Introduction
The BlueNRG-Mesh iOS SDK from ST is based on the BLE Mesh Profile 1.0 connectivity specification from Bluetooth® SIG that extends network reach beyond the BLE point to point range.

The SDK allows iOS devices to communicate with BLE mesh nodes based on BlueNRG-1/2/MS platforms. It provides the functions to provision new devices to the mesh network, and mesh node control and configuration features.

The BlueNRG-Mesh iOS SDK includes iOS static library and demo mesh application sources. The API is implemented in Objective-C and can be called in both Objective-C and Swift applications.

A demo application in Swift 4 is provided for reference and all code snippets in this document are in Swift 4.

The requirements to use the SDK are listed below:
- Tools: XCode 9+
- Operating system: iOS 10.0
- Device: 64 bit capable (iPhone 5s onwards)

Figure 1. Low Energy mesh schematic
BlueNRG-Mesh SDK specifications

The Bluetooth® Mesh specifications are defined in two parts:
1. Bluetooth® Mesh Profile specifications for mesh nodes in general
2. Bluetooth® Mesh Model specifications for specific applications like lighting, sensors, etc.

1.1 BlueNRG-Mesh SDK modules

The following BlueNRG-Mesh SDK modules cover various aspects of Bluetooth® Mesh specifications.
1. **MeshManager**: the main module that controls basic mesh network functionality and provides factory methods for other modules.
2. **MeshProvisioner**: provides the functions and callbacks necessary for adding a node to a mesh network (provisioning).
3. **ConfigurationModel**: provides functions defined in Configuration client (Mesh Profile Specifications) to configure a mesh node (e.g., enabling and disabling various features, setting and updating keys, etc.)
4. **GenericModel**: provides functions defined in Generic client (Mesh Model Specifications) for generic control of the mesh nodes.
5. **VendorModel**: provides a vendor-defined (STMicroelectronics) interface to control the mesh nodes. The functionality of the standard Bluetooth® SIG model offers little scope for customization. The vendor model provides the flexibility to define interfaces for specific needs and applications.
6. **Health Model**: provides functions defined in the Health client (Mesh profile specification) to monitor the node health status.
7. **Heartbeat Model**: provides functions and callbacks to check if the nodes are still alive in the network. It also provides functionality to configure the heartbeat functionality.
2 How to set up and use the BlueNRG-Mesh SDK

Mesh stack functionality is accessed through the STMeshManager class.

**Step 1.** First, we need to create an instance of STMeshManager class as follows:

```
let meshManager = STMeshManager.getInstance(self)
```

*Note:* The ViewController must implement the STMeshManagerDelegate protocol to receive important events.

### 2.1 How to set up the mesh network

#### Initializing the library

**Step 1.** Initialize the library before you call any other function.

If there are multiple provisioners in a network, they must be initialized with unique unicast addresses.

```
/* The unicast address of provisioner is passed as parameter */
meshManager.createNetwork(1)
```

#### Set the network information

**Step 2.** The application should retrieve the Network configuration and provide it to the mesh library as soon as it starts.

The demo provides routines to store network data in JSON format. You can also use the JSON format to sync with a cloud application and other provisioners.

```
meshManager.setNetworkData(networkSettings)
```

*Note:* Any undocumented BlueNRG-Mesh iOS SDK headers you find that are not listed in this document represent future or untested functionality and are not recommended for use.

### 2.2 How to use the BlueNRG-Mesh library

The BlueNRG-Mesh library can function in Provisioning mode to add nodes to a mesh network and in Network mode to communicate with provisioned nodes in the network. Entering one mode automatically interrupts operation of the other.

#### 2.2.1 How to use the mesh library in Network mode (with Swift 4 code sample)

**Step 1.** Determine proxy connection method.

The BlueNRG-Mesh Library requires a mesh proxy node to connect to a mesh network. There are two ways to connect to a proxy node:

a. **Using NetworkID:** the BlueNRG-Mesh Library connects to any capable proxy node in the mesh network identified by an 8-octet key.
   
   ```
   proxyAddress = 0
   ```

b. **Using Node Identity:** the BlueNRG-Mesh Library connects to a specific node in the mesh network through a 16-octet key. This may be useful when you need to transmit a lot of data to a node directly over the GATT connection.

   ```
   proxyAddress = unicast address of a specific proxy node
   ```

   *Note:* For this connection method, the target node must be in direct radio range and advertise with Node Identity (refer to Mesh Profile 1.0 specifications).

