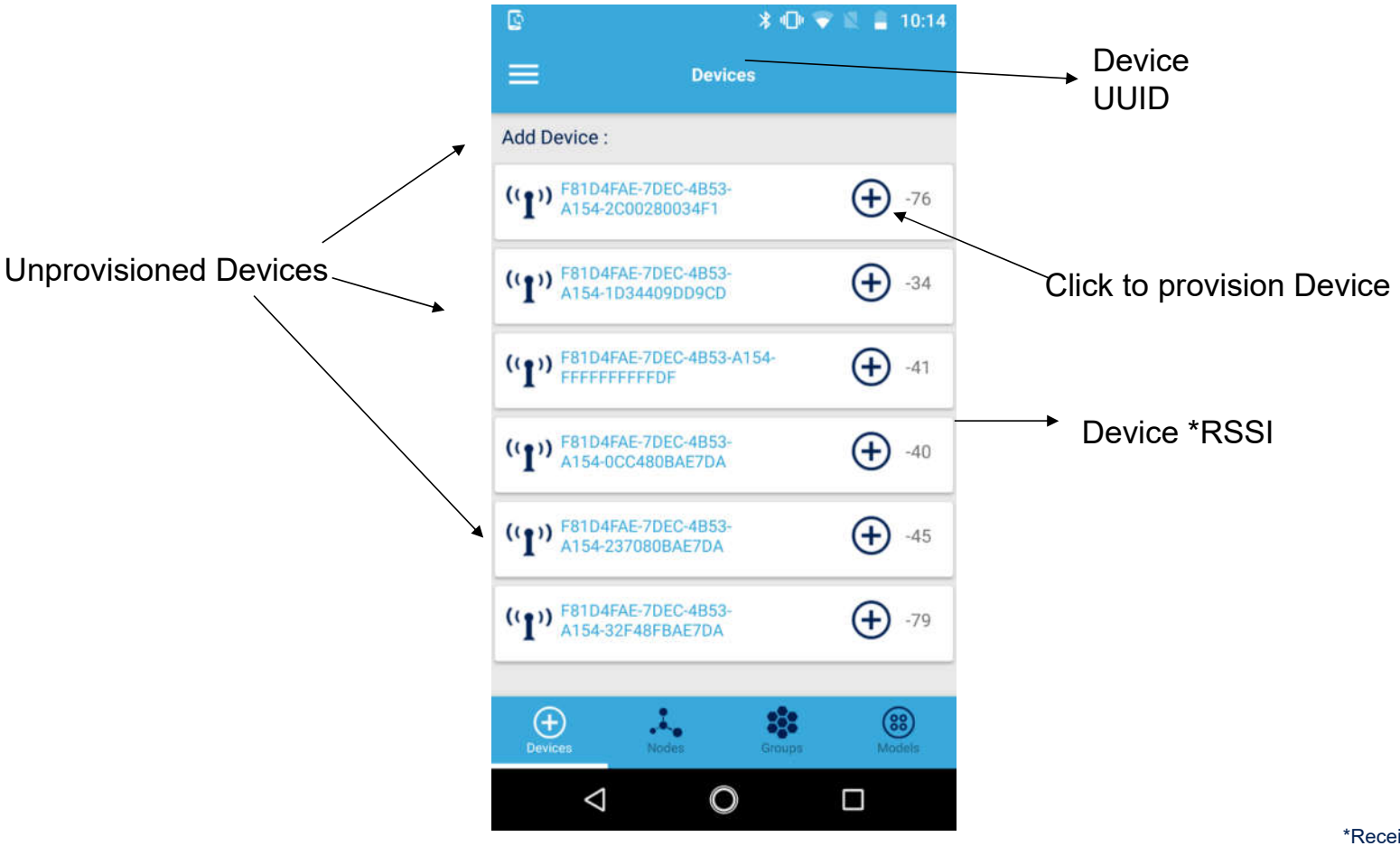
Application opens in the “Nodes” View showing the provisioned devices





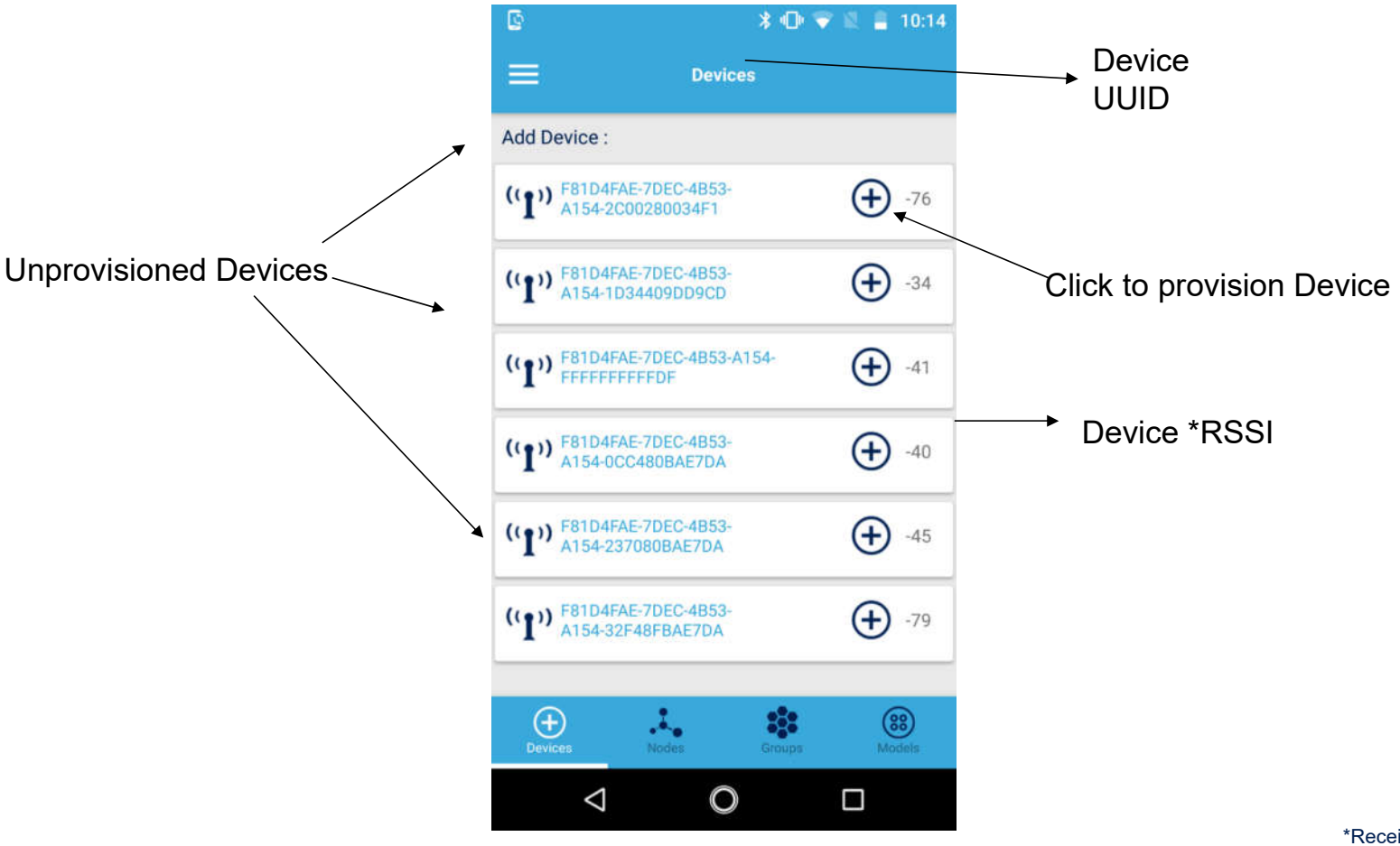
life.augmented

Android App Walk Through : Devices View



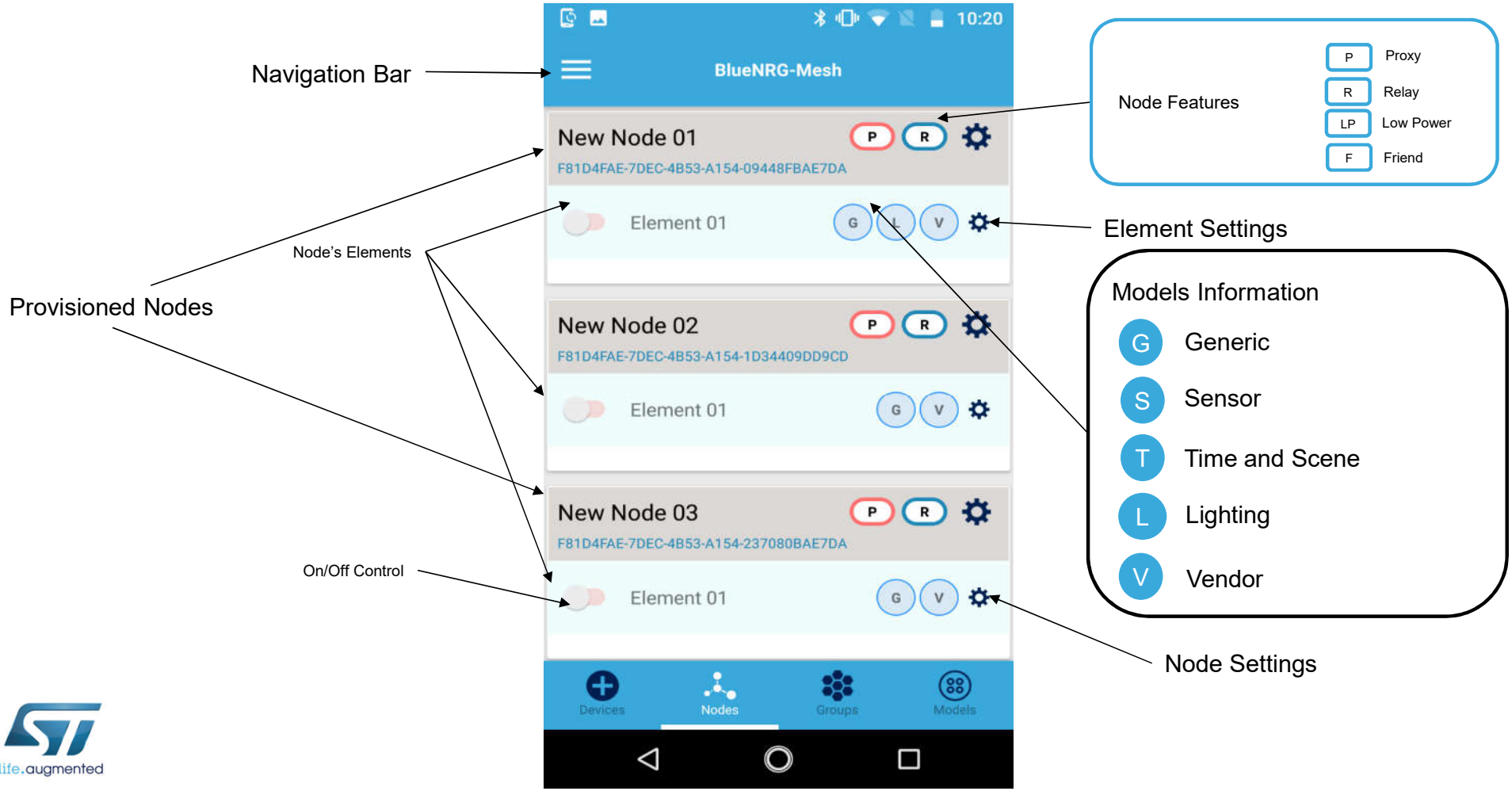
*Received signal strength indication

安卓 App 流程: 发现设备



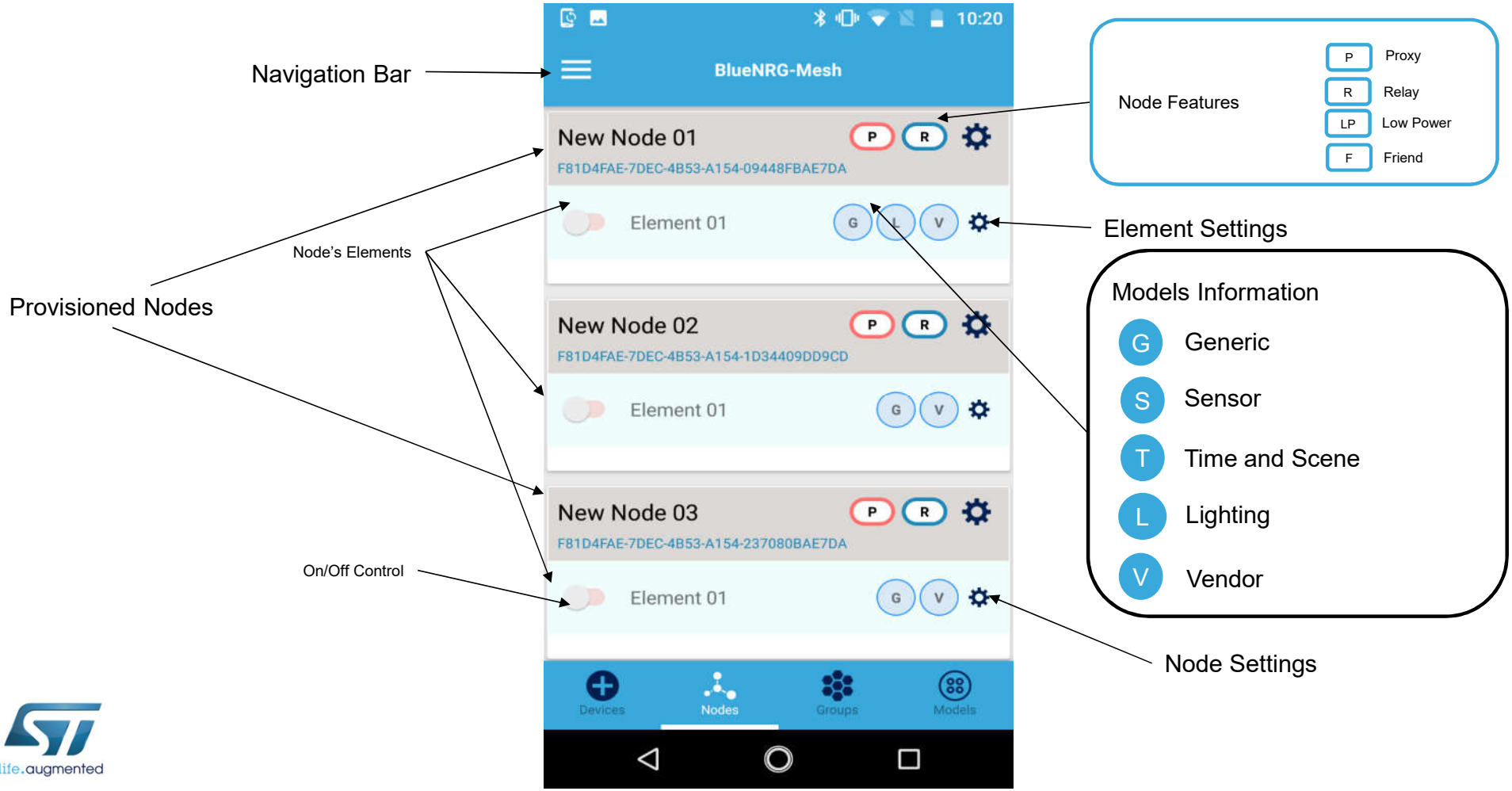
*Received signal strength indication

Android App Walk Through : Nodes View



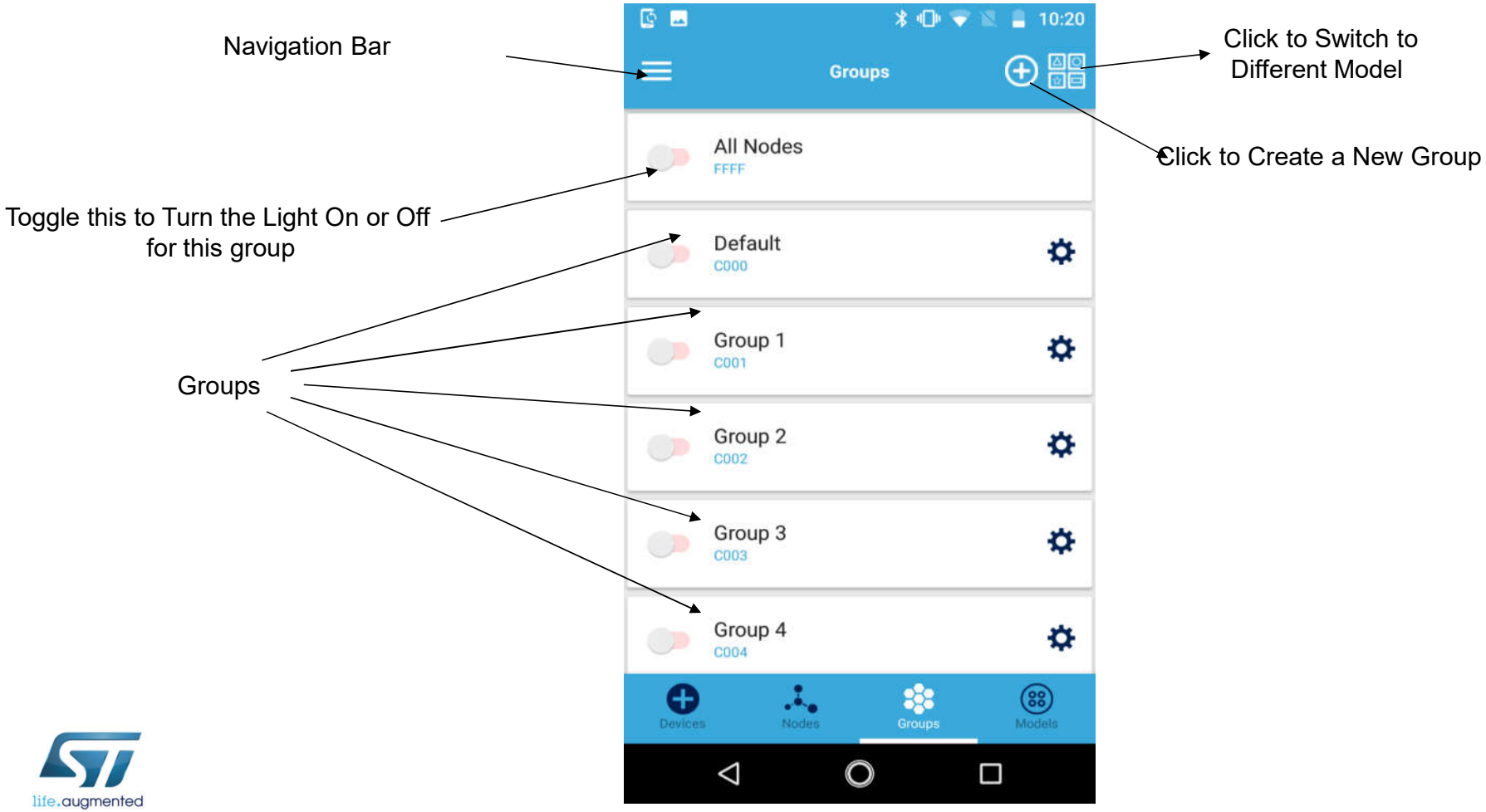
life.augmented

安卓 App 流程: 查看节点



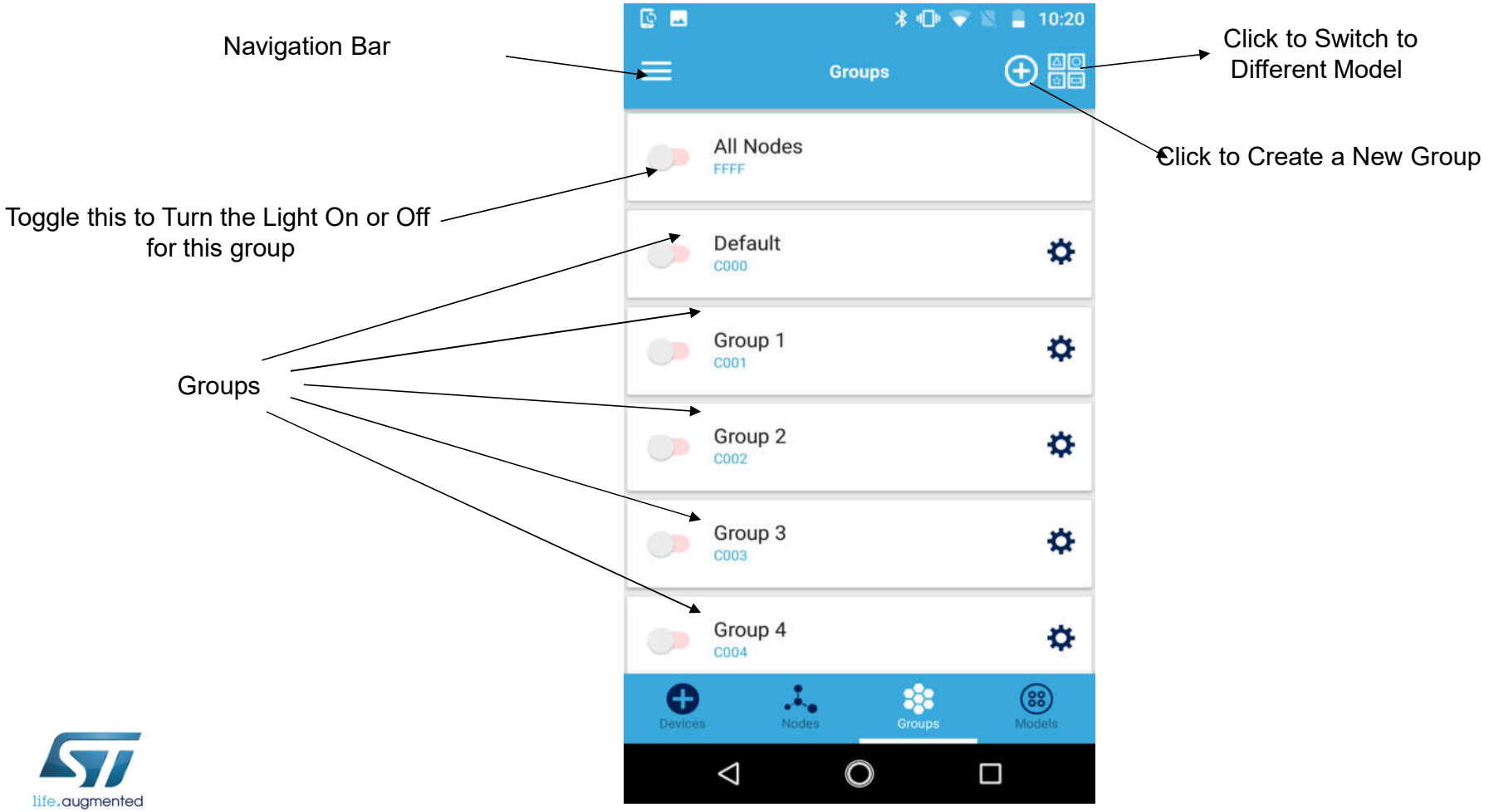
life.augmented

Android App Walk Through : Groups View



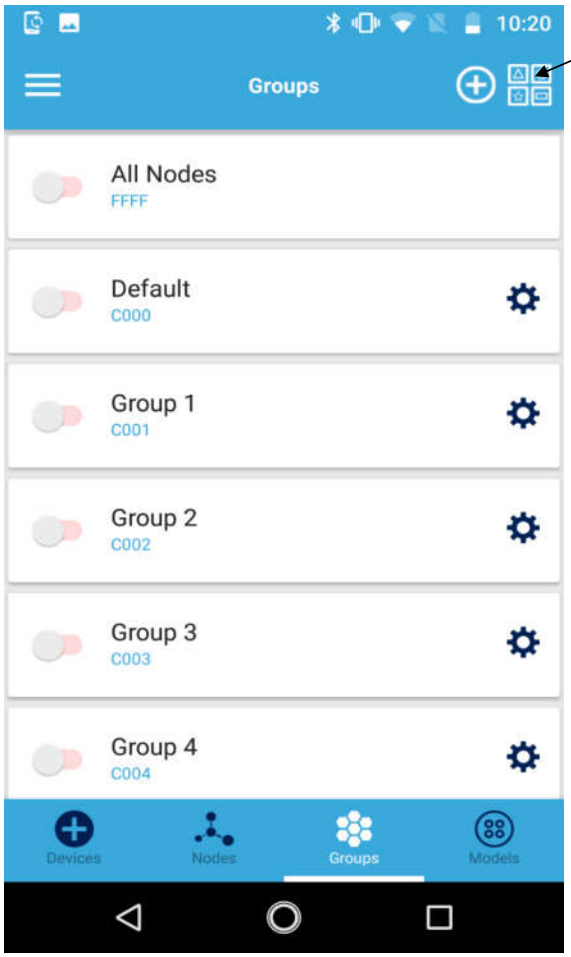
life.augmented

安卓 App 流程: 查看分组

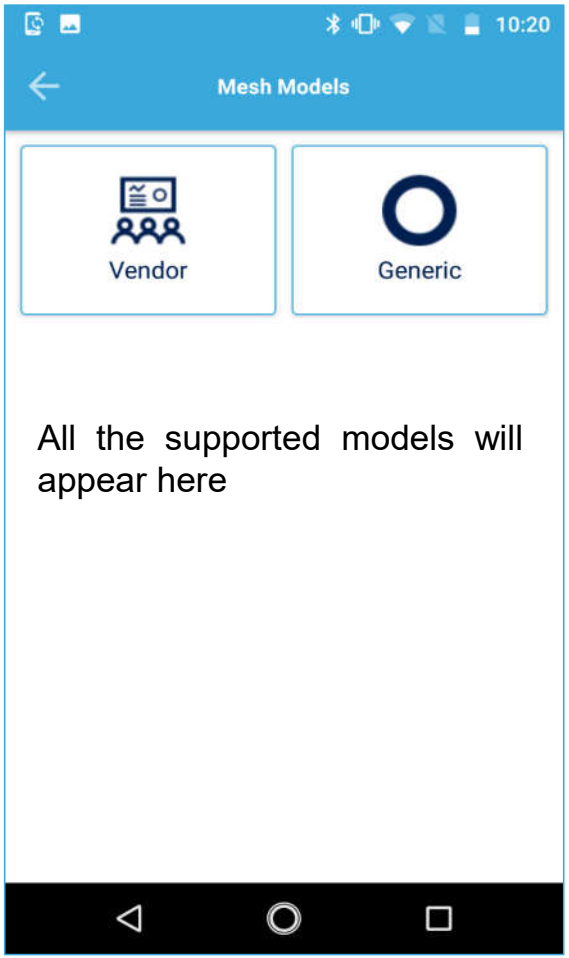


life.augmented

Android App Walk Through : Models View

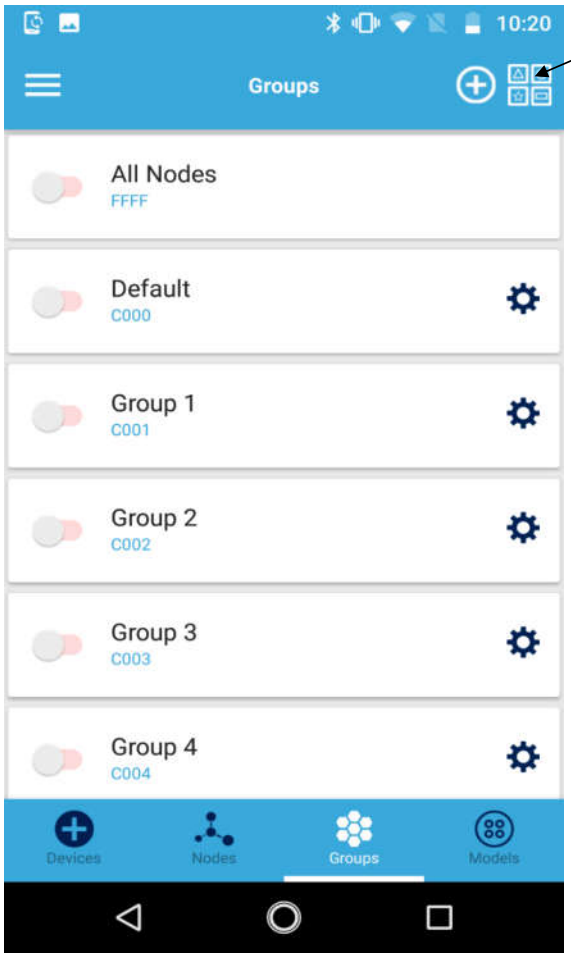


Click this to switch models

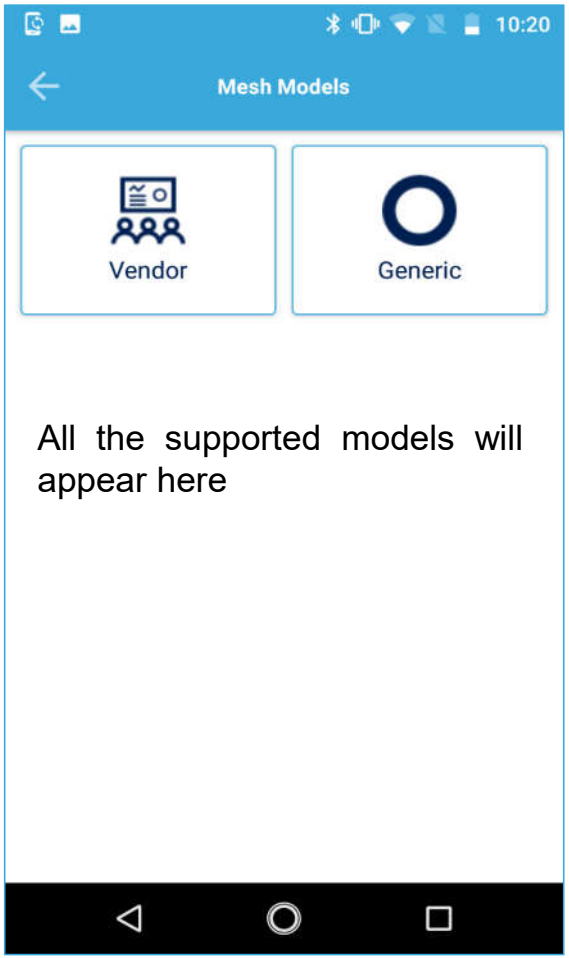


All the supported models will appear here

安卓 App 流程: 查看模型

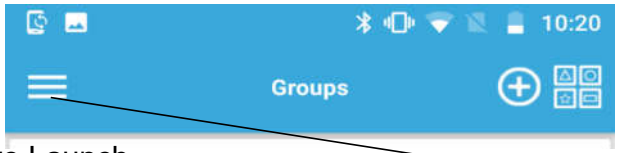


Click this to switch models

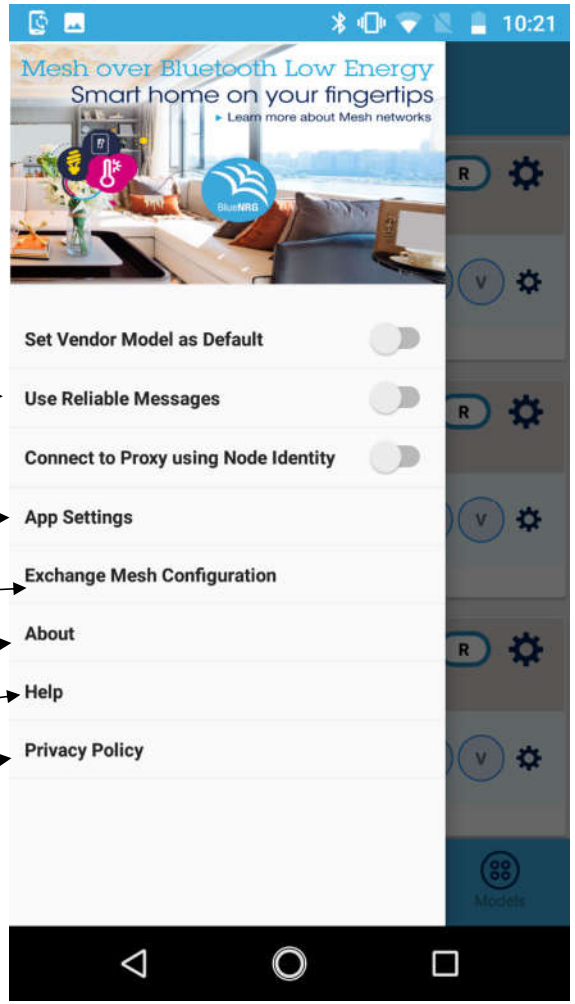


life.augmented

Android App Walk Through : Navigation Bar



Click to Launch Navigation Bar



Switch for switching between Vendor model and Generic Model

Use of Reliable/Unreliable Messages

This will open a new Setting page required by the app as shown in the Slide#11

Option to share Provisioner Database Configuration via Mail or Cloud.

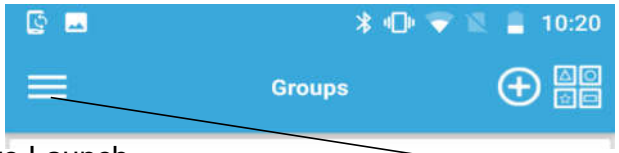
Link to the App Version

Link to the App UM/Tutorials

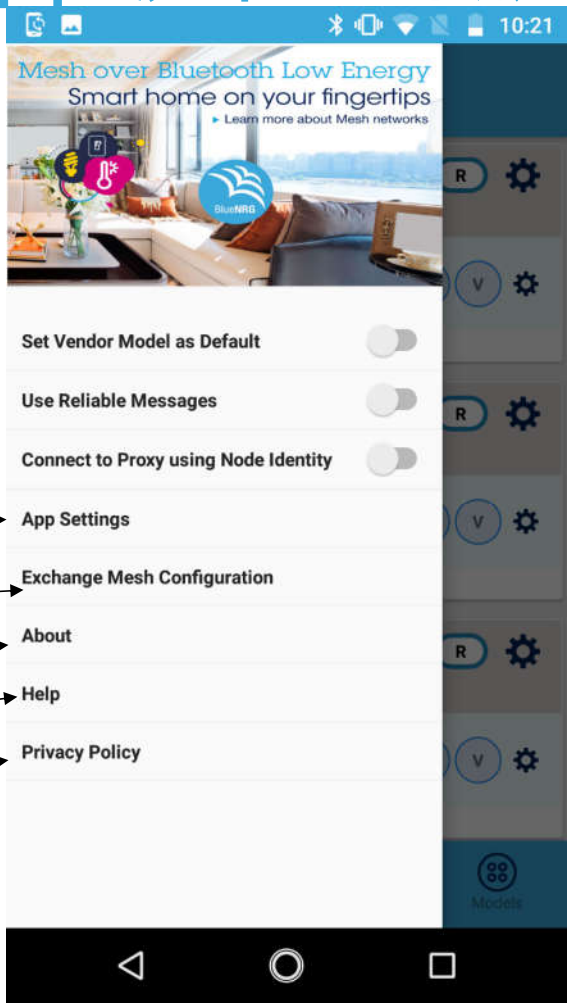
Link to the ST Privacy Policy

life.augmented

安卓 App 流程: 导航 Bar



Click to Launch Navigation Bar



Switch for switching between Vendor model and Generic Model

Use of Reliable/Unreliable Messages

This will open a new Setting page required by the app as shown in the Slide#11

Option to share Provisioner Database Configuration via Mail or Cloud.

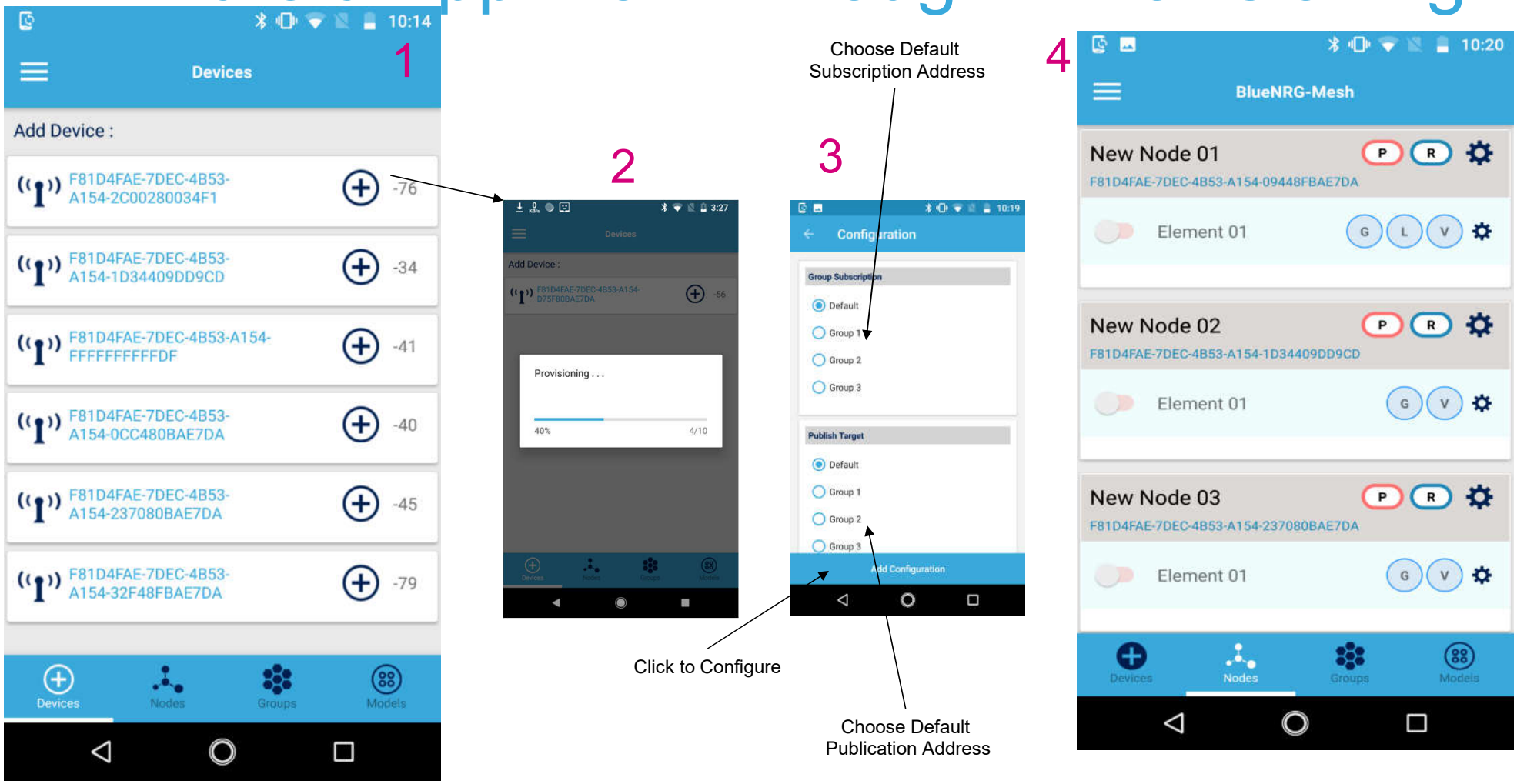
Link to the App Version

Link to the App UM/Tutorials

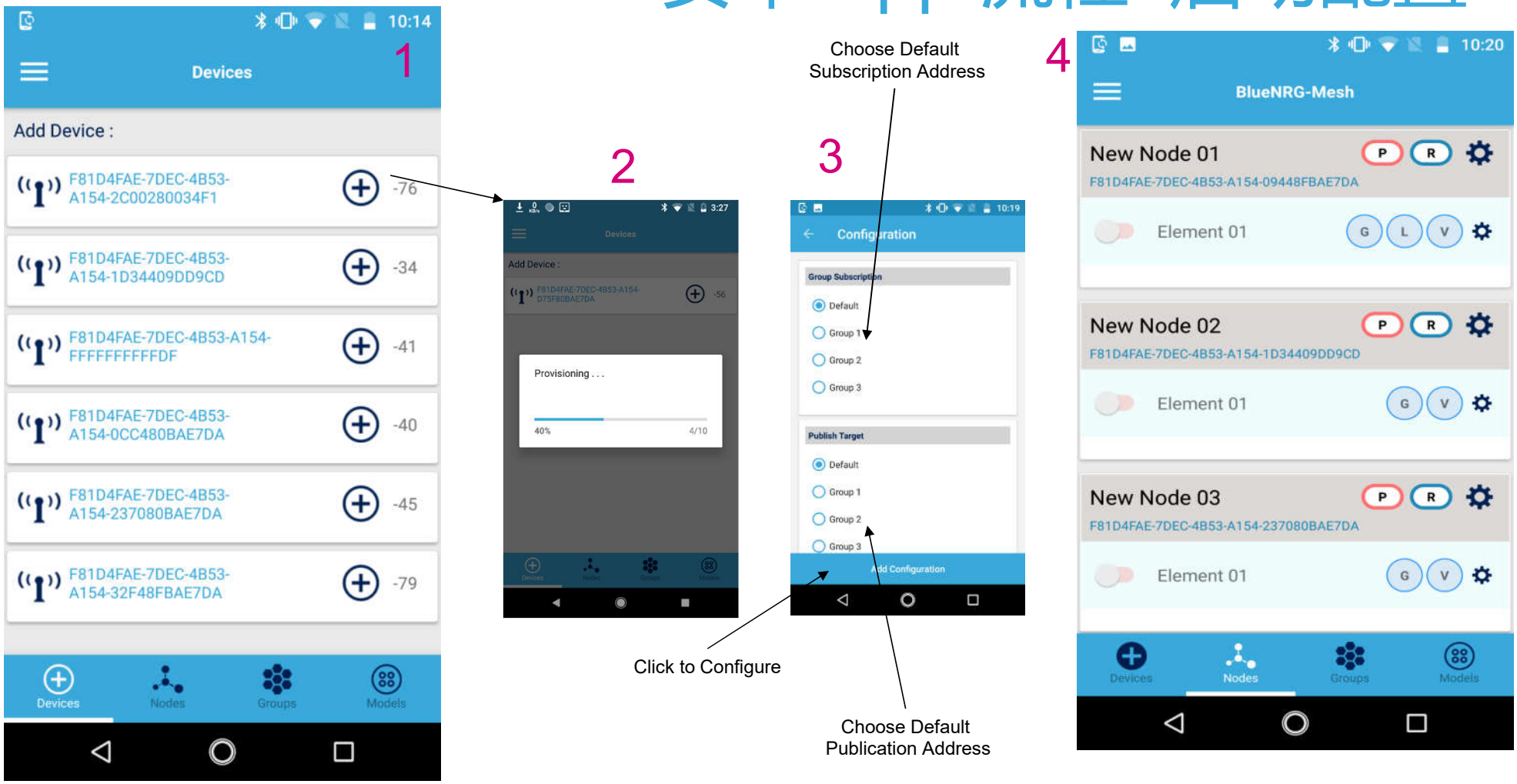
Link to the ST Privacy Policy

life.augmented

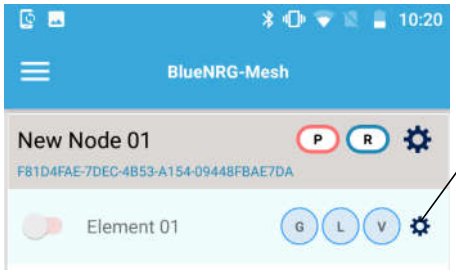
Android App Walk Through : Provisioning

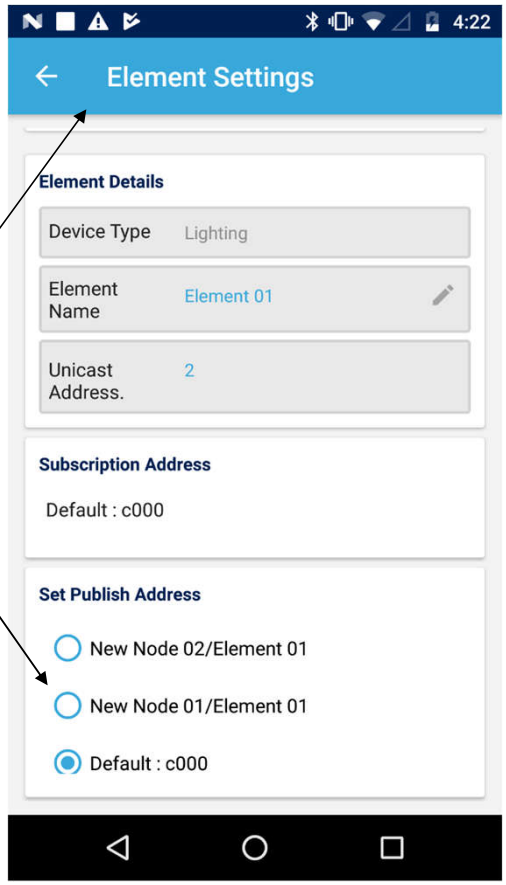


安卓 App 流程: 启动配置



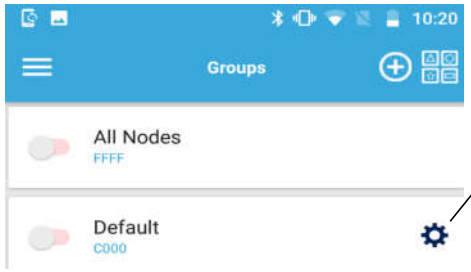
Change Default Configuration : Publication



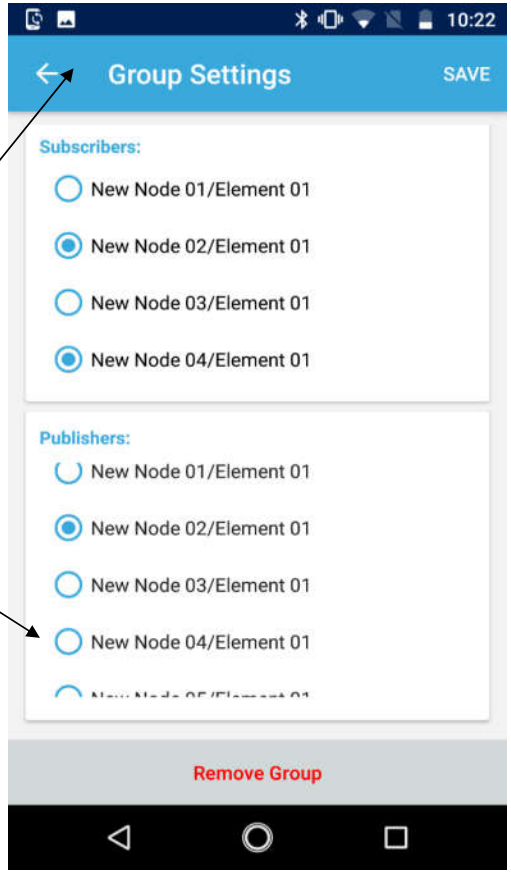


Select the Publish address.
Only one can be selected.