**Step 2.** Start in Network mode.

This is the default operation mode used for normal mesh network operations. In this mode we can send and receive messages from mesh nodes.

```
meshManager.startNetwork(proxyAddress)
```
Step 3. Suspend mesh operation.  
This command stops the network.

```swift
meshManager.stopNetwork()
```

The following Swift 4 sample code shows how you can start and stop the mesh network in an application.

```swift
var currentNetworkDataManager = NetworkDataManager.sharedInstance()  
var manager: STMeshManager?

override func viewDidLoad() {
    super.viewDidLoad()
    manager = STMeshManager.getInstance(self)
    /* All the functions below are to be called only once.*/
    /* Initialize provisioner node with unicast address 1 */
    manager?.createNetwork(1)
    /* Read network settings from storage */
    currentNetworkDataManager.populateNodesFromStorage()
    //provide networkSetting/Provisioning data to library
    manager?.setNetworkData(currentNetworkDataManager.currentNetworkData)
}

override func viewWillAppear(_ animated: Bool) {
    /* For the cases when we come back from other screens where stopNetwork may have been called. */
    manager = STMeshManager.getInstance(self)
    manager?.startNetwork(0)
}

/* Sending vendor model commands to mesh node. */
func switchStateDidChange(index: IndexPath, switchState:Bool) -> Bool {
    let node = currentNetworkSetting.nodes[index.row] as! STMeshNode
    //Send toggle command
    let data = NSData(bytes: [0x03, 0x00] as [UInt8], length: 2)
    letvendorModel = manager?.getVendorModel()
    vendorModel?.setRemoteData(node.unicastAddress , usingOpCode:3,
        send: data as Data!, isResponseNeeded: false)
    return true
}

/* Use this call-back instead of using CBCentralManager for BLE status */
func meshManager(_ manager:STMeshManager!,  
didBTStateChange state:STMeshBleRadioState) {
    if(state != STMeshBleRadioState_PoweredOn) {
        //Display error / prompt user to switch on Bluetooth
    }
}
```

2.2.2 How to add new nodes in Provisioning Mode (with Swift 4 code sample)

**Step 1.** Search for unprovisioned devices and add them to the network using the provisioning procedure.  
Call the following method(s):

```swift
let provisioner = STMeshManager.getProvisioner()  
//Start scan for unprovisioned devices
//The timeout parameter is passed. Not used in current version 1.0.1.000
manager.startDeviceScan(0)
```

**Step 2.** Use the `didDeviceAppearedWithUUID:RSSI` callback when advertisement packets are received.  
Your application may update the RSSI values and visibility of the nodes in the user interface.  
Once the node to be provisioned is identified through its UUID, provisioning can start.

**Step 3.** Provide a new instance of `STMeshNode` with the UUID address.

**Step 4.** Use the `didProvisionStageChanged` callback during provisioning to provide status information regarding the process.

**Step 5.** Pause scanning for nodes.
This is useful to save power.

```swift
provisioner?.stopDeviceScan()
```

**Step 6.** Refer to the following Swift 4 sample code to build your own application.

The following Swift 4 sample code shows how you can provision nodes in an application.

```swift
var manager: STMeshManager?
override func viewWillAppear(_ animated: Bool) {
    manager = STMeshManager.getInstance(self)
    provisioner = manager.getProvisioner()
    /\ Switch to provisioning mode from proxy mode */
    /* Parameter : scan interval, not used in current SDK 1.01.000 */
    self.provisioner?.startDeviceScan(0);
}

/* invoked whenever unprovisioned node’s data packet is received
   UI can be updated to show nodes and also update RSSI value */
func provisioner(_ provisioner:STMeshProvisioner!,
    didDeviceAppearedWithUUID uuid: String, rssi: Int32) {
    /* Update or add new nodes in UI here. */
    addOrUpdateNode(uuid: uuid, rssi: rssi)
}

/* Start the provisioning process */
func startProvisioning(nodeToProvision: STMeshNode) -> Void {
    //Supply instance of STMeshNode, unicast addr, initial subscription addr
    manager?.provisionDevice(nodeToProvision, meshNode: 0, groupAddr: 0,
        identificationTime: 10)
}

/* invoked when node capability packet is received */
func provisioner(_ provisioner:STMeshProvisioner!,
    didReceiveCapabilitiesElementCount elementCount: UInt8) {
    /* Element count would be required for allocating addresses */
}

/* invoked when provisioning status changes */
func provisioner(_ provisioner:STMeshProvisioner!,
    didProvisionStageChanged percentage: Int32, updateMessage message: String!, hasError error: Bool) {
    /* Update UI to display provisioning progress to user */
}

override func viewWillDisappear(_ animated: Bool) {
    /* This may be called when leaving the provisioning screen */
    manager?.stopDeviceScan()
}
```
### Table 1. `enum STMeshStatus` options

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>STMeshStatus_Success = 0</td>
<td>Command completed successfully</td>
</tr>
<tr>
<td>STMeshStatus_False = 1</td>
<td>Operation did not occur but no error encountered</td>
</tr>
<tr>
<td>STMeshStatus_Fail = 2</td>
<td>General error</td>
</tr>
<tr>
<td>STMeshStatus_InvalidArg = 3</td>
<td>Invalid argument supplied</td>
</tr>
<tr>
<td>STMeshStatus_OutOfMemory = 4</td>
<td>Lack of memory</td>
</tr>
<tr>
<td>STMeshStatus_Timeout = 5</td>
<td>Operation timed out</td>
</tr>
<tr>
<td>STMeshStatus_NoConnection = 6</td>
<td>Operation failed as bearer not connected</td>
</tr>
<tr>
<td>STMeshStatus_NoBLE = 7</td>
<td>BLE radio is off or not present</td>
</tr>
</tbody>
</table>