Setting Publication Using the Element Settings



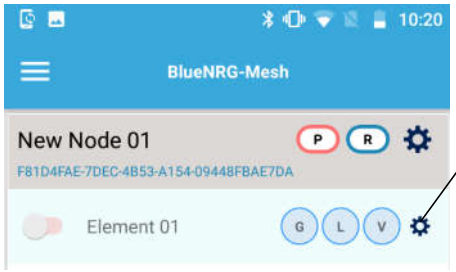
Select the Element
Which will be given the publish address.
Multiple elements can be selected

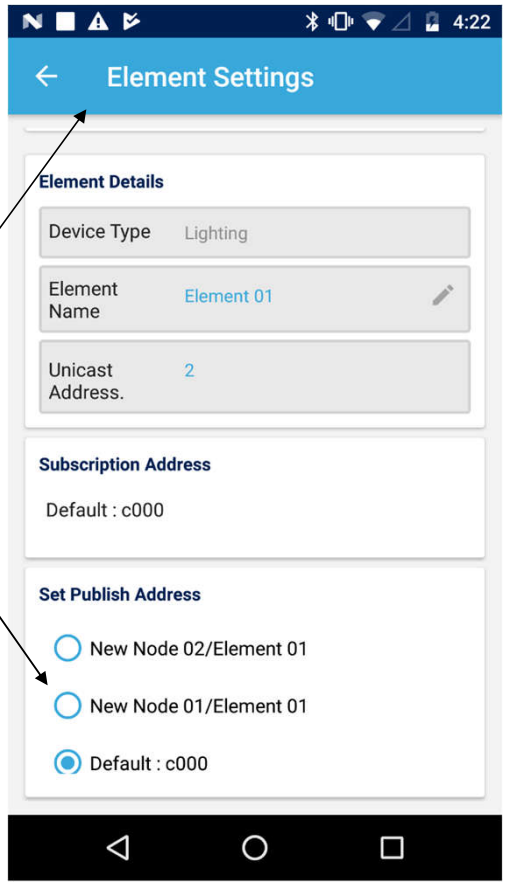


Setting Publication Using the Group Settings

life.augmented

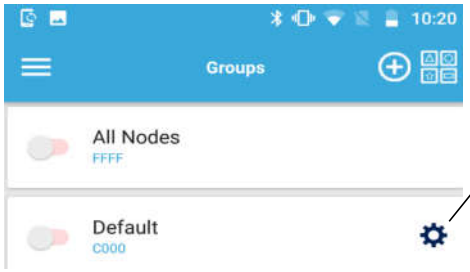
改变默认设置：发布



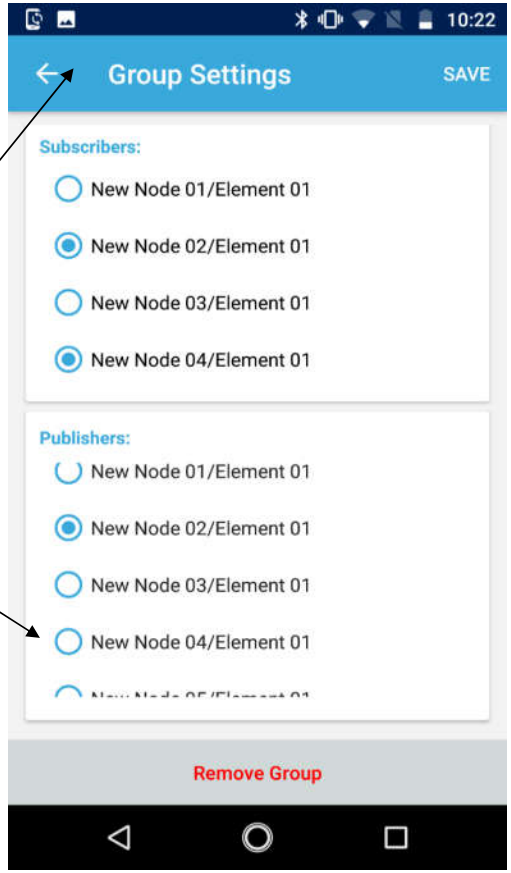


Select the Publish address.
Only one can be selected.

Setting Publication Using the Element Settings



Select the Element
Which will be given the publish address.
Multiple elements can be selected

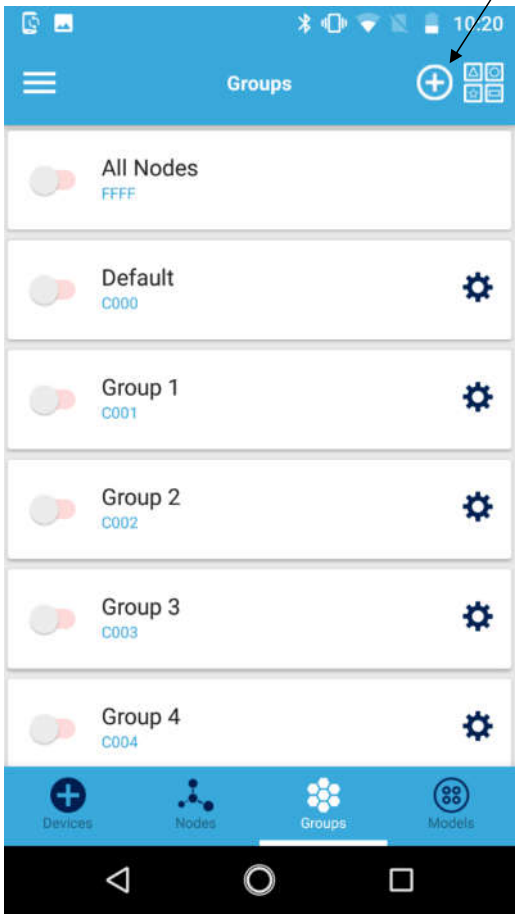


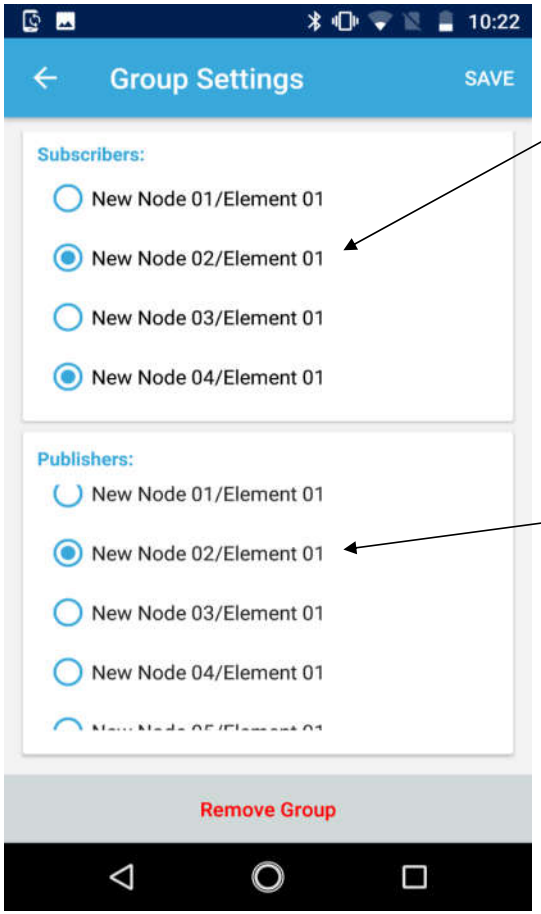
Setting Publication Using the Group Settings

life.augmented

Change Default Configuration : Subscription

Click to Create New Group





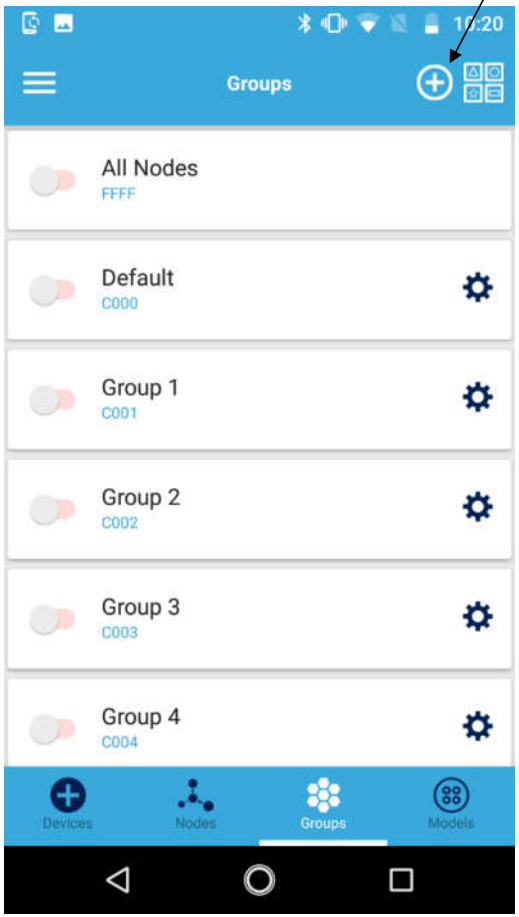
Click any Element to Add it to Group

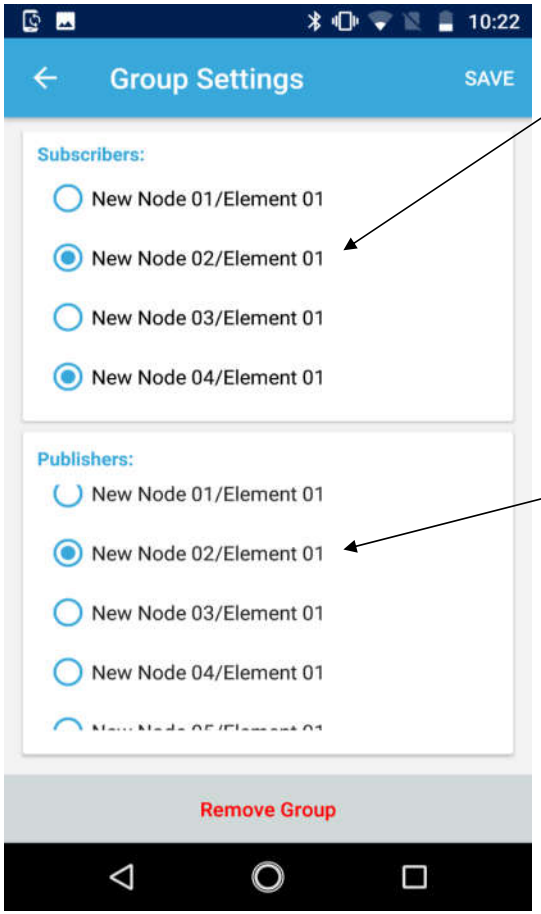
Click any Element to Give Publication Address to it. Publication address can Also be given from the Element Setting.