### Table 2. `enum STMeshMode` options

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>STMeshMode_Provisioner = 0x00</td>
<td>Provisioner mode of the BLE mesh device or node</td>
</tr>
<tr>
<td>STMeshMode_ProxyClient = 0x01</td>
<td>Proxy mode of the BLE mesh device or node</td>
</tr>
<tr>
<td>STMeshMode_Offline = 0xFF</td>
<td>Offline mode of BLE mesh node. Not connected to any mesh device or node</td>
</tr>
</tbody>
</table>
4 BlueNRG-Mesh iOS library API: functions (Objective-C)

4.1 STMeshManager methods

4.1.1 Create Mesh Manager class object

(instancetype) getInstance

All mesh interaction is done through an instance of STMeshManager class. This function returns a single instance of STMeshManager.

4.1.2 Create network

(STMeshStatus) createNetwork:(uint16_t) address

Initializes the mesh network with given address.

Returns

STMeshStatus

Status of the command

Parameters

address

Uint16: Local address of node (usually 1 for smartphones)

4.1.3 Start and stop network processing

(STMeshStatus) startNetwork:(uint16_t) proxyAddress

Mesh network messages cannot be sent or received until this function is called.

Returns

STMeshStatus

Status of the command

Parameters

proxyAddress

Uint16: Address of preferred proxy node to connect to.

0 = the iOS device can connect to any proxy capable node in the network.

(STMeshStatus) stopNetwork

This method stops the network processing. It may be called to save power when user is not interacting with the app.

Returns

STMeshStatus

Status of the command.

4.1.4 Set network data

(STMeshStatus) setNetworkData:(STMeshNetworkSettings *)networkSettings;

This function is used to provide networkSettings to the BlueNRG-Mesh library. The Network settings contain the data required to communicate with the nodes in the network.

In the application, these settings are used to populate the list of provisioned nodes.

Returns

STMeshStatus

Status of the command.

Parameters

networkSettings

STMeshNetworkSettings: object containing the network information. To be managed by user application.

4.1.5 Get model instances

(STMeshConfigurationModel *) getConfigurationModel;

(STMeshGenericModel *) getGenericModel;

(STMeshVendorModel *) getVendorModel;
(STMeshLightingModel*) getLightingModel;
(STMeshHealthModel *)getHealthModel;
(STMeshHeartbeatModel *)getHeartbeatModel;

These functions are used to get instances of various models to access model specific commands and callbacks.

Returns
STMeshStatus
Status of the command.

4.1.6 Proxy connection

(BOOL) isConnectedToProxyService;

Checks whether the BlueNRG-Mesh Library is currently connected to any proxy nodes. This check is useful before sending commands to nodes to ensure there is a connection.

Returns
BOOL
Status of connection with proxy node.

4.1.7 Dummy mode operation

(void) setDummyMode:(BOOL *)dummyMode;

BlueNRG-Mesh iOS SDK includes Dummy mode functionality that allows app development even without access to actual BlueNRG mesh nodes and is also useful when fine tuning the UI on simulator.

When dummy mode is active, the actual Bluetooth radio is not used and the API simulates the behavior of actual devices. If you call provisioner startDeviceScan() in this mode, you receive callbacks from virtual devices that you can provision normally.

Note: The simulated behaviors are approximated and do not model a real mesh device perfectly. It is provided for convenience but you should not base your developments on them.

Parameters
dummyMode
BOOL: true to enable and false to disable dummy mode operation.

4.2 Mesh Manager callbacks

4.2.1 BT State is Changed

(void) meshManager:(STMeshManager *)manager didBTStateChange:(STMeshBleRadioState)status

This is called when local Bluetooth chip has changed state. It indicates to the application whether Bluetooth is supported and switched on.

Parameters
status
STMeshBleRadioState: returns status of the BLE radio.

4.3 Network mode and proxy callbacks

4.3.1 Proxy connection changes

(void) meshManager:(STMeshManager *)manager didProxyConnectionChanged :(BOOL)isConnected

This method is called when a connection to a proxy is made or when it terminates. You can update the UI with this information and perhaps disable commands when there is no connection to a proxy.

Parameters
isConnected
BOOL: true if connected, false if disconnected.

4.4 Provisioning methods
4.4.1 Scan for unprovisioned devices

(STMeshStatus) startDeviceScan:(uint32_t)timeOut

Initiates scan of unprovisioned devices. Callback method didDeviceAppearedWithUUID is called by the library when an unprovisioned device is found.

Returns
STMeshStatus
Status of the command.

Parameters
timeOut
Scanning stops after this period (in seconds) expires. This parameter is not used in the current version of the library 1.02.000.