life.augmented

改变默认设置：订阅

点击创建新的组



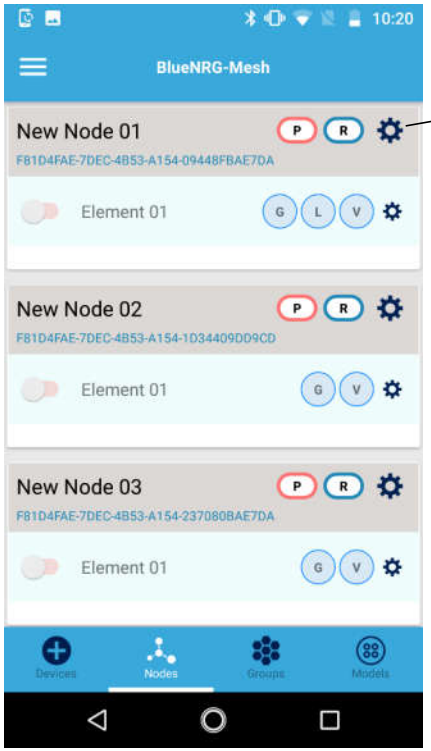


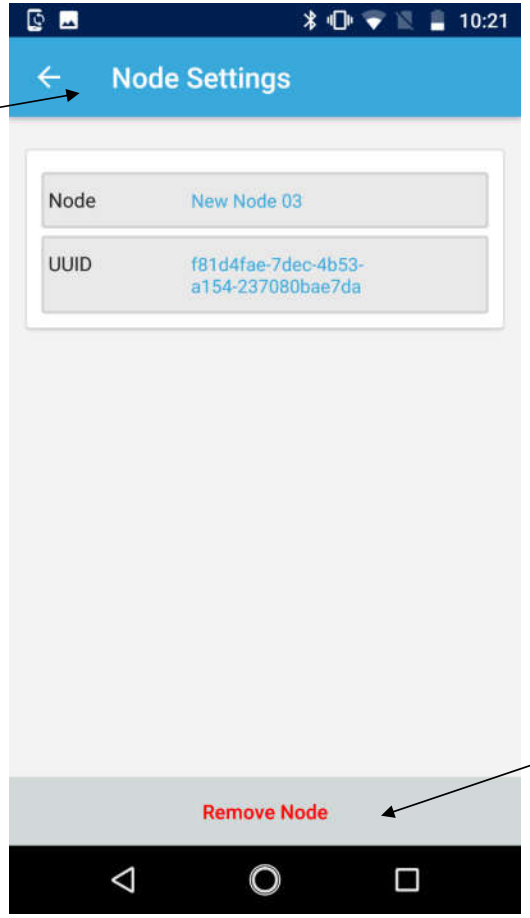
点击任何元素将它添加到组

点击任何元素给出发布地址
发布地址也可以从元素设定中给出

life.augmented

Unprovision

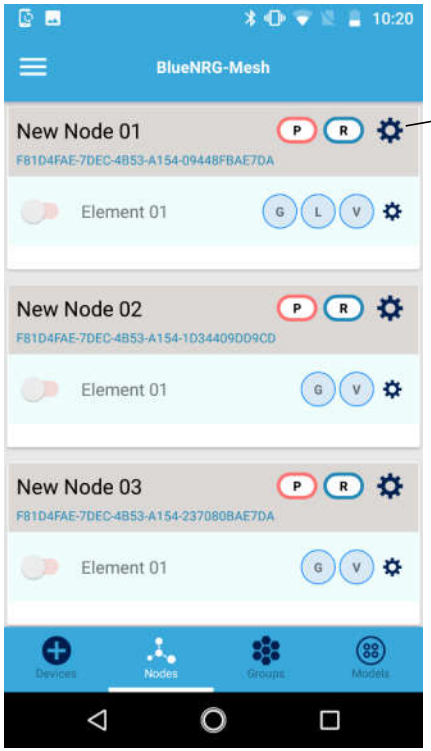


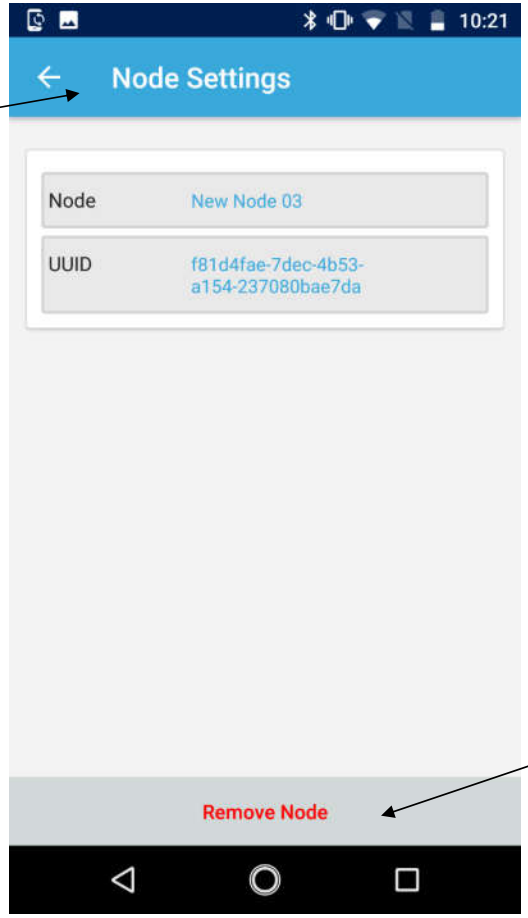


Click to unprovision the Node

life.augmented

解除配置

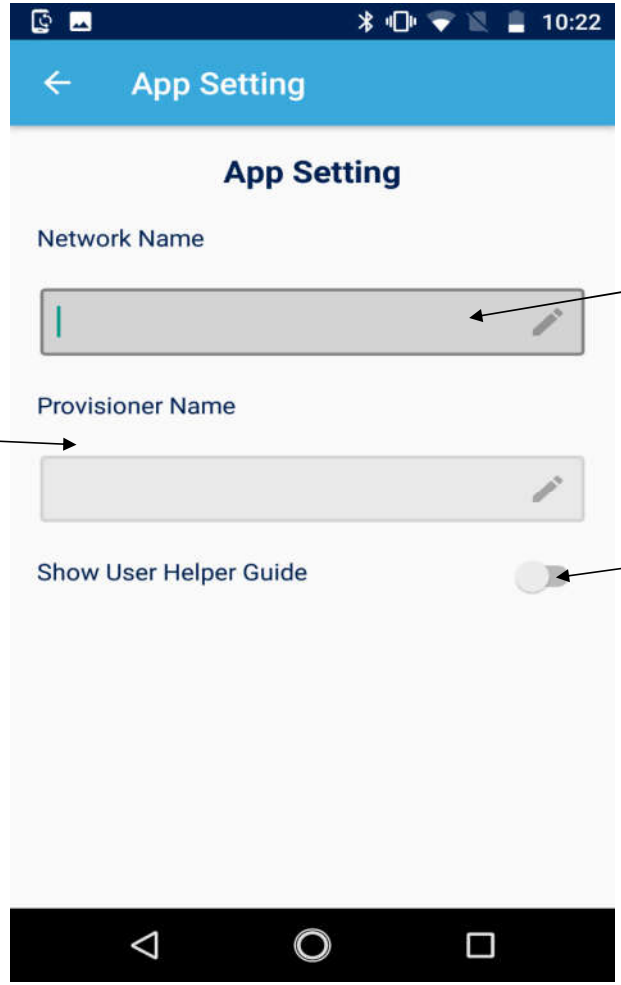




点击解除配置

life.augmented

Settings : App Settings



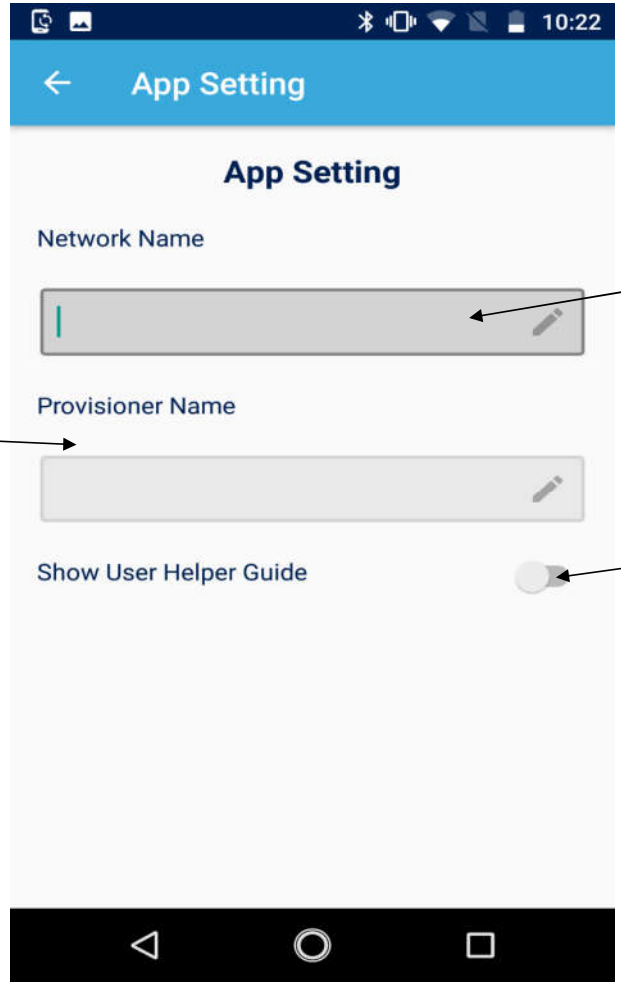
Name of the Network

Provisioner's Name

Tutorial Screen On/Off

life.augmented

设置：app设置



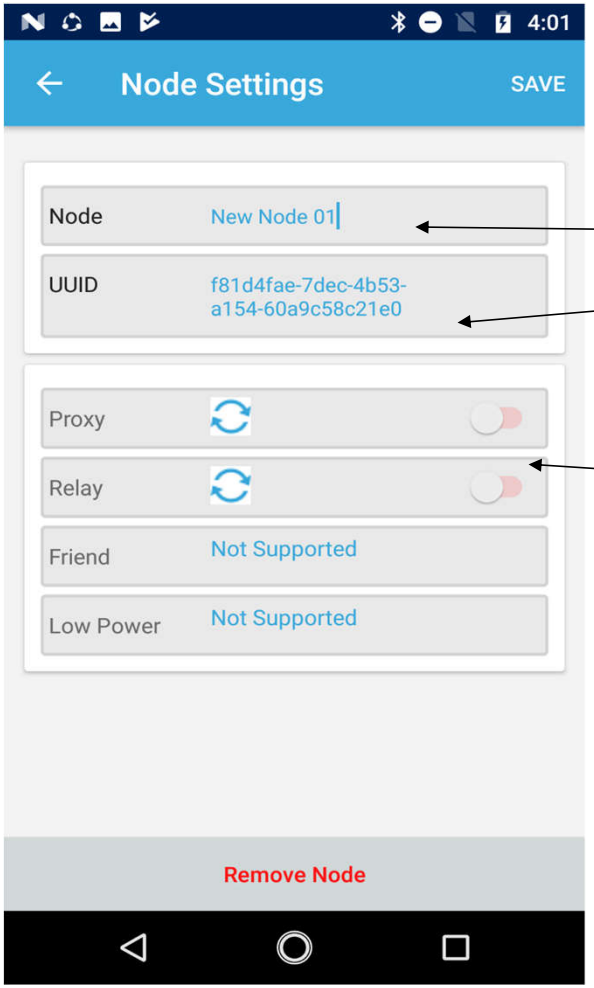
Name of the Network

Provisioner's Name

Tutorial Screen On/Off

life.augmented

Settings : Node Settings



Node Name

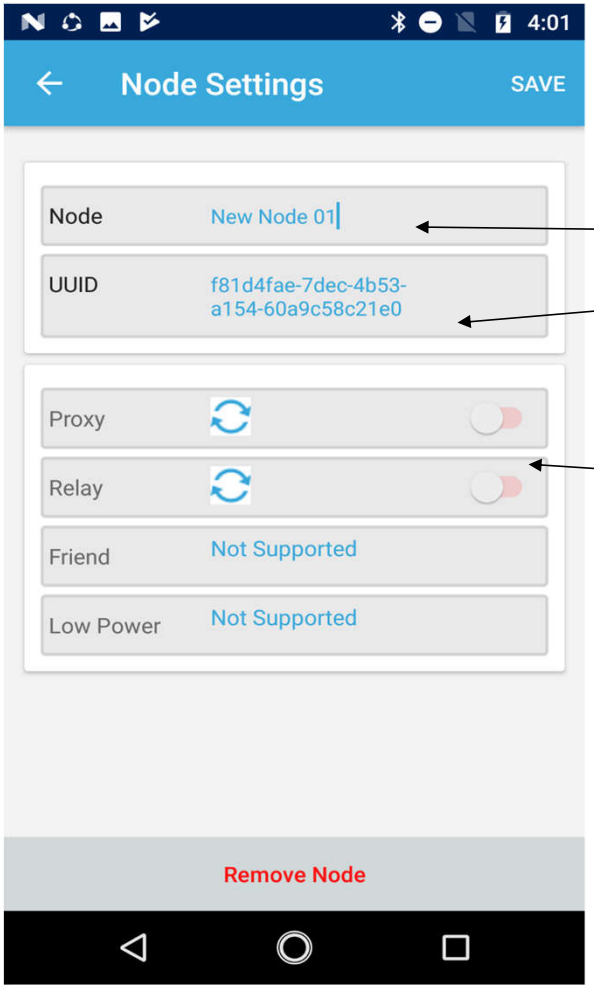
Node UUID

Node Features Enabled/ Disabled

Types of Nodes Supported / Not Supported

life.augmented

设置：节点设置



Node Name

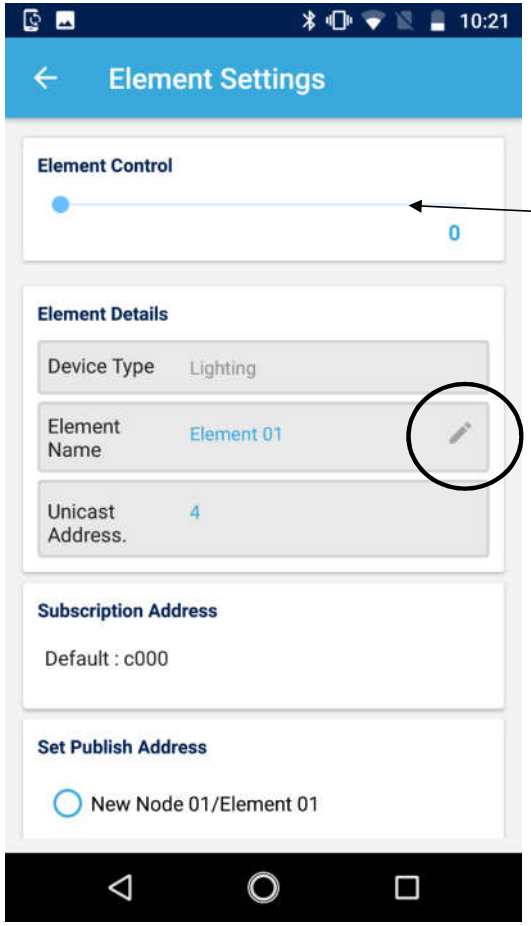
Node UUID

Node Features Enabled/ Disabled

Types of Nodes Supported / Not Supported

life.augmented

Settings : Element Settings

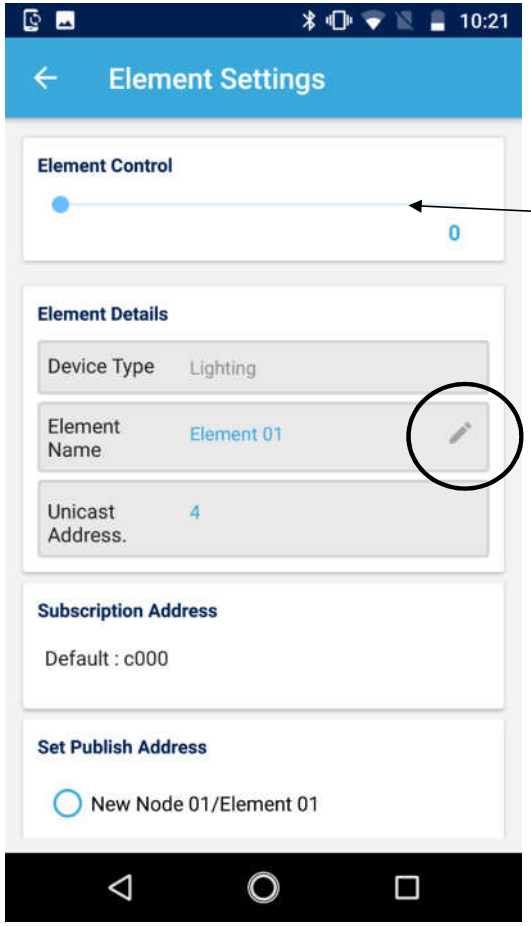


editable

Intensity Control in the steps of 0% -100 %.Both Generic or Vendor command can be sent based on the navigation bar settings.

life.augmented

设置：元素设置

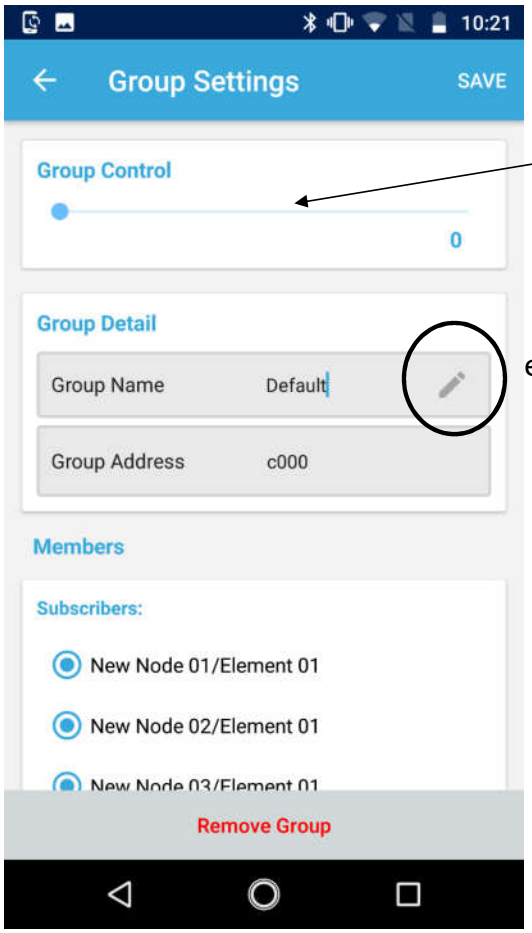


editable

亮度控制0% -100 %.通用和厂商命令都可以基于导航bar设置被发送

life.augmented

Settings : Group Settings

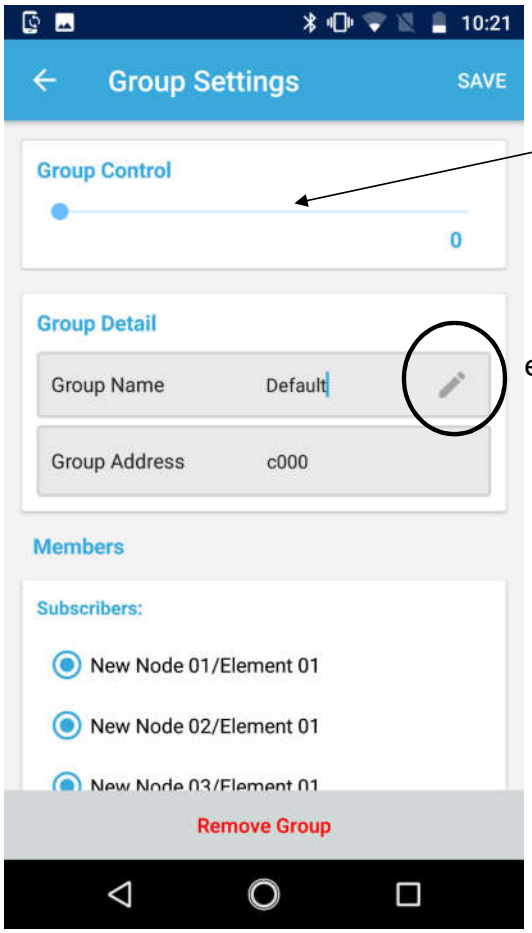


Intensity Control in the steps of 0-100 %. Generic or Vendor Command can be sent based on the navigation bar settings.