Note: Network operations are stopped while this scan is running and the application must call meshManager.startNetwork after provisioning is completed to resume the network processing.

(STMeshStatus) stopDeviceScan

Stops the scanning of unprovisioned devices. This method can be used to save energy.

Returns
STMeshStatus
Status of the command.

4.4.2 Provision a device

(STMeshStatus) provisionDevice:(STMeshNode *)node deviceAddress:(uint16_t)addr identificationTime:(uint32_t)duration

This method starts provisioning the target device. Callback method didProvisionStageChanged is called to report the progress of the provisioning process. 100 % provisioning means that the process has completed successfully.

Returns
STMesh_Status
STMESH_SUCCESS: if the provisioning was started.
STMESH_FAIL: if the provisioning could not be started.

Note: The return value does not tell us if provisioning has completed successfully, it only tells us whether the process could be started or not. didProvisionStageChanged delegate, reports the status of provisioning.

Parameters
node
STMeshNode: new instance of STMeshNode with node UUID field filled in.

addr
uint16_t: the mesh unicast address to be assigned to the node being provisioned.

duration
uint32_t: the duration (in seconds) of the identify state of an unprovisioned device. In this state, the device signals for user attention in order to identify itself; e.g., with a blinking LED or beeping sound. This feature is not available in current release version 1.02.000.

4.5 Provisioning callbacks

4.5.1 Device provisioning updates

(void) meshManager:(STMeshManager *)manager didProvisionStageChanged :(int32_t)percentage updateMessage:(NSString *)message hasError:(BOOL)error

This method is called multiple times during the provisioning process to indicate the progress. Applications can reflect this information in UI (e.g., a progress bar) based on these callbacks.

Parameters
percentage
int32_t: provisioning process progress in percent.

message
NSString: text message to denote the progress of provisioning process. This field does not provide meaningful information in the current version of SDK (1.02.000)

error

BOOL: this flag is true if an error occurs during provisioning process.

### 4.5.2 New device found

`(void) meshManager:(STMeshManager *)manager didDeviceAppearedWithUUID:(NSString *)uuid RSSI: (int32_t) rssi;

This method is called when advertisement packets are received from an unprovisioned device after the meshManager.startDeviceScan is called. Applications can update the UI to display the new node. This callback may be called multiple times for the same node to update RSSI value, for example.

**Parameters**

- **uuid**
  - NSString: UUID of device
- **rssi**
  - int32_t: RSSI strength

### 4.5.3 Error

```objc
(meshManager:(STMeshManager *)manager didErrorOccurred:(NSString *)errMessage;
```

Called by the mesh library to indicate error condition.

**Parameters**

- **errorMessage**
  - NSString: error type

### 4.6 Configuration model methods

These methods are used by configuration client to update configuration of the mesh node.

#### 4.6.1 Unprovision a device

`-(STMeshStatus) resetConfigNode:(uint16_t) peerAddress;

This command instructs a mesh node to delete its network information and so become unprovisioned.

#### 4.6.2 Key management

`- (STMeshStatus) addConfigAppKeyOnNode:(uint16_t) peerAddress appKeyIndex:(uint16_t) appKeyIndex
 netKeyIndex:(uint16_t) netKeyIndex;

This method sets an application key on the target node. After provisioning, the app key must be configured before any application commands can be accepted by the node.

**Note:** In current version 1.02.000 non-zero netKeyIndex is not supported.

#### 4.6.3 Subscription and group management

`- (STMeshStatus) addConfigModelSubscriptionToNode:(uint16_t) peerAddress elementAddress:(uint16_t) elementAddress:
 (uint16_t)subscriptionAddress modelIdentifier:(uint32_t) modelIdentifier;

This method is used to subscribe a node to a group address.

`- (STMeshStatus) deleteConfigModelSubscriptionFromNode:(uint16_t) peerAddress elementAddress:(uint16_t) elementAddress:
 (uint16_t)elementAddress group:(uint16_t) groupAddress;

This method is used to remove group subscription from a node element.

`(STMeshStatus)ConfigSubscriptionDeleteAll:(uint16_t) peerAddress elementAddress:(uint16_t) elementAddress
 modelIdentifier:(uint32_t) modelIdentifier isVendor:(BOOL) isVendorModelId;

This method is used to remove all group subscription from a node element.

#### 4.6.4 Publication

`- (STMeshStatus) setConfigModelPublicationOnNode:(uint16_t) peerAddress elementAddress:
 (uint16_t)elementAddress publishAddress:(uint16_t) publishAddress appKeyIndex:(uint16_t) appKeyIndex
 credentialFlag:(BOOL) credentialFlag publishTTL:(uint8_t) publishTTL publishPeriod:
4.6.1 Composition

(STMeshStatus)getConfigCompositionData:(uint16_t)peerAddress pageNumber:(uint8_t)pageNumber;
This method is used to send requests for reading composition data.

4.6.6 Relay

(STMeshStatus)getConfigNodeRelay:(uint16_t)peerAddress;
This method is used to send requests for relay status data.

(STMeshStatus)getConfigNodeRelay:(uint16_t)peerAddress relay(BOOL)enableRelay retransmitCount:(uint8_t)count
retransmitInterval:(uint16_t)interval;
This method is used to send requests to set relay features.

4.6.7 GATTProxy

(STMeshStatus)getConfigNodeGATTProxy:(uint16_t)peeraddress;
This method is used to send requests for the GATTProxy feature status data.

(STMeshStatus)getConfigNodeGATTProxy:(uint16_t)peerAddress proxyState(BOOL)isEnableProxy;
This method is used to send requests to set relay features.

4.6.8 Node Friend

(STMeshStatus)getConfigNodeFriend:(uint16_t)peeraddress;
This method is used to send requests for the friend feature status data.

(STMeshStatus)getConfigNodeFriend:(uint16_t)peerAddress friendState(BOOL)isEnableFriend;
This method is used to send requests to set the node friend feature.

4.6.9 Node Identity

(STMeshStatus)getConfigNodeIdentity:(uint16_t)peerAddress netKeyIndex:(uint16_t)netKeyIndex
identity:(ConfigState)identity;
This method sends requests for enabling/disabling proxy advertisement using node identity.

4.6.10 Beacon

(STMeshStatus)getConfigBeacon:(uint16_t)peerAddr;
This method is used to send requests for Beacon status data.

(STMeshStatus)getConfigBeacon:(uint16_t)peerAddress isEnable:(BOOL)isEnabled;
This method is used to send requests for enabling/disabling secure network Beacon.

4.6.11 TTL

(STMeshStatus)getConfigTTL:(uint16_t)peerAddr;
This method is used to send requests for config TTL status data.

(STMeshStatus)getConfigTTL:(uint16_t)peerAddress ttlValue:(uint16_t)ttlValue;
This method is used to send requests for enabling/disabling secure network Beacon.

4.7 Configuration method callbacks

(void) meshConfigModel:(STMeshConfigurationModel*)configModel didReceiveSubscriptionStatus:
( ConfigModelStatus) ConfigurationStatus peerAddress:(uint16_t)peerAddress elementAddress:
(uint16_t)elementAddress modelIdentifier:(uint32_t)modelIdentifier;
This callback is invoked when subscription status is received from the element.

(void) meshConfigModel:(STMeshConfigurationModel*)configModel didReceivePublishStatus:
( ConfigModelStatus) ConfigurationStatus peerAddress:(uint16_t)peerAddress elementAddress:
(uint16_t)elementAddress publishAddress:(uint16_t)publishAddress appKeyIndex:(uint16_t)appKeyIndex
Arguments used by configuration model methods and callbacks

4.8 Arguments used by configuration model methods and callbacks

Parameters

peerAddress

uint16_t: address of node which is being configured

Subscription or group address

uint16_t: group address to be added subscription list of the node.
The current version supports up to 10 group addresses per node.

Publish address

uint16_t: address at which the node should publish its messages.

Element address

uint16_t: address of element which is being configured within the node.

App key index

uint16_t: global app key Index.
Default value is 0.

Net key index

uint16_t: global net key Index.
Default value is 0.

Model Identifier

uint32_t: Model ID of the model for which the command is being sent.

credentialFlag

BOOL: the publish friendship credential flag is a 1-bit state controlling the credentials used to publish messages from a model.
0: master security material is used for publishing
1: friendship security material is used for publishing
Publish TTL
uint8_t: TTL of the message published by the element of node.
Value from 0x00-0x7F
0xFF: the messages use the Default TTL.

Publish period
uint32_t: the Publish Period state determines the interval (in milliseconds) in which status messages are published by a model.

Retransmit count
uint8_t: specifies the number of times that a message published will be retransmitted after the initial send.

Retransmit interval
uint16_t: time in milliseconds between each retransmission.

4.9 Generic model methods
These methods are used by the generic client to send generic model commands to the mesh node. The generic model allows nodes to perform generic automation functions while maintaining the highest level of interoperability.

- (STMeshStatus) setGenericOnOff:(uint16_t)peerAddress isOn:(BOOL) isOn transitionTime:(uint8_t) time withDelay:(uint8_t) delay isUnacknowledged:(BOOL) responseFlag;
This method is used to send Generic On/Off commands to an element of a peer node with transition time.

- (STMeshStatus) readGenericOnOff:(uint16_t)peerAddress;
This method is used to read Generic On/Off status information from an element of a peer node.

- (STMeshStatus) setGenericLevel:(uint16_t)peerAddress level:(uint16_t) levelValue transitionTime:(uint8_t) time withDelay:(uint8_t) delay isUnacknowledged:(BOOL) responseFlag;
This method is used to send Generic level commands to an element of a peer node with transition time.

- (STMeshStatus) readGenericLevel:(uint16_t)peerAddress;
This method is used to read Generic level status information from an element of a peer node.