editable

Group Subscribers list

设置：分组设置



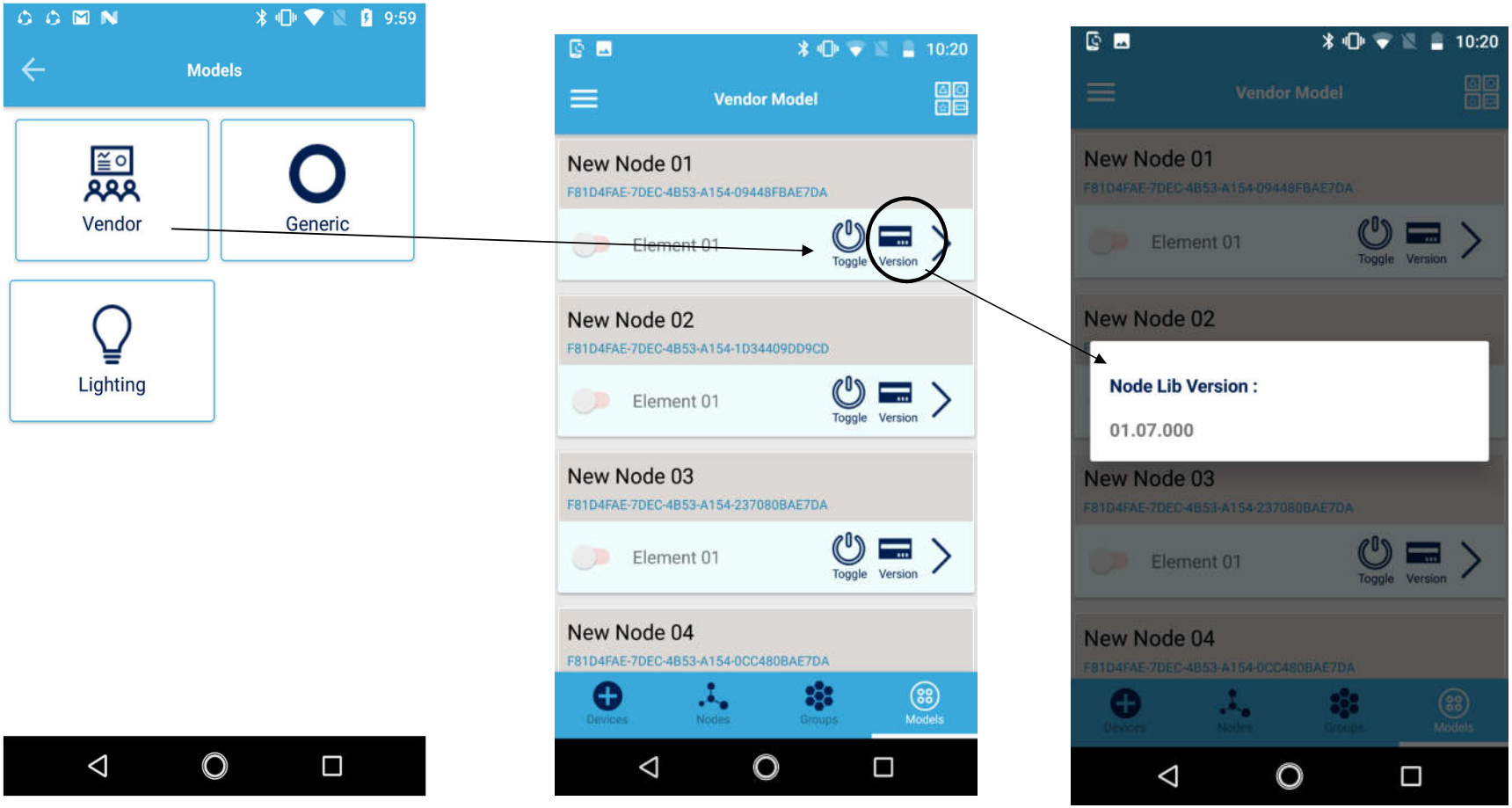
亮度控制0% -100 %.通用和厂商命令都可以基于导航bar设置被发送

editable

Group Subscribers list

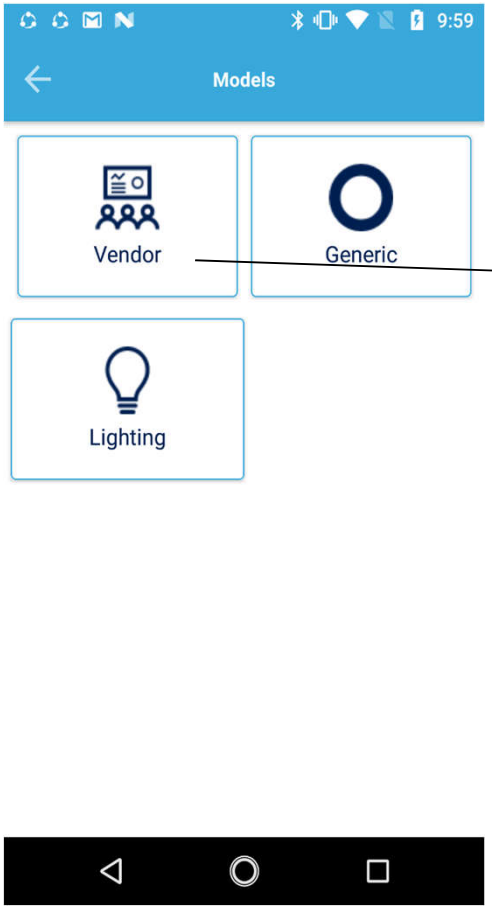
life.augmented

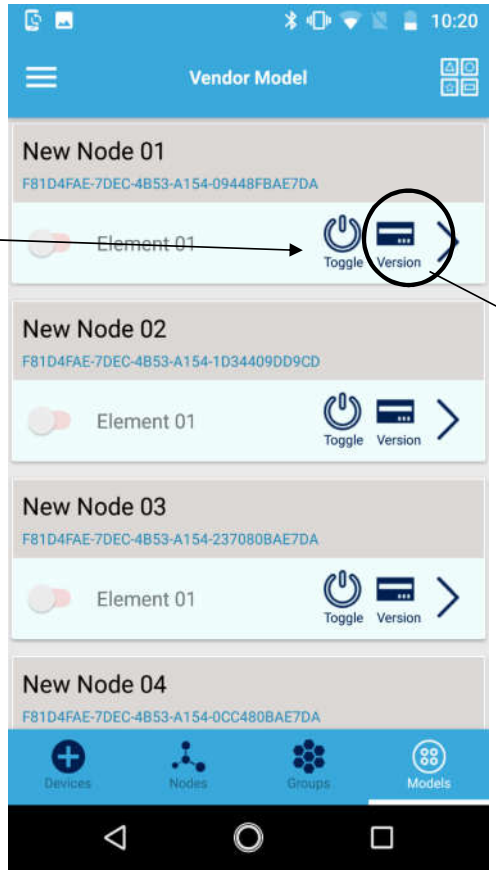
Models : Vendor

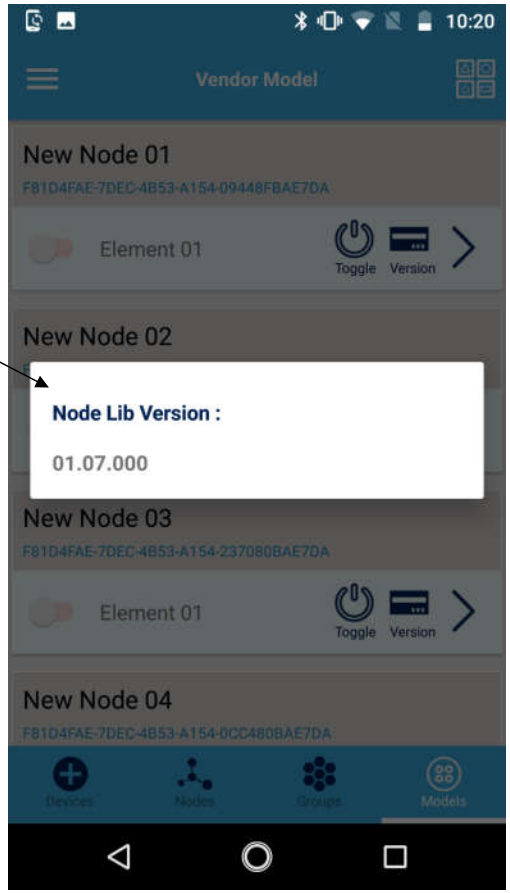


life.augmented

模型：厂商

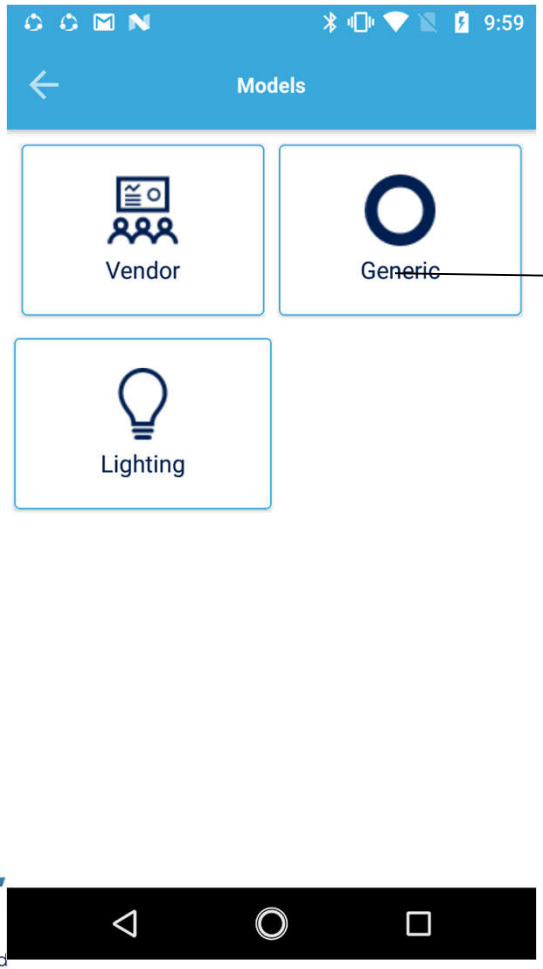


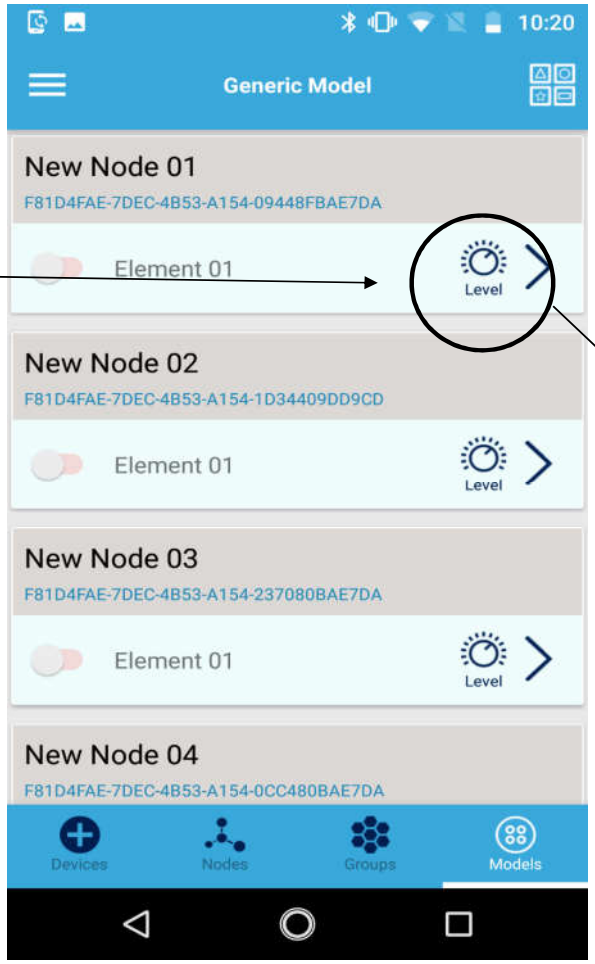


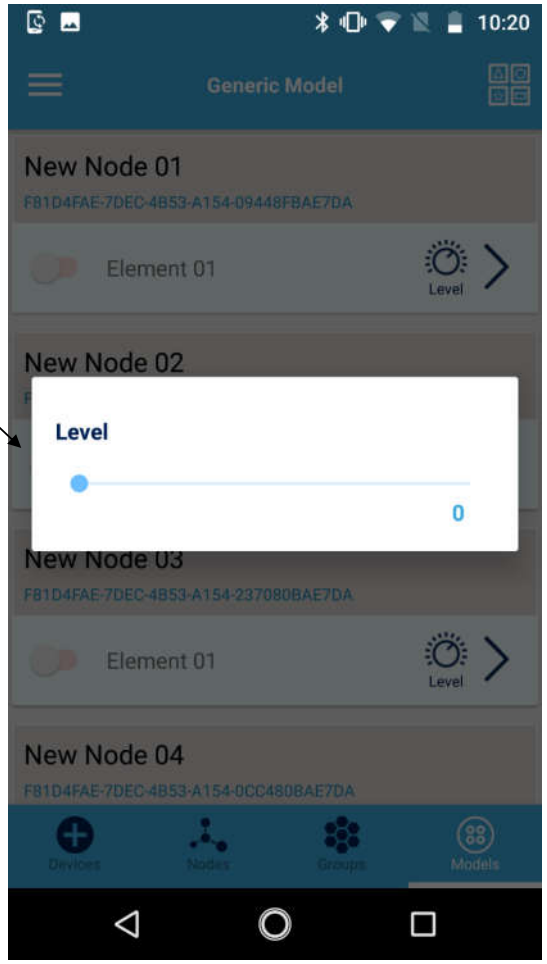


life.augmented

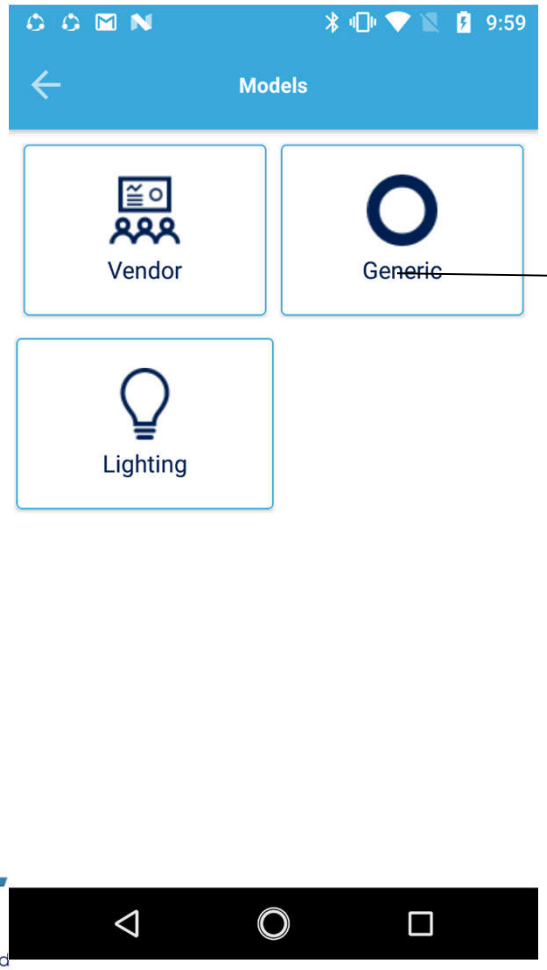
Models : Generic

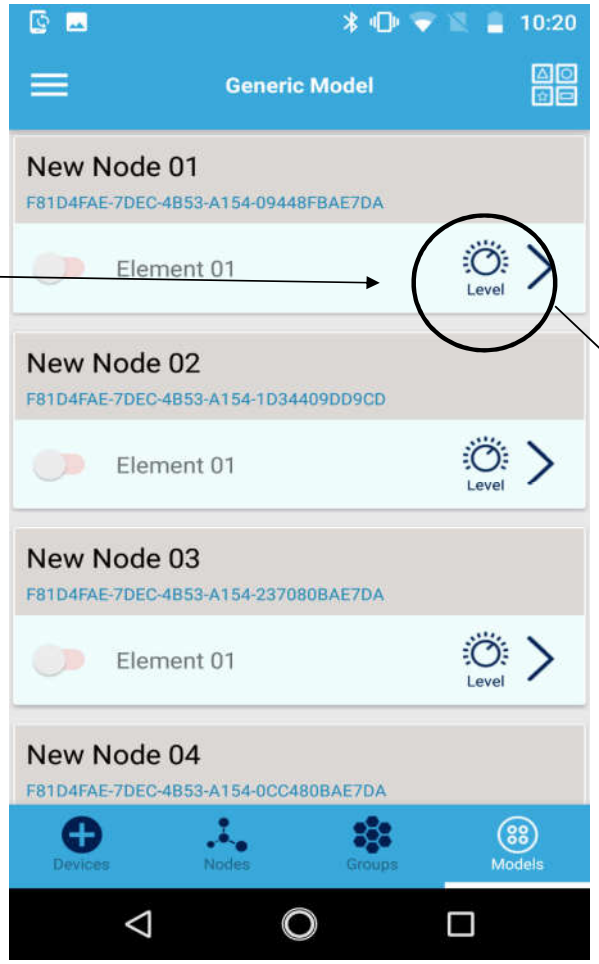


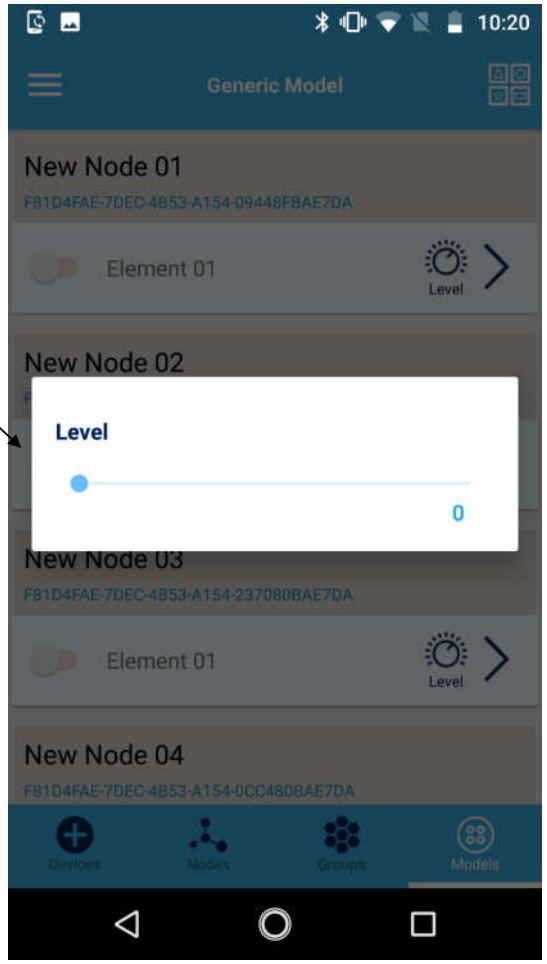




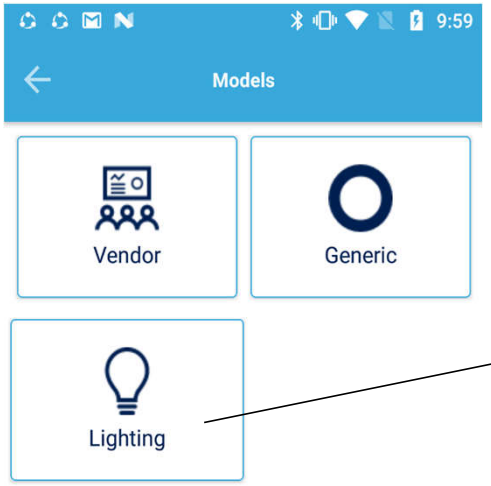
模型：通用


9:59
Models
Vendor
Generic
Lighting

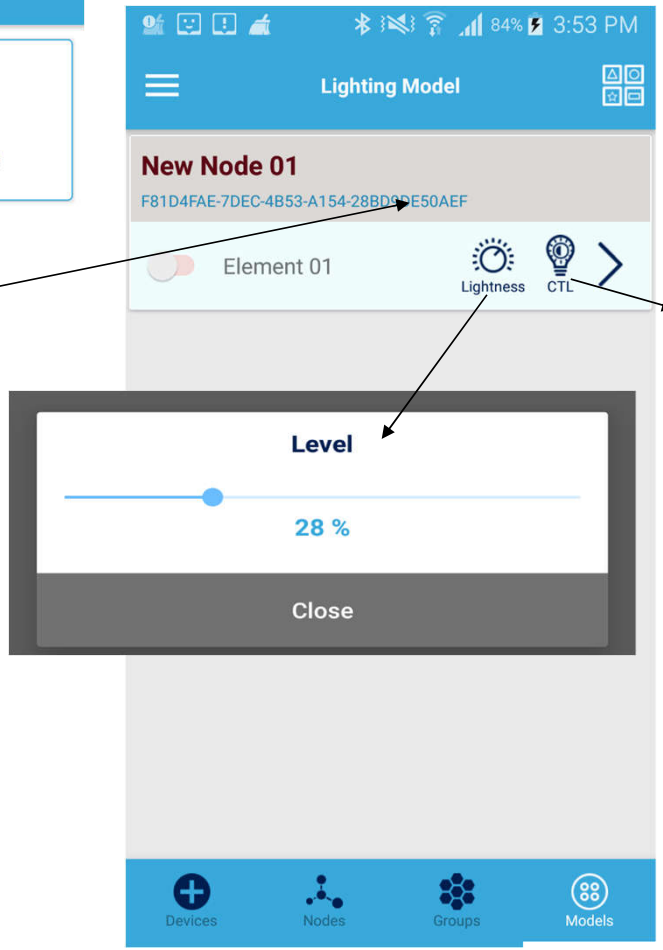

10:20
Generic Model
New Node 01
F81D4FAE-7DEC-4B53-A154-09448FBAE7DA
Element 01
Level
New Node 02
F81D4FAE-7DEC-4B53-A154-1D34409DD9CD
Element 01
Level
New Node 03
F81D4FAE-7DEC-4B53-A154-237080BAE7DA
Element 01
Level
New Node 04
F81D4FAE-7DEC-4B53-A154-0CC480BAE7DA
Devices
Nodes
Groups
Models

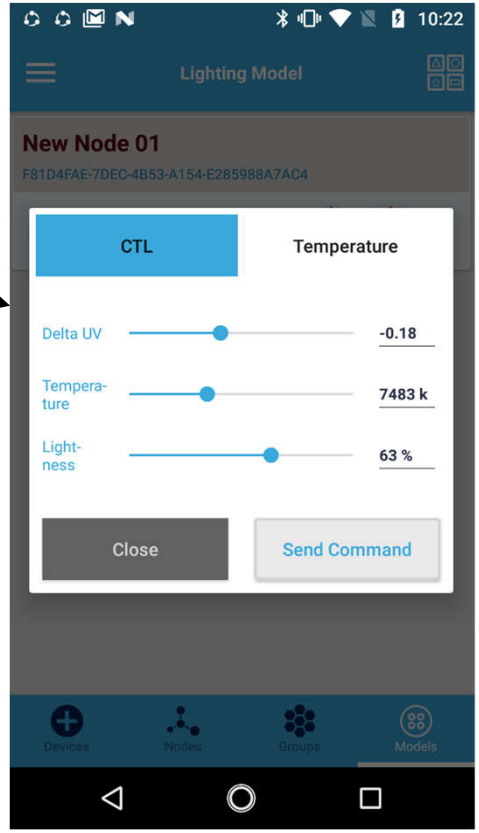

10:20
Generic Model
New Node 01
F81D4FAE-7DEC-4B53-A154-09448FBAE7DA
Element 01
Level
New Node 02
Level
0
New Node 03
F81D4FAE-7DEC-4B53-A154-237080BAE7DA
Element 01
Level
New Node 04
F81D4FAE-7DEC-4B53-A154-0CC480BAE7DA
Devices
Nodes
Groups
Models

Models : Lighting

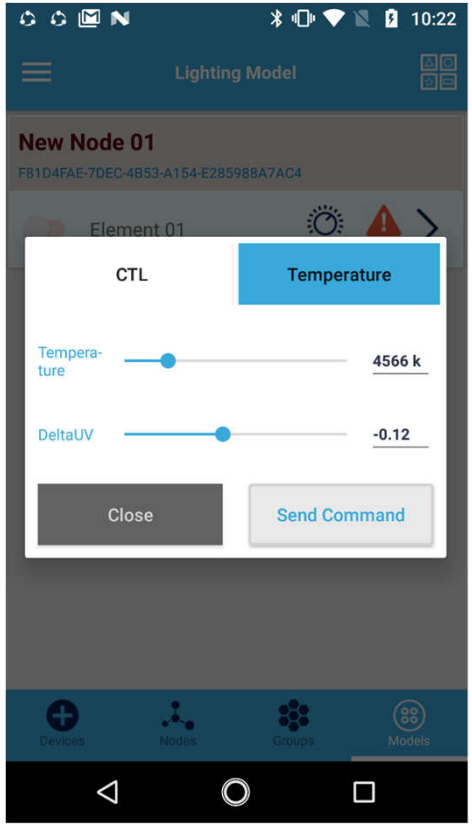


Lighting Lightness control





Lighting CTL* control

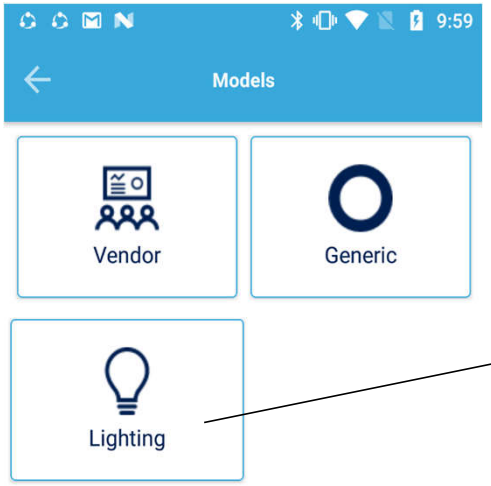


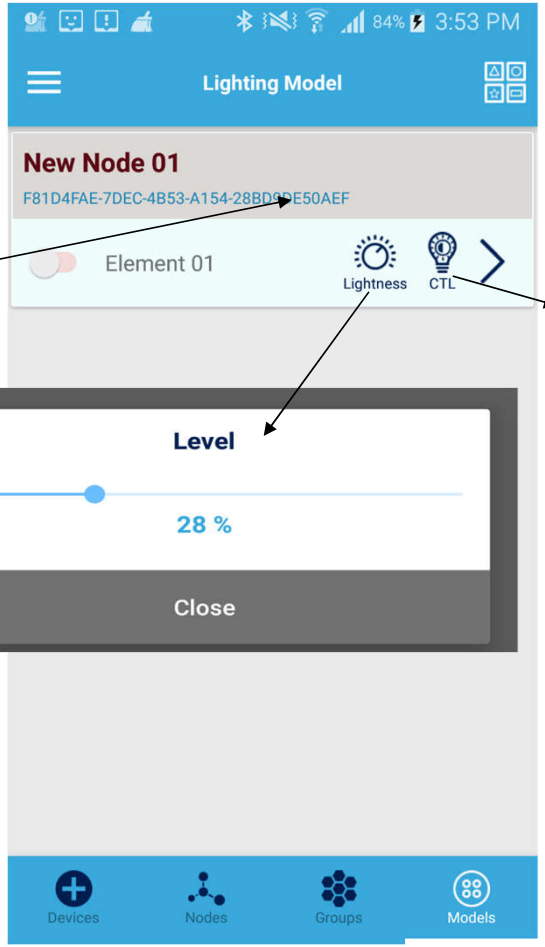
Lighting CTL Temperature control

CTL* Color , Temperature and Lightness

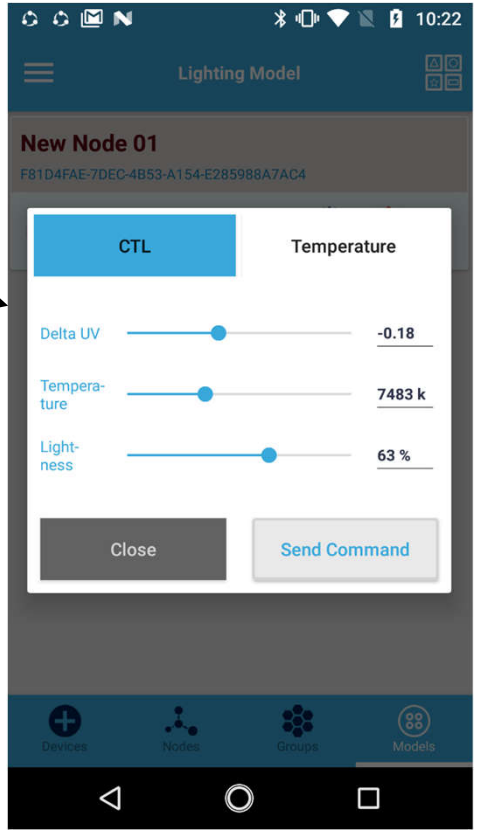
life.augmented

模型：灯

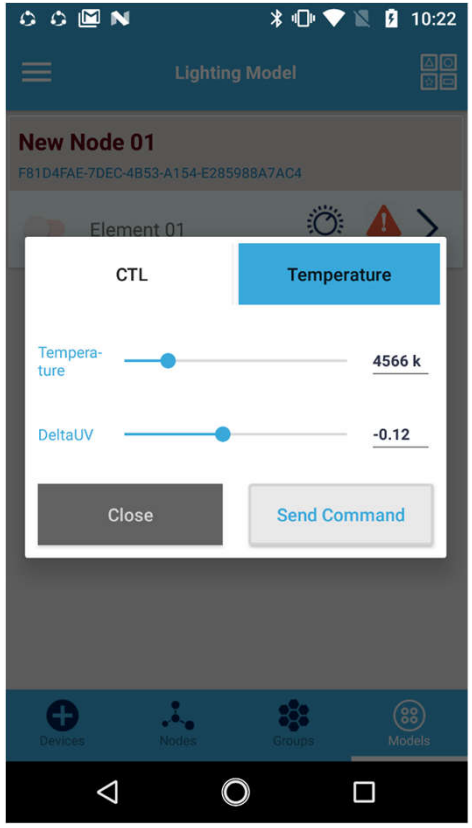




Lighting Lightness control



Lighting CTL* control

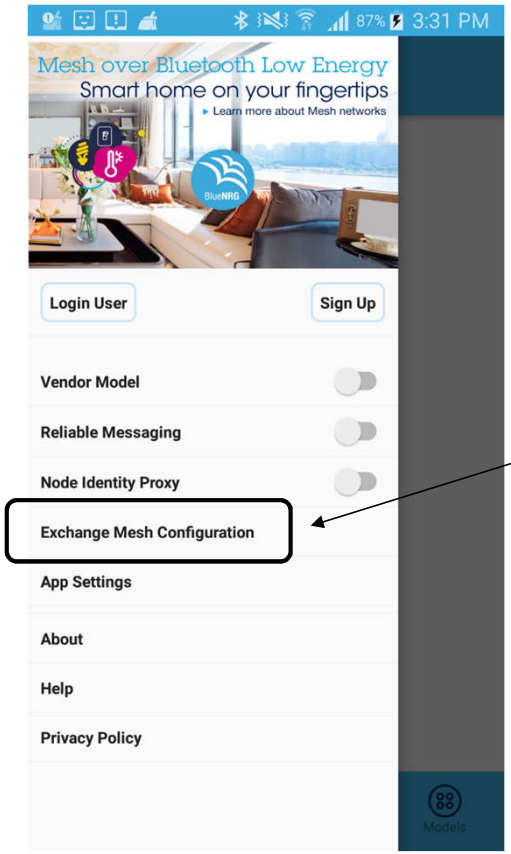


Lighting CTL Temperature control

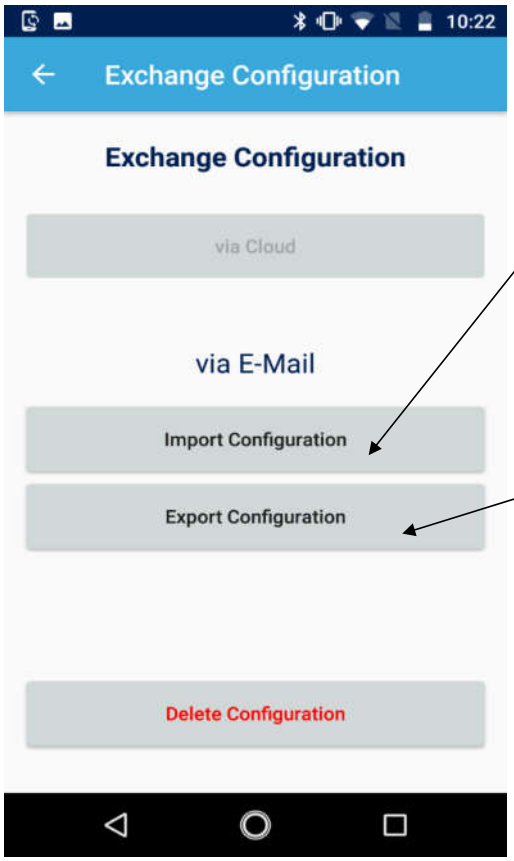
CTL* Color , Temperature and Lightness

life.augmented

Exchange Mesh Configuration

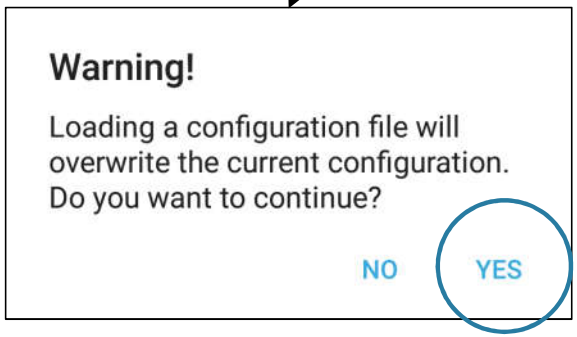


Click this



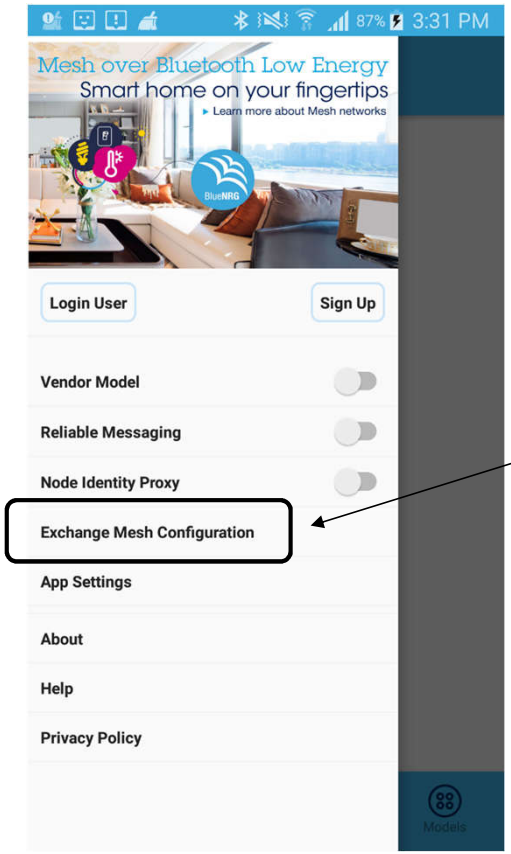
It will import JSON configuration from the Download Folder(JSON previously downloaded)