- (STMeshStatus) setGenericDelta:(uint16_t)peerAddress deltaLevel:(uint16_t) deltaValue transitionTime:(uint8_t) time withDelay:(uint8_t) delay isUnacknowledged:(BOOL) responseFlag;
This method is used to send Generic delta commands to an element of a peer node.

- (STMeshStatus) setGenericMove:(uint16_t)peerAddress deltaLevel:(uint16_t) deltaValue transitionTime:(uint8_t) time withDelay:(uint8_t) delay isUnacknowledged:(BOOL) responseFlag;
This method is used to send Generic move commands to an element of a peer node.

4.10 Generic model callbacks
These callbacks are called after receiving generic status messages.

- (void) genericModel:(STMeshGenericModel*)genericModel didReceiveOnOffStatusFromAddress:
  (uint16_t)peerAddress presentOnOff:(uint8_t) presentState targetOnOff:(uint16_t) targetState
  remainingTime:(uint16_t) time isTargetStatePresent:(uint16_t) stateFlag;
This callback is called when Generic On/Off status is received from the element of a peer node.

- (void) genericModel:(STMeshGenericModel*)genericModel didReceiveLevelStatusFromAddress:
  (uint16_t)peerAddress presentOnOff:(uint8_t) presentState targetOnOff:(uint16_t) targetState
  remainingTime:(uint16_t) time isTargetStatePresent:(uint16_t) stateFlag;
This callback is called when Generic On/Off status is received from the element of a peer node.

4.11 Arguments used by generic model methods and callbacks
Parameters
peerAddress
uint16_t: Address of element of the node involved.
isOn
BOOL: Requested On/Off state in generic On/Off command.
0x00: Off
0x01: On
transitionTime:time
uint8_t: Transition time determines how long an element shall take to transition from a present state to a new state. The parameter has a 6-bit field for the number of steps and a 2-bit field for the time for each step:
Number of steps (6-bit field):
0x00: Immediate transition
0x01-0x3E: number of steps
Time resolution per step (2-bit field):
0b00: Resolution 100ms
0b01: Resolution 1 seconds
0b10: Resolution 10 seconds
0b11: Resolution 10 minutes

withDelay: Delay
uint8_t: Message execution delay in 5 millisecond steps.

isUnacknowledged: responseFlag
BOOL: Determines whether the command is acknowledged or unacknowledged. Acknowledged requires the receiver to send a response message.
False: Acknowledged
True: Unacknowledged

presentState
uint8_t /uint16_t: Current On/Off or level state of the model.

targetState
uint8_t /uint16_t: Target level field identifies the target Generic Level state that the element should reach.

remainingTime
uint8_t: Remaining time field identifies the time it should take the element to complete the transition to the target Generic Level state of the element. The format of the field is same as transitionTime field.

isTargetStatePresent
BOOL: Target state and remainingTime fields are optional in Generic On/Off and Level status messages. This parameter indicates whether the optional fields are present.
False: Optional fields not present.
True: optional fields are present.

4.12 Vendor model methods
These methods are used to send and receive data from a mesh node.

4.12.1 Send command

(STMeshStatus) setRemoteData:(uint16_t)peerAddress dataMap:(uint32_t)opcode sendData:(NSData*)data isResponseNeeded:(BOOL)responseFlag

This function is used to send a command (with or without a payload) to a mesh network node with destination address. The user application is responsible for serializing data into the data buffer.

Returns
STMeshStatus
Status of the command

Parameters
peerAddress
uint16_t: destination address. May also be set to group address.

opcode
uint32_t: opcode of the command being sent. Only 4 LSB bits of opcode are used.

data
NSData: data buffer.
To get best results, payload length should be limited to 8 bytes.

responseFlag
BOOL: true if a response is requested for the command, false otherwise.

Note: Commands with response flag set are called reliable commands. If a reliable command is sent to a group address, an acknowledgement must be sent by each node in the group, which will cause extra traffic and may degrade network performance if there are many nodes in the group.

4.12.2 Read data

(STMeshStatus) readRemoteData:(uint16_t)peerAddress dataMap:(uint32_t)opcode
Requests data from the given destination address.

Returns
STMeshStatus
Status of the command.

Parameters
peerAddress
uint16_t: destination address. It can be a group address also. Use of group addresses in read commands is not advised.

opcode
uint32_t: opcode of the read command being sent. Only 4 LSB bits of opcode are used.

4.12.3 Read version data

(STMeshStatus) readDeviceVersionData:(uint16_t)peerAddress usingOpcode:(uint8_t)opcodesendData:(NSData*)data

Returns STMeshStatus, status of the command and parameters.

4.13 Vendor model callbacks

These callbacks are called after receiving vendor model responses.

-(void) meshManager:(STMeshManager *)manager gotResponseFrom:(uint16_t)peerAddress
commandStatus:(STMeshCmdStatus)status recvData:(NSData*)data

This method is called when response of command sent earlier is received.