It will open the default mail client to transfer JSON Configuration

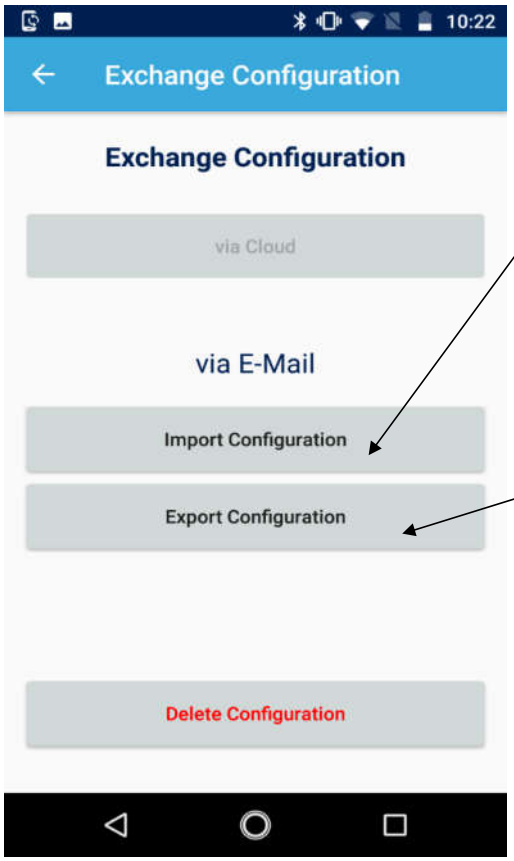


life.augmented

交换Mesh设置

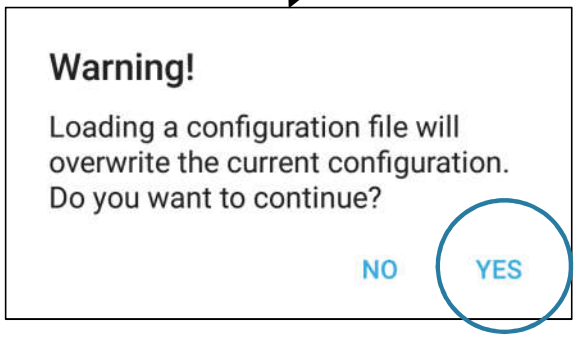


Click this



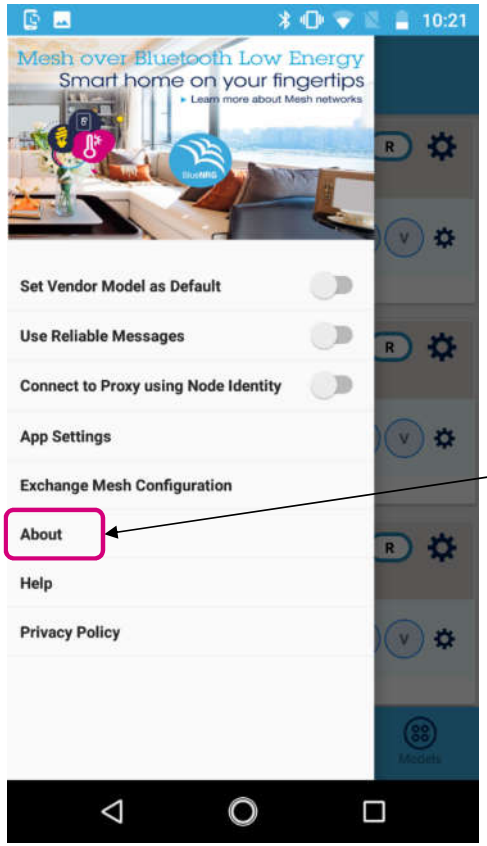
它将会从下载的文件夹（JSON先前下载好的）导入JSON设置

它将会打开默认邮件代理端去发送JSON设置



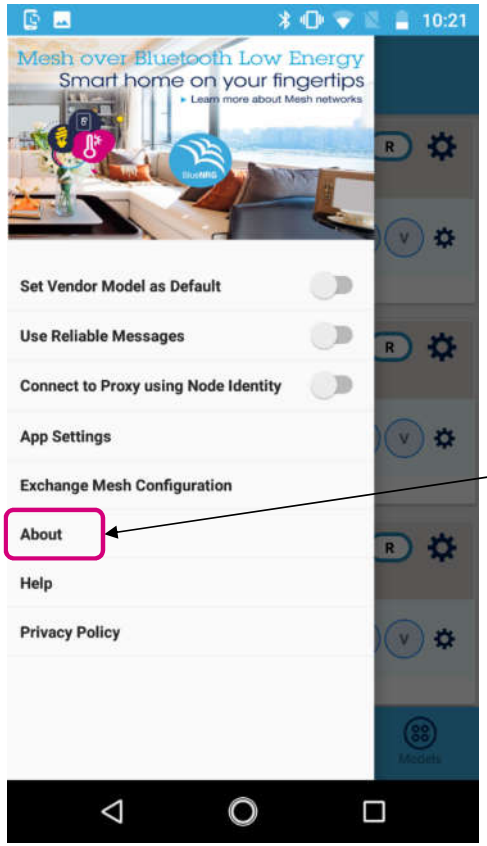
life.augmented

App Version



life.augmented

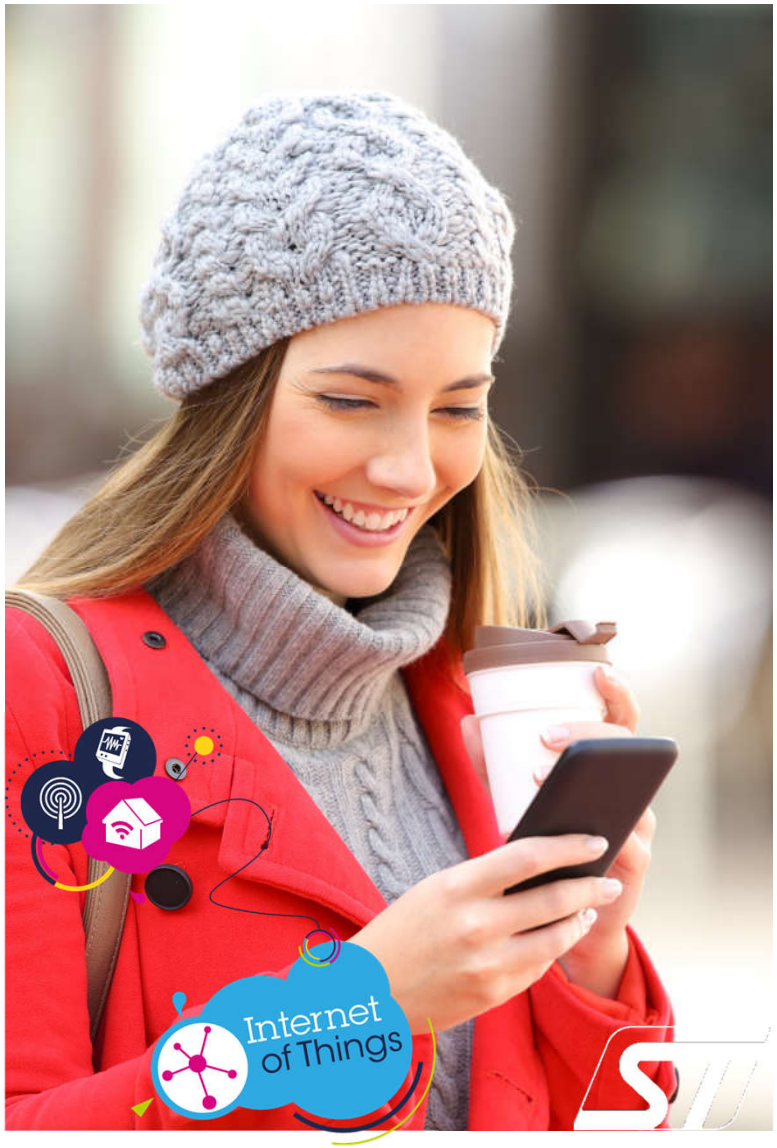
App 版本



life.augmented

BLE MESH Android App

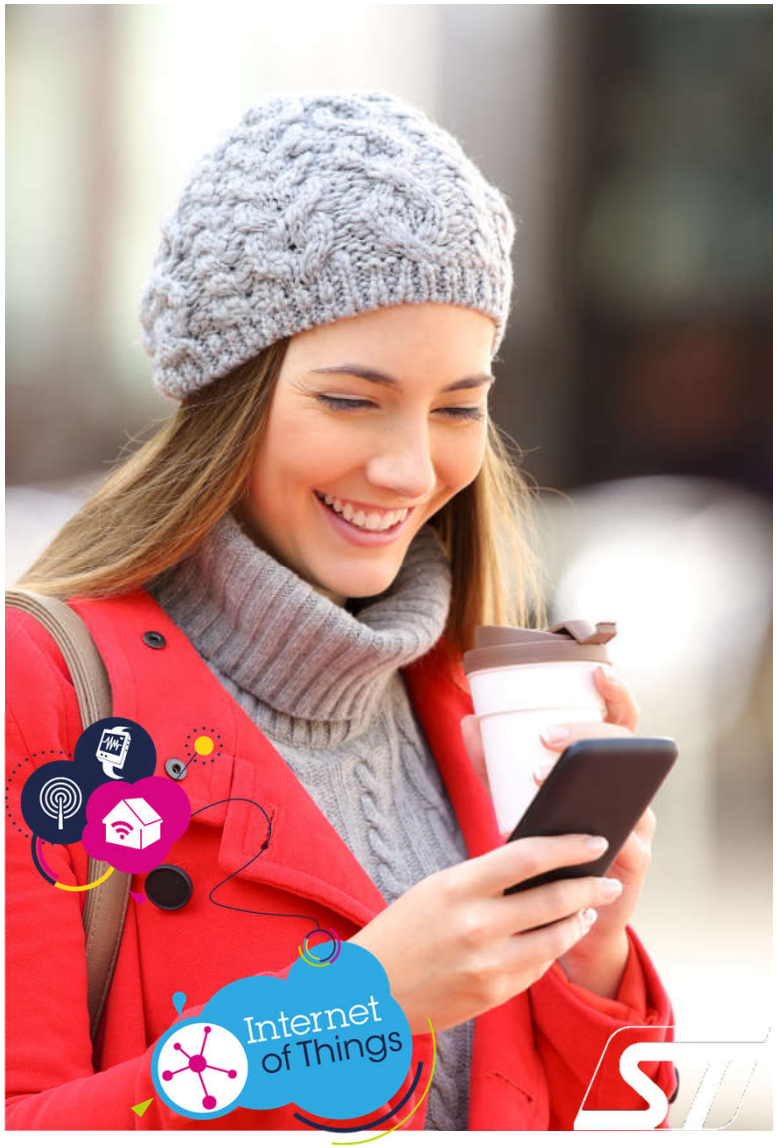
- Bluetooth Low Energy Mesh Introduction
- BLE Mesh over Android Framework
- App Walkthrough
- API Walkthrough and Hands on Session



life.augmented

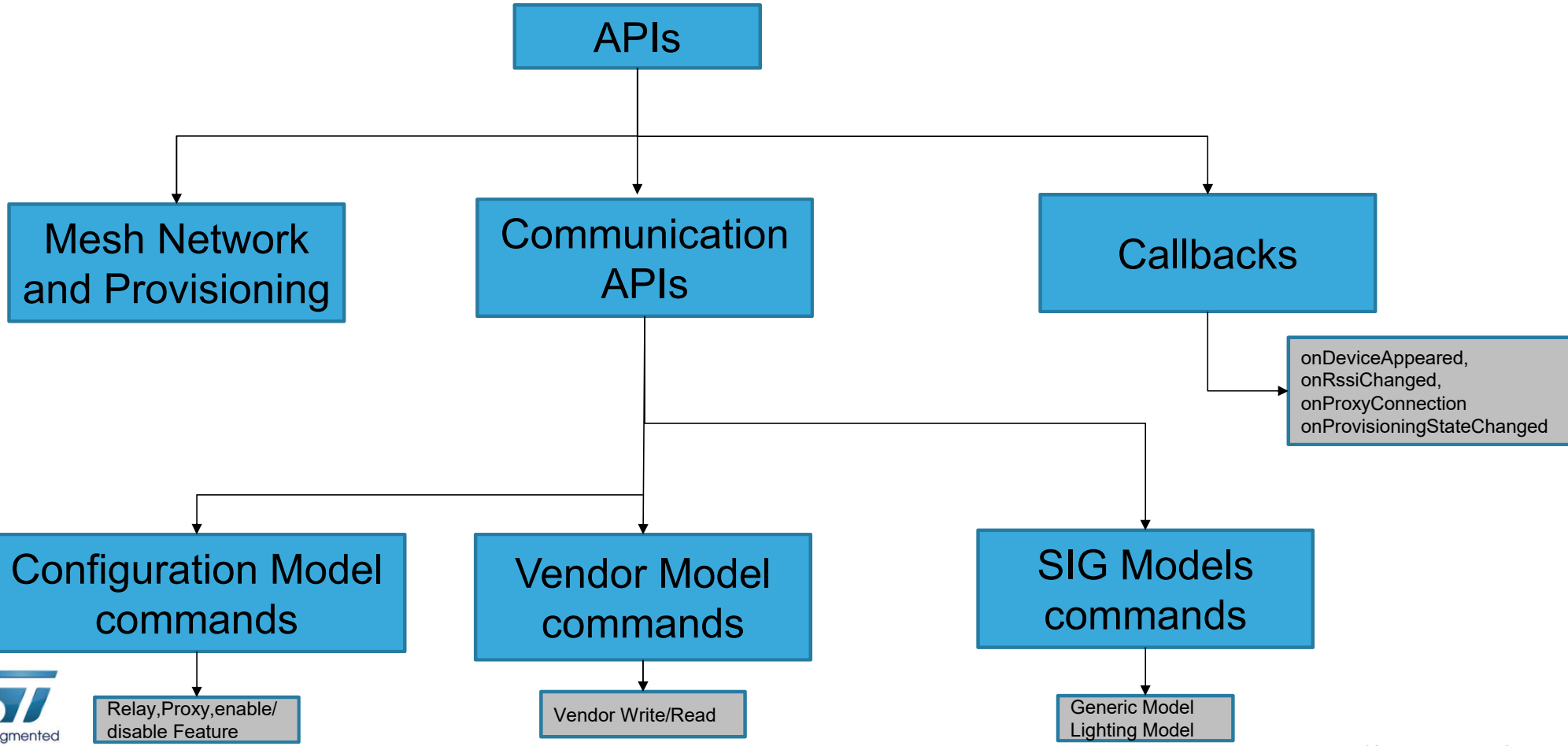
BLE MESH 安卓 App

- Bluetooth Low Energy Mesh 介绍
- 基于安卓框架的BLE Mesh
- App 流程
- API 流程以及实战练习任务



life.augmented

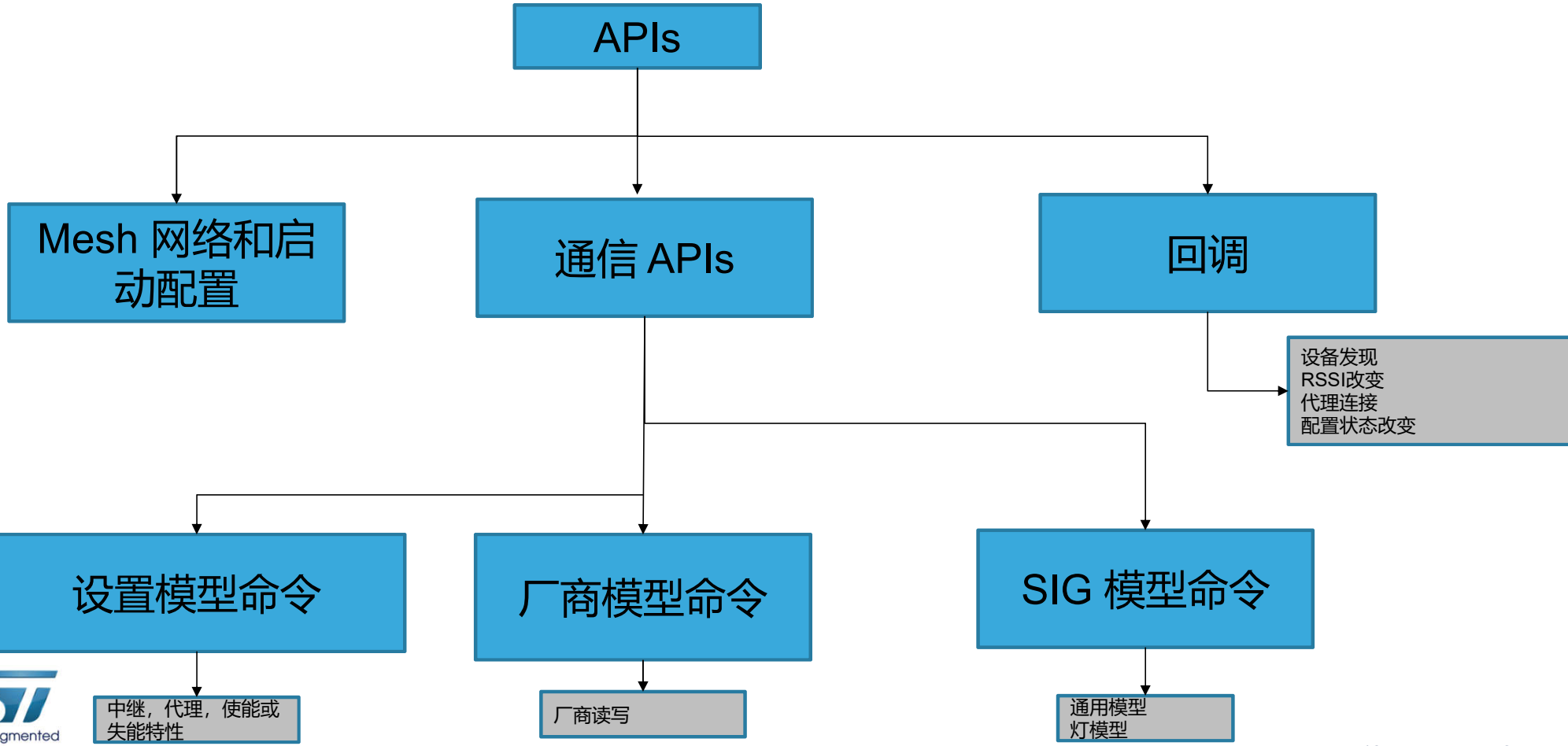
Android APIs Overview



life.augmented

*Specia Integrest Group

安卓APIs综述



life.augmented

*Specia Integrest Group

CreateNetwork(1/3)

public static mobleNetwork createNetwork(mobleAddress address) Method to create new network with given address* of Provisioner	
address	local address of Smartphone(address of the provisioner) 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);

life.augmented

*Address refferred in the slides is the moble(mesh) address

创建网络(1/3)

public static mobleNetwork createNetwork(mobleAddress address) 创建新网络的方式，使用由启动配置器给出的地址	
地址	智能手机的本地地址（启动配置器的地址） 不能和任何元素地址或者其他配置器地址相同
返回	参考 mobleNetwork object
用法	private mobleNetwork mNetwork; mNetwork = mobleNetwork.createNetwork(address);

life.augmented

*Address refferred in the slides is the moble(mesh) address

CreateNetwork(2/3)

public static mobleNetwork createNetwork(mobleAddress address,
java.lang.String netKey,
java.lang.String appKey)

Method to Create 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(address of the provisioner)
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 ”);

创建网络(2/3)

public static mobleNetwork createNetwork(mobleAddress address,
java.lang.String netKey,
java.lang.String appKey)

创建新网络的方式，使用启动配置器和网络密钥（字符串格式）、应用密钥（字符串格式）给出的地址（长度：16字节）
e.g. “1431ea1afeb05224ab892a0217ccab38”

address	智能手机的本地地址（启动配置器的地址）
netKey	网络密钥。对每个网络都必须是惟一的。大小：16字节
appKey	应用密钥。对每个网络都必须是唯一的。大小：16字节
returns	reference to the mobleNetwork object
用法	private mobleNetwork mNetwork; mNetwork = mobleNetwork.createNetwork(address, “1431ea1afeb05224ab892a0217ccab38”,” 6da9698c95f500e4edce3bb47f92754f ”);

CreateNetwork(3/3)

public static mobleNetwork createNetwork(mobleAddress address, byte[] netKey, byte[] appKey) Method to create 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(address of the provisioner)
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());

创建网络(3/3)

public static mobleNetwork createNetwork(mobleAddress address, byte[] netKey, byte[] appKey) 创建新网络的方法，使用由配置器，网络密钥（字节格式）和应用密钥（字节格式）给出的地址（长度:16字节） e.g. 1431ea1afeb05224ab892a0217ccab38..	
address	智能手机的本地地址（启动配置器的地址）
netKey	唯一的网络密钥
appKey	唯一的应用密钥
returns	reference to the mobleNetwork object
用法	private mobleNetwork mNetwork; mNetwork = mobleNetwork.createNetwork(address, new 1431ea1afeb05224ab892a0217ccab38”.toByteArray(),” 6da9698c95f500e4edce3bb47f92754f.toByteArray());

Start Network

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

启动网络

public mobleStatus start(android.content.Context context) 启动安卓ble mesh协议栈的方法	
Context	用户应用程序的对象
用法	private mobleNetwork mNetwork; mNetwork.start() ;
returns	操作的状态.
先决条件	这个BLE mesh网络必须准备被创建，使用createNetwork() API.