Parameters
peerAddress
uint16_t: address of device that has sent the response.

status
STMeshCommandStatus: status of the command.

data
NSData: data buffer containing the response payload.

4.14 Lighting model method

4.14.1 Lighting lightness

(STMeshStatus)readLightingLightnessStatus:(uint16_t)peerAddress;
This method is used to read Lighting lightness status from an element of a peer node.

(STMeshStatus)setLightingLightness:(uint16_t)peerAddress lightnessValue:(uint16_t)lightness
isUnacknowledged:(BOOL)responseFlag;
This method is used to send Lighting lightness command to element of a peer node.

(STMeshStatus)setLightingLightness:(uint16_t)peerAddress lightnessValue:(uint16_t)lightness
transitionTime:(uint8_t)time withDelay:(uint8_t)delay isUnacknowledged:(BOOL)responseFlag;
This method is used to send Lighting lightness command with optional parameters to an element of a peer node.

(STMeshStatus)readLightingLightnessLinearStatus:(uint16_t)peerAddress;
This method is used to read Lighting lightness Linear status from an element of a peer node.

(STMeshStatus)setLightingLightnessLinear:(uint16_t)peerAddress lightnessValue:(uint16_t)lightness
isUnacknowledged:(BOOL)responseFlag;
This method is used to send Lighting lightness Linear command to an element of a peer node.

(STMeshStatus)setLightingLightnessLinear:(uint16_t)peerAddress lightnessValue:(uint16_t)lightness
transitionTime:(uint8_t)time withDelay:(uint8_t)delay isUnacknowledged:(BOOL)responseFlag;
This method is used to send Lighting lightness Linear command with optional parameters to an element of a peer node.

(STMeshStatus)readLightingLightnessLastStatus:(uint16_t)peerAddress;
This method is used to read Lighting lightness Last status from an element of a peer node.

(STMeshStatus)readLightingDefaultStatus:(uint16_t)peerAddress;
This method is used to read Lighting lightness default status from an element of a peer node.
(STMeshStatus)setLightingLightnessDefault:(uint16_t)peerAddress lightnessValue:(uint16_t)lightness isUnacknowledged:(BOOL)responseFlag;
This method is used to send lightness default command to an element of a peer node.

(STMeshStatus)readLightingRangeStatus:(uint16_t)peerAddress;
This method is used to read Lighting lightness range status from an element of a peer node.

(STMeshStatus)setLightingLightnessRange:(uint16_t)peerAddress rangeMinValue:(uint16_t)rangeMin rangeMaxValue:(uint16_t)rangeMax isUnacknowledged:(BOOL)responseFlag;
This method is used to send lightness range command to an element of a peer node.

4.14.2 Lighting CTL

(STMeshStatus)readLightCTLStatus:(uint16_t)peerAddress;
This method is used to read Lighting CTL status from an element of a peer node.

(STMeshStatus)setLightingCTL:(uint16_t)peerAddress lightnessValue:(uint16_t)lightness temperatureValue:(uint16_t)temperature deltaUVValue:(int16_t)deltaUV isUnacknowledged:(BOOL)responseFlag;
This method is used to send Lighting CTL command from an element of a peer node.

(STMeshStatus)setLightingCTL:(uint16_t)peerAddress lightnessValue:(uint16_t)lightness temperatureValue:(uint16_t)temperature deltaUVValue:(int16_t)deltaUV transatimnTime:(uint16_t)time withDelay:(uint8_t)delay isUnacknowledged:(BOOL)responseFlag;
This method is used to send Lighting CTL command with optional parameters from an element of a peer node.

(STMeshStatus)readLightCTLDefaultStatus:(uint16_t)peerAddress;
This method is used to read Lighting CTL default status from an element of a peer node.

(STMeshStatus)setLightingCTLDefault:(uint16_t)peerAddress lightnessValue:(uint16_t)lightness temperatureValue:(uint16_t)temperature deltaUVValue:(int16_t)deltaUV isUnacknowledged:(BOOL)responseFlag;
This method is used to send Lighting CTL default command from an element of a peer node.

(STMeshStatus)readLightCTLTemperatureStatus:(uint16_t)peerAddress;
This method is used to read Lighting CTL Temperature status from an element of a peer node.

(STMeshStatus)setLightingCTLTemperature:(uint16_t)peerAddress temperatureValue:(uint16_t)temperature deltaUVValue:(int16_t)deltaUV isUnacknowledged:(BOOL)responseFlag;
This method is used to send Lighting CTL Temperature command from an element of a peer node.

(STMeshStatus)setLightingCTLTemperature:(uint16_t)peerAddress temperatureValue:(uint16_t)temperature deltaUVValue:(int16_t)deltaUV transatimnTime:(uint16_t)time withDelay:(uint8_t)delay isUnacknowledged:(BOOL)responseFlag;
This method is used to send Lighting CTL temperature command optional parameters from an element of a peer node.