Stop Network

public mobleStatus stop(android.content.Context context) Method to Stop the functioning of Android BLE Mesh stack.	
Context	Object of User Application
usage	private mobleNetwork mNetwork; mNetwork.stop() ;
returns	status of operation.
Prerequisite	A BLE Mesh network is already created and started using createNetwork() and start() API respectively.

结束网络

public mobleStatus stop(android.content.Context context) 结束Android BLE Mesh 协议栈功能的方法.	
Context	用户应用程序的对象
用法	private mobleNetwork mNetwork; mNetwork.stop() ;
returns	操作的状态
先决条件	这个BLE Mesh网络必须准备好被创建和启动，分别使用createNetwork() 和 start() API

Restore Network


```
public static mobleNetwork restoreNetwork(mobleAddress address,
                                          String netKey ,
                                          String appKey,
                                          String meshdata
                                         ) throws IOException
```

Method to restore the previously formed Mesh Network using the Provisioner Address,Network Key,Application Key and Mesh JSON data.

Address	local address of Smartphone(address of the provisioner)
netKey	Unique Network Key
appKey	Unique Application Key
returns	reference to the mobleNetwork object
Meshdata	JSON* String for the Mesh Data

life.augmented

恢复网络


```
public static mobleNetwork restoreNetwork(mobleAddress address,
                                          String netKey ,
                                          String appKey,
                                          String meshdata
                                         ) throws IOException
```

恢复先前成型的mesh网络的方法，使用启动配置器地址，网络密钥，应用密钥和mesh JSON数据。

Address	智能手机的本地地址（启动配置器的地址）
netKey	唯一的网络密Unique Network Key
appKey	唯一的应用密钥
returns	reference to the mobleNetwork object
Meshdata	Mesh 数据的JSON字符串

life.augmented

Provision(1/4)


```
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)
```

Method to provision the device

Context	Object of User Application
address	MAC address of the Target Device(AA:BB:CC:DD:EE:FF)
mobleAddress	Target Address(Element Address - 1,2,3,4..) . Should not be equal to the Provisioner's address or any previously allocated Unicast Address.
indentifyduration	Duration in seconds upto which the Target device can identify itself(by blinking, etc) - e.g 10 secs.

启动配置(1/4)


```
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)
```

配置设备的方法

Context	用户应用程序的目标
address	目标设备的MAC地 (AA:BB:CC:DD:EE:FF)
mobleAddress	目标地址(元素地址 - 1,2,3,4..) .不能跟启动配置器或者任何先前分配的单播地址相同
indentifyduration	持续数秒，最大可以到目标设备自己能确认的时间（通过blinking，等等）- 例如 10 秒.

启动配置(2/4)

statusListener	Callback when the provisioning is completed.
capabilitiesLstnr	Callback when the Node's capabilities are received.
psc	Callback when provisioner state has been changed.Can be used to update the progress on the GUI.
completionDelay	Duration in miliseconds after which the configuration starts after re-connection to the proxy service
returns	Status of the provisioning operation
cpr	Custom provisioning instance. Currently kept null.(Not required)

启动配置(2/4)

statusListener	配置完成时回调
capabilitiesLstnr	节点功能被接收时回调
psc	配置器状态被改变后回调。可以在GUI上被用于更新进程。
completionDelay	回连到代理服务之后，配置启动之后，持续数毫秒
returns	启动配置操作的状态Status of the provisioning operation
cpr	用户启动配置实例。当前保持为null。（没有请求）

Provision(3/4)

Usage	private mobleSettings mSettings; mSettings = ((UserApplication) getApplication()).mConfiguration.getNetwork().getSettings(); mSettings.provision(MainActivity.this, address, mobleAddress 10, statusListener, capabilitiesLstnr, psc, completionDelay,//5000 miliseconds cpr);

启动配置 (3/4)

用法	private mobleSettings mSettings; mSettings = ((UserApplication) getApplication()).mConfiguration.getNetwork().getSettings(); mSettings.provision(MainActivity.this, address, mobleAddress 10, statusListener, capabilitiesLstnr, psc, completionDelay,//5000 miliseconds cpr);

Provision(4/4)

- Provisioning process if asynchronous
- Application shall implement interface mobleSettings.capabilitiesListener to confirm that proper Unprovisioned device is chosen for provisioning
  - 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 status

启动配置(4/4)

- 配置过程是否异步
- 应用应该实现接口 mobleSettings.capabilitiesListener 来确认合适的未被配置设备被选择后进行配置
  - 应用应该实现接口 mobleSettings.capabilitiesListener 来确认合适的未被配置设备被选择后进行配置
  - 应用可能实现接口 mobleSettings.provisionerStateChanged来跟踪配置过程的当前步骤
  - 应用可能实现接口 mobleSettings.onProvisionComplete 来跟踪配置过程完成状态：成功或失败
- mobleSettings.cancel() 停止配置过程
  - 如果接口mobleSettings.onProvisionComplete被实现，然后它会被回调并带有状态

Add Subscription/Group(1/2)


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

Method to add the subscription address for the Element.(Group creation)

Context	Object of User Application
address	Target Address(Element Address)
mElementaddress	Address of the Element
group	Address of the Group
Listener	Response of the command

添加订阅/组 (1/2)

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

添加元素订阅地址的方法。（组创建）

Context	用户应用程序的对象
address	目标地址（元素地址）
mElementaddress	元素地址
group	组地址
Listener	命令响应

Add Subscription/Group(2/2)

returns	status of operation.
Usage	app.mConfiguration.getNetwork().getSettings().addGroup (MainActivity.this, Nodeaddress, elementAddress, groupAddress mSubscriptionListener);

添加订阅/组 (2/2)

返回	操作的状态
用法	app.mConfiguration.getNetwork().getSettings().addGroup (MainActivity.this, Nodeaddress, elementAddress, groupAddress mSubscriptionListener);

Response for Subscription Command

Interface	ConfigurationModelClient.ConfigModelSubscriptionStatusCallback

Usage	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
	}
	};

订阅命令的响应

接口	ConfigurationModelClient.ConfigModelSubscriptionStatusCallback

用法	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
	}
	};

Set Publication(1/2)

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

Method to set the publication address for the Element

Context	Object of User Application
nodeAddress	Target Address(Element Address)
mElementaddress	Address of the Element
publishAddress	Address for the publication (Address can be of any provisioned node or any group)
callback	Response of the command
returns	status of operation

设置发布 (1/2)

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

设置元素发布地址的方法

Context	用于应用程序对象
nodeAddress	目标地址（元素地址）
mElementaddress	元素地址
publishAddress	发布地址（地址可以是任何配置过的节点或任何组）
callback	命令的响应
returns	操作的状态

Set Publication(2/2)

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

设置发布 (2/2)

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

Response for Publication Command

Interface	ConfigurationModelClient.ConfigModelPublicationStatusCallback

Usage	public final ConfigurationModelClient.ConfigModelPublicationStatusCallback mPublicationListener = new ConfigurationModelClient.ConfigModelPublicationStatusCallback() {
	@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
	}
	};

发布命令响应

接口	ConfigurationModelClient.ConfigModelPublicationStatusCallback

用法	public final ConfigurationModelClient.ConfigModelPublicationStatusCallback mPublicationListener = new ConfigurationModelClient.ConfigModelPublicationStatusCallback() {
	@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
	}
	};

Generic Model : GenericOnOff Set(1/2)


```
public boolean setGenericOnOff(boolean reliable,
                               ApplicationParameters.Address address,
                               ApplicationParameters.OnOff state,
                               ApplicationParameters.TID tid,
                               ApplicationParameters.Delay delay,
GenericOnOffModelClient.GenericOnOffStatusCallback callback)

Method to set the OnOff Value of the Lighting Element
```

reliable	Whether a response is required or Not
address	Target Address(Element Address)
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	Response of the command
returns	Status of the operation

通用模型：通用开关设置(1/2)


```
public boolean setGenericOnOff(boolean reliable,
                               ApplicationParameters.Address address,
                               ApplicationParameters.OnOff state,
                               ApplicationParameters.TID tid,
                               ApplicationParameters.Delay delay,
GenericOnOffModelClient.GenericOnOffStatusCallback callback)
```

设置灯元素的开关值的方法

reliable	是否需要响应
address	目标地址（元素地址）
state	开关状态—使能（1）或失能（0）
tid	交易身份
transitionTime	转变时间
delay	对目标会产生影响后的延时
callback	命令响应
returns	操作状态

Generic Model : GenericOnOff Set(2/2)

Usage	UserApplication app = (UserApplication) context.getApplicationContext(); ((UserApplication) ((MainActivity) context).getApplication()).mConfiguration.getNetwork().getOnOffModel().setGenericOnOff(true , Address state, tid, transitionTime, delay, Utils.mOnOffCallback);

通用模型：通用开关设置(2/2)

用法	UserApplication app = (UserApplication) context.getApplicationContext(); ((UserApplication) ((MainActivity) context).getApplication()).mConfiguration.getNetwork().getOnOffModel().setGenericOnOff(true , Address state, tid, transitionTime, delay, Utils.mOnOffCallback);

Generic Model : GenericOnOff Get

public boolean getGenericOnOff(ApplicationParameters.Address address, GenericOnOffModelClient.GenericOnOffStatusCallback callback) Method to get the OnOff Status of the Lighting Element	
Context	Object of User Application
address	Target Address(Element Address)
callback	Callback of the Publication
returns	Status of the operation

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

通用模型：通用开关设置

public boolean getGenericOnOff(ApplicationParameters.Address address, GenericOnOffModelClient.GenericOnOffStatusCallback callback) 获取灯元素开关状态的方法	
Context	用户应用程序的对象
address	目标地址（元素地址）
callback	发布的回调
returns	操作的状态

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

Response for GenericModel OnOff Command

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 {
	}
	}
	};

通用开关命令的响应

接口	public static interface GenericOnOffModelClient.GenericOnOffStatusCallback

用法	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 {
	}
	}
	};

Generic Model : Generic Level Set(1/2)


```
public boolean setGenericLevel(boolean reliable,
                               ApplicationParameters.Address address,
                               ApplicationParameters.Level level,
                               ApplicationParameters.TID tid,
                               ApplicationParameters.Delay delay,
                               GenericLevelStatusCallback callback)
```

Method to set the Level of the Lighting Element

reliable	Whether a response is required or Not
address	Target Address(Element Address)
level	Level Value
tid	Transaction Id
transitionTime	Transition Time
delay	delay Time
returns	Status of the operation

通用模型：通用等级设定 (1/2)


```
public boolean setGenericLevel(boolean reliable,
                               ApplicationParameters.Address address,
                               ApplicationParameters.Level level,
                               ApplicationParameters.TID tid,
                               ApplicationParameters.Delay delay,
                               GenericLevelStatusCallback callback)
```

设置等元素的等级的方法

reliable	是否需要响应
address	目标地址（元素地址）
level	等级值
tid	交易身份
transitionTime	转变时间
delay	延时时间
returns	操作状态

Generic Model : Generic Level Set(2/2)

Usage	mGenericLevelModel = ((UserApplication) ((MainActivity) context).getApplication()).mConfiguration.getNetwork().getLevelModel(); mGenericLevelModel.setGenericLevel(true,elementAddress, level,new ApplicationParameters.TID(2), null,//transition time null,//delay mLevelCallback);

通用模型：通用等级设定(2/2)

用法	mGenericLevelModel = ((UserApplication) ((MainActivity) context).getApplication()).mConfiguration.getNetwork().getLevelModel(); mGenericLevelModel.setGenericLevel(true,elementAddress, level,new ApplicationParameters.TID(2), null,//transition time null,//delay mLevelCallback);

Generic Model : Generic Level Get

public boolean getGenericLevel(ApplicationParameters.Address address, GenericLevelStatusCallback callback) Method to get the Level of the Lighting Element	
address	Target Address(Element Address)
callback	Response of the command
returns	status of operation.

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

通用模型：通用等级获取

public boolean getGenericLevel(ApplicationParameters.Address address, GenericLevelStatusCallback callback) 获取等元素等级的方法	
address	目标地址（元素地址）
callback	命令响应
returns	操作状态

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

Response for GenericModel Level Command

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();
	}
	
	}
	};

通用模型等级命令的响应

接口	public interface GenericLevelStatusCallback

用法	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();
	}
	
	}
	};

Lighting Model : LightLightness Set

public boolean setLightnessLevel(boolean reliable,
ApplicationParameters.Address address,
ApplicationParameters.Lightness lightness,
ApplicationParameters.TID tid, ApplicationParameters.Delay delay,
LightLightnessModelClient.LightingLightnessStatusCallback callback)

Used to set the Lightness of the Lighting Element

Reliable	Whether a response is required or Not
Address	Target Address(Element Address)
Lightness	Light Lightness value (0- 0xFFFF)
Tid	Transaction ID(unique transaction Id number)
Delay	Delay value
returns	Status of the operation
callback	Response of the command

灯模型：亮度设定

public boolean setLightnessLevel(boolean reliable,
ApplicationParameters.Address address,
ApplicationParameters.Lightness lightness,
ApplicationParameters.TID tid, ApplicationParameters.Delay delay,
LightLightnessModelClient.LightingLightnessStatusCallback callback)

用于设定等元素的亮度

Reliable	是否需要响应
Address	目标地址（元素地址）
Lightness	灯亮度值
Tid	交易身份（唯一交易ID号）
Delay	延时Delay value
returns	操作状态
callback	命令响应

Lighting Model : LightLightness Set

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

灯模型：亮度设定

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

Lighting Model : LightLightness Get

public boolean getLightnessLevel(ApplicationParameters.Address address, LightLightnessModelClient.LightingLightnessStatusCallback callback Used to get the current Lightness of the Lighting Element	
address	Target Address(Element Address)
callback	response of the command
returns	Succes or Failure
Usage	mLightingLightnessModel.getLightnessLevel(TEST_M_ADDRESS, mLightnessStatusCallback);

灯模型：亮度设定

public boolean getLightnessLevel(ApplicationParameters.Address address, LightLightnessModelClient.LightingLightnessStatusCallback callback 用于获取灯元素的当前亮度	
address	目标地址（元素地址）
callback	命令乡响应
returns	成功或失败
Usage	mLightingLightnessModel.getLightnessLevel(TEST_M_ADDRESS, mLightnessStatusCallback);

Response for LightingLightness Command

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 ");
	}
	}
	};

灯亮度命令的响应

public static interface	LightLightnessModelClient.LightingLightnessStatusCallback

用法	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 ");
	}
	}
	};

Lighting Model : Light CTL* Set(1/2)


```
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)
```

Used to set the Color,Temperature and the Lightness of the Lighting Element

Reliable	Whether a response is required or Not
Address	Target Address(Element Address)
Lightness	Light Lightness Value
Temperature	Temperature value
deltaUV	deltaUV value
Tid	Trasaction Id
Delay	Delay

life.augmented

*Color Temperature Lightness

灯模型：灯CTL*设定 (1/2)


```
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)
```

用于设定灯元素的颜色，温度和亮度

Reliable	是否需要响应
Address	目标地址（元素地址）
Lightness	灯亮度值
Temperature	温度值
deltaUV	deltaUV 值
Tid	交易身份
Delay	延时

life.augmented

*Color Temperature Lightness

Lighting Model : Light CTL Set(2/2)

public static interface	LightLightnessModelClient.LightingLightnessStatusCallback
Callback	Response of the command
Returns	Status of the operation

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

灯模型：灯CTL设定(2/2)

public static interface	LightLightnessModelClient.LightingLightnessStatusCallback
Callback	命令响应
Returns	操作状态
Usage	mLightCTLModelClient = network.getLightnessCTLModel(); mLightCTLModelClient.setLightCTL(true,TEST_M_ADDRESS,CTL_lightness,CTL_temperature,CTL_temperatureDeltaUV,CTL_tid,CTL_transitionTime,CTL_del, mLightCTLStatusCallback);

Lighting Model : Light CTL Get

public boolean getLightCTL(ApplicationParameters.Address address, LightCTLModelClient.LightCTLStatusCallback callback) Used to get the current Color,Temperature and Lightness of the Lighting Element	
address	Target Address(Element Address)
callback	Response of the command
returns	status of operation.

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

灯模型：灯CTL获取

public boolean getLightCTL(ApplicationParameters.Address address, LightCTLModelClient.LightCTLStatusCallback callback) 用于获取灯元素的当前颜色，温度和亮度	
address	目标地址（元素地址）
callback	命令响应
returns	操作状态

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

Response for LightingCTL Command

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 ");
	}
	}
	};

灯CTL命令的响应

public static interface	public static interface LightCTLModelClient.LightCTLStatusCallback

用法	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 ");
	}
	}
	};

Lighting Model : Light CTL Temperature Set

public boolean setLightCTLTemperature(boolean reliable, ApplicationParameters.Address address, ApplicationParameters.Temperature temp, ApplicationParameters.TemperatureDeltaUV deltaUV, ApplicationParameters.TID tid, ApplicationParameters.Delay delay, LightCTLModelClient.LightCTLTemperatureStatusCallback callback) Used to set the Temperature of the Lighting Element	
Reliable	Whether a response is required or Not
Address	Target Address(Element Address)
Temp	Temperature Value
deltaUV	Delta UV Value
Tid	Transaction ID
Delay	Delay
returns	Status of the operation
Callback	Response of the command

灯模型：灯CTL温度设定

public boolean setLightCTLTemperature(boolean reliable, ApplicationParameters.Address address, ApplicationParameters.Temperature temp, ApplicationParameters.TemperatureDeltaUV deltaUV, ApplicationParameters.TID tid, ApplicationParameters.Delay delay, LightCTLModelClient.LightCTLTemperatureStatusCallback callback)

用于设定灯元素的温度

Reliable	是否需要响应
Address	目标地址（元素地址）
Temp	温度值
deltaUV	Delta UV 值
Tid	交易身份
Delay	延时
returns	操作状态
Callback	命令响应

Lighting Model : Light CTL Temperature Set

Usage	mLightCTLModelClient = network.getLightnessCTLModel();
	
	mLightCTLModelClient.setLightCTLTemperature(true,TEST_M_ADDRESS,temp,tempDUV,tid,del,mLightCTLTemperatureStatusCallback);

灯模型：灯CTL稳定设定

用法	mLightCTLModelClient = network.getLightnessCTLModel();
	
	mLightCTLModelClient.setLightCTLTemperature(true,TEST_M_ADDRESS,temp,tempDUV,tid,del,mLightCTLTemperatureStatusCallback);

Lighting Model : Light CTL Temperature Get

public boolean getLightCTLTemperature(ApplicationParameters.Address address, LightCTLModelClient.LightCTLTemperatureStatusCallback callback) Used to get the current Temperature of the Lighting Element	
returns	status of operation.
address	Target Address(Element Address)
callback	Response of the command
returns	Status of the operation

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

灯模型：灯CTL温度获取

public boolean getLightCTLTemperature(ApplicationParameters.Address address, LightCTLModelClient.LightCTLTemperatureStatusCallback callback) 用于获取灯元素的当前温度值	
returns	操作状态
address	目标地址（元素地址）
callback	命令响应
returns	操作状态

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

Response for LightingCTL Temperature Command

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
	}
	}
	};

灯CTL温度命令的响应

public static interface	LightCTLModelClient.LightCTLTemperatureStatusCallback

用法	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
	}
	}
	};

Vendor Model : setRemoteData

java.lang.Object setRemoteData(mobleAddress peer,
int opcode,
int count,
byte[] data,
response)

Used to send(write) the LED On command to the Node using write Vendor Command

Peer	Target Address(Element Address)
Opcode	Opcode to be sent
Count	Count (redundant)
Data	Byte data to be sent
response	Response of the command required or not
returns	Status of the operation
Usage	network.getApplication().setRemoteData(addre, Nucleo.APPLI_CMD_LED_CONTROL, 1, new byte[]{Nucleo.APPLI_CMD_LED_ON}, ((MainActivity) context).rel_unrel);

供应商模型：设定远程数据

java.lang.Object setRemoteData(mobleAddress peer,
 int opcode,
int count,
byte[] data,
 response)

用于发送（写）LED开启命令到节点，使用供应商命令

Peer	目标地址（元素地址）
Opcode	待发送的操作码
Count	计数（冗余的）
Data	待发送的字节数据
response	命令是否需要被响应
returns	操作状态
Usage	network.getApplication().setRemoteData(addre, Nucleo.APPLI_CMD_LED_CONTROL, 1, new byte[]{Nucleo.APPLI_CMD_LED_ON}, ((MainActivity) context).rel_unrel);

Vendor Model : readRemoteData

java.lang.Object readRemoteData(mobleAddress peer, int opcode, int count, byte[] data, boolean response) Used to recieve(read) the Version of the firmware from the Node	
Peer	Target Address(Element Address)
opcode	Opcode to be sent
Count	Count (redundant) = 1 by default
Data	Byte data to be sent
response	Response of the command required or not
returns	Status of the operation
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);

供应商模型：读远程数据

java.lang.Object readRemoteData(mobleAddress peer, int opcode, int count, byte[] data, boolean response) 用于接收（读）节点的固件版本	
Peer	目标地址（元素地址）
opcode	待发送的操作码
Count	计数（冗余）=默认1
Data	待发送的字节数据
response	命令是否需要响应
returns	操作的状态
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);

Response for Vendor Model Command

void onResponse(mobleAddress peer, java.lang.Object cookies, byte status, byte[] data)	
peer	Target Address(Element Address)
cookies	Deprecated(Not used any currently)
Status	Status of the command
Data	Data sent in the response
returns	Status of the operation

Usage	((UserApplication) ((MainActivity) context).getApplication()).mConfiguration.getNetwork().advise(Utils.mLibraryVersionCallback);
	
	public static final defaultAppCallback mLibraryVersionCallback = new defaultAppCallback() {
	@Override
	public void onResponse(mobleAddress peer, Object cookies, byte status, byte[] data) {
	if (status == STATUS_SUCCESS) {
	//success
	} else {
	//failure
	}
	}
	};

厂商模型命令响应

void onResponse(mobleAddress peer, java.lang.Object cookies, byte status, byte[] data)	
peer	目标地址（元素地址）
cookies	弃用的（当前任何不用使用的）
Status	命令状态
Data	待响应的已发送数据
returns	操作状态

Usage	((UserApplication) ((MainActivity) context).getApplication()).mConfiguration.getNetwork().advise(Utils.mLibraryVersionCallback);
	
	public static final defaultAppCallback mLibraryVersionCallback = new defaultAppCallback() {
	@Override
	public void onResponse(mobleAddress peer, Object cookies, byte status, byte[] data) {
	if (status == STATUS_SUCCESS) {
	//success
	} else {
	//failure
	}
	}
	};

Register/Unregister Callbacks

void advise(mobleLayerApplication applicationCallback) void unadvise(mobleLayerApplication applicationCallback) methods to get the callbacks from the asynchronous events	
calback	Application Callbacks
Usage	Register : ((UserApplication) getApplication()).mConfiguration.getNetwork().advise(<application-Callbacks>); Unregister : ((UserApplication) getApplication()).mConfiguration.getNetwork().unadvise(<application-Callbacks>);

注册、注销回调函数

void advise(mobleLayerApplication applicationCallback) void unadvise(mobleLayerApplication applicationCallback) 从异步事件中获取回调	
calback	应用程序回调
Usage	Register : ((UserApplication) getApplication()).mConfiguration.getNetwork().advise(<application-Callbacks>); Unregister : ((UserApplication) getApplication()).mConfiguration.getNetwork().unadvise(<application-Callbacks>);

Callbacks : onDeviceAppeared

void onDeviceAppeared(java.lang.String bt_addr) Called if detected unconfigured MOBLE* device.	
Bt_addr	Bluetooth Address of the Device
prerequisite	Advise the callback : app.mConfiguration.getNetwork().advise(mOnDeviceAppearedCallback);

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

回调：在设备出现时

void onDeviceAppeared(java.lang.String bt_addr) 如果检测到为被配置的手机设备会调用	
Bt_addr	设备的蓝牙地址Bluetooth Address of the Device
prerequisite	建议回调: app.mConfiguration.getNetwork().advise(mOnDeviceAppearedCallback);

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

Callbacks : 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) {
	;
	}
	};

回调：当配置状态改变时

public static interface mobleSettings.provisionerStateChanged 如果启动配置器状态发生改变时调用.	
State	配置器当前状态
Label	状态标签

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

Callbacks : 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
	}
	};

回调：当配置完成时

public static interface mobleSettings.onProvisionComplete 如果配置完成时会调用	
状态	配置过程成功或失败

用法	// 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
	}
	};

Callbacks : onRssiChanged

void onDeviceRssiChanged(java.lang.String bt_addr, int mRssi) Called if device's RSSI value changed.	
Bt_addr	Bluetooth Address of the Device
mRSSI	RSSI value of the Device (dbm)
prerequisite	Advise the callback : app.mConfiguration.getNetwork().advise(onDeviceRssiChangedCallback);

Usage	public defaultAppCallback onDeviceRssiChangedCallback = new defaultAppCallback() {
	
	@Override
	public void onDeviceRssiChanged(final String bt_addr, final int mRssi) {
	super.onDeviceRssiChanged(bt_addr, mRssi);
	
	//User code
	}
	};

回调：当Rssi改变时

void onDeviceRssiChanged(java.lang.String bt_addr, int mRssi) 如果设备Rssi值改变时调用	
Bt_addr	设备的蓝牙地址
mRSSI	设备的RSSI值（dbm）
prerequisite	Advise the callback : app.mConfiguration.getNetwork().advise(onDeviceRssiChangedCallback);

Usage	public defaultAppCallback onDeviceRssiChangedCallback = new defaultAppCallback() {
	
	@Override
	public void onDeviceRssiChanged(final String bt_addr, final int mRssi) {
	super.onDeviceRssiChanged(bt_addr, mRssi);
	
	//User code
	}
	};

Callbacks : onProxyConnectionEvent

public void onProxyConnectionEvent(boolean process, String proxyAddress) Called when a proxy connection is made with the device	
Process	Proxy Connection Process
proxyAddress	Mac address of the proxy Device
prerequisite	Advise the callback : app.mConfiguration.getNetwork().advise(mProxyConnectionEventCallback);

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

回调：当代理产生连接事件时

public void onProxyConnectionEvent(boolean process, String proxyAddress) 当一个代理连接到设备时调用	
Process	Proxy Connection Process
proxyAddress	Mac address of the proxy Device
prerequisite	Advise the callback : app.mConfiguration.getNetwork().advise(mProxyConnectionEventCallback);

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

Config Model : getCompositionData

public boolean getDeviceCompositionData(
Address address,
Page page,
DeviceCompositionDataStatusCallback callback)

Used to Get the Device Composition Data using the Configuration Model command

address	Target Address(Element Address)
Page	Page 0 of the Configuration Page
callback	Response of the command

Usage	ConfigurationModelClient mConfigModel;
	mConfigModel = mobleNetwork.getConfigurationModelClient();
	
	mConfigModel.getDeviceCompositionData(new ApplicationParameters.Address(nodeAddress),ApplicationParameters.Page.PAGE0, deviceCompositionDataStatus_callback);

配置模型：获取组成数据时


```
public boolean getDeviceCompositionData(
        Address address,
        Page page,
        DeviceCompositionDataStatusCallback callback)
```

用户获取设备组成数据，使用配置模型命令

address	Target Address(Element Address)
Page	Page 0 of the Configuration Page
callback	Response of the command
Usage	ConfigurationModelClient mConfigModel; mConfigModel = mobleNetwork.getConfigurationModelClient(); mConfigModel.getDeviceCompositionData(new ApplicationParameters.Address(nodeAddress),ApplicationParameters.Page.PAGE0, deviceCompositionDataStatus_callback);

Get DeviceKey,AppKey,NetworkKey

mobleNetwork.getaDeviceKey() Get Device Key of the Current node being Provisioned mobleNetwork.getaNetworkKey() Get Network Key of the Current Network mobleNetwork. getaAppKey() Get Application Key of the Current Application	
Usage	mobleNetwork network = ((UserApplication) ((MainActivity) context).getApplication()).mConfiguration.getNetwork(); network.getaDeviceKey() network.getaNetworkKey() network. getaAppKey()

获取设备密钥，应用密钥，网络密钥

mobleNetwork.getaDeviceKey() 获取正在被配置的当前节点的设备密钥 mobleNetwork.getaNetworkKey() 获取当前网络的网络密钥 mobleNetwork. getaAppKey() 获取当前应用的应用密钥	
用法	mobleNetwork network = ((UserApplication) ((MainActivity) context).getApplication()).mConfiguration.getNetwork(); network.getaDeviceKey() network.getaNetworkKey() network. getaAppKey()

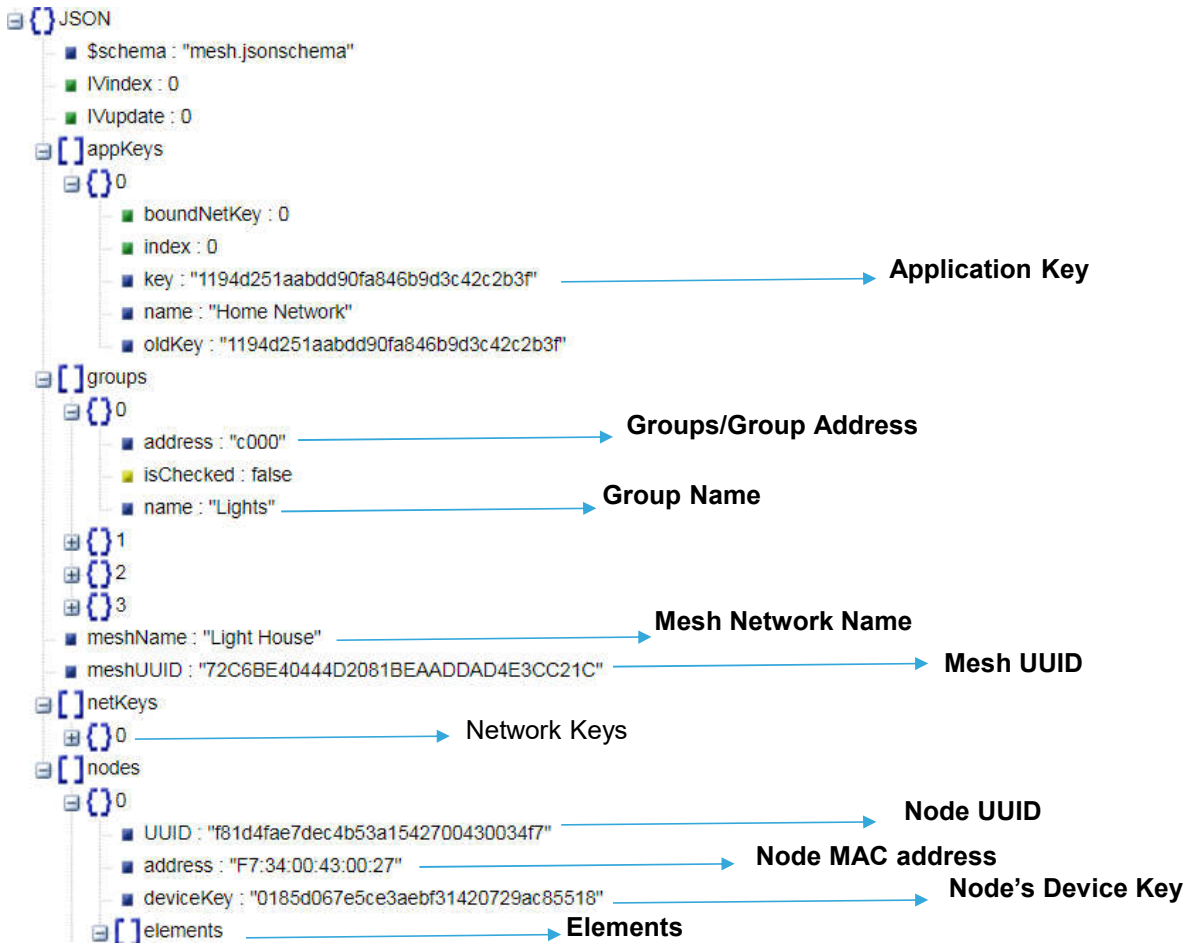
Mesh Network Configuration Database : JSON

- A book keeping way to store all the Network Configuration in a JSON file.
- This JSON is used to retrieve the network credentials and load the network in another Smartphone.
- This JSON can be shared via mail or sent to the cloud to sync the Network Credentials
- Copy paste the JSON content and View it in http://jsonviewer.stack.hu/

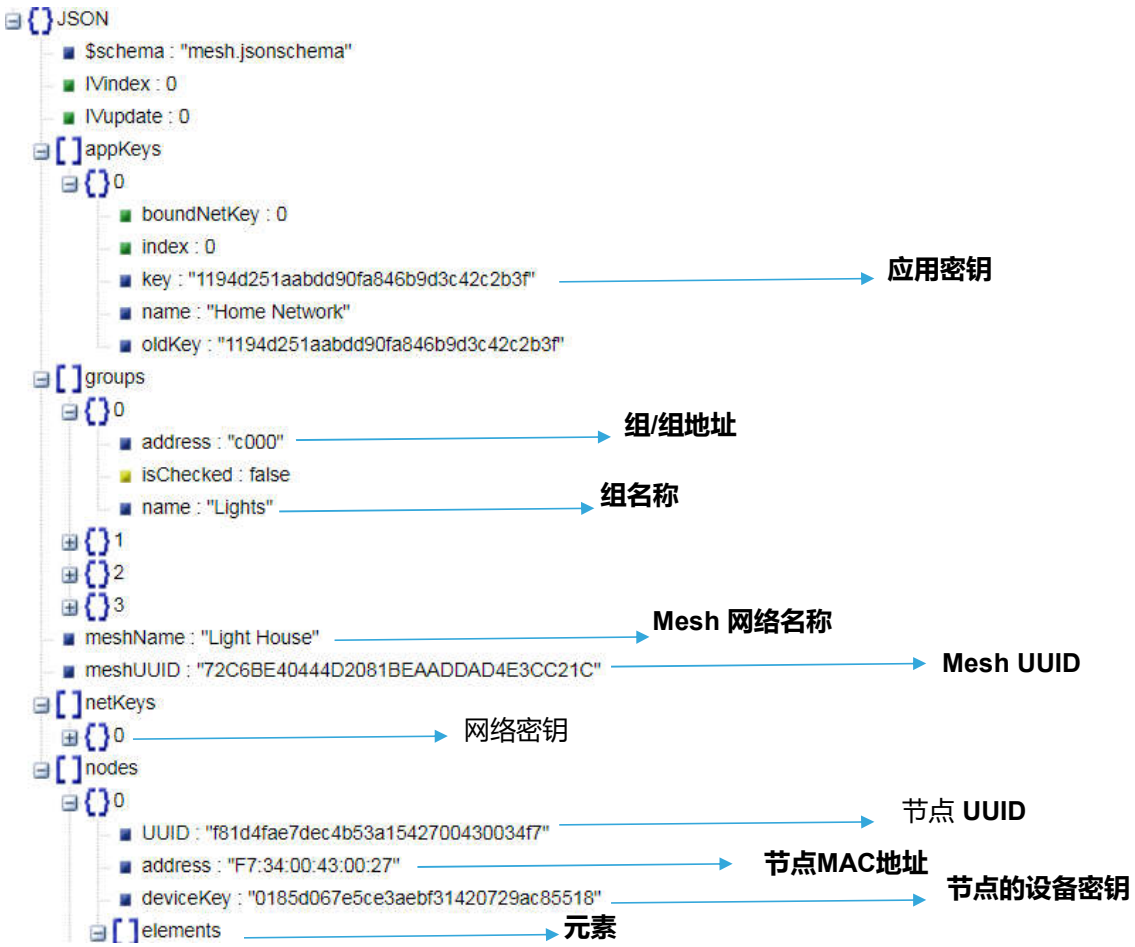
Mesh 网络配置数据库：JSON

- 在一个JSON文件中存储所有的网络配置
- 这个JSON被用于恢复网络证书并且在另外一台智能手机上导入网络
- 这个JSON能够通过邮件被分享或发送到云端去同步网络证书
- 复制粘贴JSON内容并且在http://jsonviewer.stack.hu/ 中查看它

JSON Data


JSON
$schema : "mesh.jsonschema"
IVindex : 0
IVupdate : 0
appKeys
0
boundNetKey : 0
index : 0
key : "1194d251aabdd90fa846b9d3c42c2b3f"
Application Key
name : "Home Network"
oldKey : "1194d251aabdd90fa846b9d3c42c2b3f"
groups
0
address : "c000"
Groups/Group Address
isChecked : false
name : "Lights"
Group Name
1
2
3
meshName : "Light House"
Mesh Network Name
meshUUID : "72C6BE40444D2081BEAADDAD4E3CC21C"
Mesh UUID
netKeys
0
Network Keys
nodes
0
UUID : "f81d4fae7dec4b53a15427004300034f7"
Node UUID
address : "F7:34:00:43:00:27"
Node MAC address
deviceKey : "0185d067e5ce3aebf31420729ac85518"
Node's Device Key
elements
Elements

JSON 数据



life.augmented

HANDS ON

life.augmented

实战练习

life.augmented

Exercise 1: Vendor Models

- Creating Vendor Model Command with data bytes
- Sending command
- Putting logs and checking commands and response

Get the instance of network

```
 mobleNetwork network = ((UserApplication) ((MainActivity)
context).getApplication()).mConfiguration.getNetwork();
```

Create and send command

```
network.getApplication().setRemoteData(mobleAddress.deviceA
ddress(addr), Nucleo.CUSTOM_COMMAND, 1, new
byte[]{Nucleo.CUSTOM_SUB_COMMAND , CUSTOM_DATA},
true);
```

Get the response

```
public static final defaultAppCallback mLibraryVersionCallback =
new defaultAppCallback() {

peer, Object cookies, byte status, byte[] data) {

    } };
```

练习 1: 厂商模型

- 创建厂商模型字节命令
- 发送命令
- 打印logs并检查命令和响应

Get the instance of network

```
 mobleNetwork network = ((UserApplication) ((MainActivity)
context).getApplication()).mConfiguration.getNetwork();
```

Create and send command

```
network.getApplication().setRemoteData(mobleAddress.deviceA
ddress(addr), Nucleo.CUSTOM_COMMAND, 1, new
byte[]{Nucleo.CUSTOM_SUB_COMMAND , CUSTOM_DATA},
true);
```

Get the response

```
public static final defaultAppCallback mLibraryVersionCallback =
new defaultAppCallback() {

peer, Object cookies, byte status, byte[] data) {

    } };
```

Exercise 2: Generic Models

- Creating GenericOnOff Model Command
- Sending command
- Putting logs and checking commands and response

Get the instance of model

```
 mobleNetwork network = ((UserApplication) ((MainActivity)
context).getApplication()).mConfiguration.getNetwork();

onOffModel = network.getOnOffModel();
```

Create and send command

```
onOffModel.setGenericOnOff(true,address,state,tid,time,delay,callback)
```

Getting the response

```
onOffModel.setGenericOnOff(true,address,state,tid,time,delay,callback)

public static final GenericOnOffModelClient.GenericOnOffStatusCallback
mOnOffCallback = new
GenericOnOffModelClient.GenericOnOffStatusCallback() {

public void onOnOffStatus(){ };
```

life.augmented

练习 2: 通用模型

- 创建通用开关模型命令
- 发送命令
- 打印logs并检查命令和响应

Get the instance of model

```
 mobleNetwork network = ((UserApplication) ((MainActivity)
context).getApplication()).mConfiguration.getNetwork();

onOffModel = network.getOnOffModel();
```

Create and send command

```
onOffModel.setGenericOnOff(true,address,state,tid,time,delay,callback)
```

Getting the response

```
onOffModel.setGenericOnOff(true,address,state,tid,time,delay,callback)

public static final GenericOnOffModelClient.GenericOnOffStatusCallback
mOnOffCallback = new
GenericOnOffModelClient.GenericOnOffStatusCallback() {

public void onOnOffStatus(){ };
```

Exercise 3: Device Subscription

- Sending a Device Subscription command
- Getting the response
- Checking the behavior of the command on the device

Get the instance of Configuration Model

```
ConfigurationModelClient mConfigModel;

mConfigModel = mobleNetwork.getConfigurationModelClient();
```

Create and send command

```
mConfigModel.addSubscription())
```

Getting the response

```
ConfigurationModelClient.ConfigModelSubscriptionStatusCallback
configModelSubscriptionStatus_callback = new
ConfigurationModelClient.ConfigModelSubscriptionStatusCallback()
```

练习 3: 设备订阅

- 发送设备订阅命令
- 获取响应
- 检查设备上命令的表现

Get the instance of Configuration Model

```
ConfigurationModelClient mConfigModel;

mConfigModel = mobleNetwork.getConfigurationModelClient();
```

Create and send command

```
mConfigModel.addSubscription())
```

Getting the response

```
ConfigurationModelClient.ConfigModelSubscriptionStatusCallback
configModelSubscriptionStatus_callback = new
ConfigurationModelClient.ConfigModelSubscriptionStatusCallback()
```

Exercise 4: Device Publication

- Sending a Set Publication command
- Getting the response
- Checking the behavior of the command on the device

Get the instance of Configuration Model

```
ConfigurationModelClient mConfigModel;

mConfigModel = mobleNetwork.getConfigurationModelClient();
```

Create and send command

```
mConfigModel.setPublication()
```

Getting the response

```
ConfigurationModelClient.ConfigModelPublicationStatusCallback
configModelPublicationStatus_callback = new
ConfigurationModelClient.ConfigModelPublicationStatusCallback()
```

life.augmented

练习 4: 设备发布

- 发送设置发布的命令
- 获取响应
- 检查设备上命令的表现

Get the instance of Configuration Model

```
ConfigurationModelClient mConfigModel;

mConfigModel = mobleNetwork.getConfigurationModelClient();
```

Create and send command

```
mConfigModel.setPublication()
```

Getting the response

```
ConfigurationModelClient.ConfigModelPublicationStatusCallback
configModelPublicationStatus_callback = new
ConfigurationModelClient.ConfigModelPublicationStatusCallback()
```

Questions and Answers

life.augmented

问答

life.augmented

References

- https://www.bluetooth.com/bluetooth-technology/topology-options/le-mesh/mesh-faq
- https://developer.android.com/studio/
- https://www.bluetooth.com/bluetooth-technology/topology-options
- https://www.bluetooth.com/develop-with-bluetooth/build/developer-kits/mesh-developer-study-guide
- https://www.bluetooth.com/bluetooth-technology/topology-options/le-mesh/mesh-glossary
- https://play.google.com/store/apps/details?id=com.macdom.ble.blescanner

life.augmented

参考

- https://www.bluetooth.com/bluetooth-technology/topology-options/le-mesh/mesh-faq
- https://developer.android.com/studio/
- https://www.bluetooth.com/bluetooth-technology/topology-options
- https://www.bluetooth.com/develop-with-bluetooth/build/developer-kits/mesh-developer-study-guide
- https://www.bluetooth.com/bluetooth-technology/topology-options/le-mesh/mesh-glossary
- https://play.google.com/store/apps/details?id=com.macdom.ble.blescanner

life.augmented

Thanks

life.augmented

谢谢

life.augmented