(STMeshStatus)readLightCTLDefaultStatus:(uint16_t)peerAddress;
This method is used to read Lighting CTL default status from an element of a peer node.

4.15 Lighting model method callback

(void) lightingModel:(STMeshLightingModel*)lightingModel didReceiveLightnessStatusFromAddress:(uint16_t)peerAddress presentLightness:(uint16_t)presentLightness targetLightness:(uint16_t)targetLightness remainingTime:(uint8_t)time isTargetStatePresent:(BOOL)stateFlag;
This callback is called when lighting lightness status is received from the element of a peer node.

(void) lightingModel:(STMeshLightingModel*)lightingModel didReceiveLightnessLinearStatusFromAddress:(uint16_t)peerAddress presentLightness:(uint16_t)presentLightness targetLightness:(uint16_t)targetLightness remainingTime:(uint8_t)time isTargetStatePresent:(BOOL)stateFlag;
This callback is called when lighting lightness linear status is received from the element of a peer node.

(void) lightingModel:(STMeshLightingModel*)lightingModel didReceiveLightnessLastStatusFromAddress:(uint16_t)peerAddress lightnessValue:(uint16_t)lightness;
This callback is called when lighting lightness last status is received from the element of a peer node.
4.16 Arguments used by configuration model methods and callbacks

**peerAddress**
```
uint16_t: Address of element of the node involved
```

**lightness**
```
uint16_t: lightness of light
```

**transitionTime: Time**
```
uint8_t: Transition time determines how long an element shall take to transition from a present state to a new state. The parameter has a 6-bit field for the number of steps and a 2-bit field for the time for each step.
```

- Number of steps (6-bit field):
  - 0x00: Immediate transition
  - 0x01-0x3E: number of steps

- Time resolution per step (2-bit field):
  - 0b00: Resolution 100ms
  - 0b01: Resolution 1 second
  - 0b10: Resolution 10 seconds
  - 0b11: Resolution 10 minutes

**withDelay: Delay**
```
uint8_t: Message execution delay in 5 millisecond steps.
```

**isUnacknowledged: responseFlag**
```
BOOL: Determines whether the command is acknowledged or unacknowledged. Acknowledged requires the receiver to send a response message:
```
- False = Acknowledged
- True = Unacknowledged

**TemperatureValue: temperature**
```
Uint16_t: color temperature of tunable white light emitted by an element.
```

**DeltaUVValue: deltaUV**
```
Uint16_t: deltaUV value of from CIE 1976 curve, it determines the distance from the black body curve.
4.17 Health model methods

4.17.1 Get Faults

(STMeshStatus)healthModelGetFaults:(uint16_t)peerAddr companyIdentifier:(uint16_t)cid;
This method is used to read all the present faults related to the peer node.

4.17.2 Clear Faults

(STMeshStatus)HealthFaultClear:(uint16_t) peerAddr companyIdentifier:(uint16_t) cid
isUnacknowledgedCommand:(bool) isUnacknowledged;
This method is used to clear all the faults from the fault register without waiting for an acknowledgement message.

(STMeshStatus)HealthFaultClear:(uint16_t) peerAddr companyIdentifier:(uint16_t) cid;
This method is used to clear all the faults from the fault register (by default Acknowledge message).

4.17.3 Get Health Period

(STMeshStatus)HealthPeriodGet:(uint16_t) peerAddr;
This method is used to get the health period from a respective peer node.

4.17.4 Set Health Period

(STMeshStatus)HealthPeriodSet:(uint16_t) peerAddr fastPeriodDivisor:(uint8_t)fastPeriodDivisor
isUnacknowledgedCommand:(bool) isUnacknowledged;
This method is used to set the health period by passing the fastPeriodDivisor argument without waiting for an acknowledgement message.

(STMeshStatus)HealthPeriodSet:(uint16_t) peerAddr fastPeriodDivisor:(uint8_t)fastPeriodDivisor;
This method is used to set the health period by passing the fastPeriodDivisor argument (by default Acknowledge message).

Note: 
FastPeriodDivisor splits the current health period value.
The response to Health Period get/set message is Health Period status message.

4.17.5 Get Health Attention

(STMeshStatus)HealthAttentionGet:(uint16_t) peerAddr;
This method is used to get health attention from a respective peer node.

4.17.6 Set Health Attention

(STMeshStatus)HealthAttentionSet:(uint16_t) peerAddr attention:(uint8_t)attention
isUnacknowledgedCommand:(bool) isUnacknowledged;
This method is used to set the health attention by passing the attention argument (without waiting for the acknowledgement message).

(STMeshStatus)HealthAttentionSet:(uint16_t) peerAddr attention:(uint8_t)attention;
This method is used to set the health attention by passing the attention argument (by default Acknowledge message).

Note: 
The response to Health Attention get/set messages is Health Attention status message.

@property(weak, nonatomic) id<STMeshHealthModelDelegate> delegate;
This delegate must be confirmed to get the callbacks of the methods.

4.17.7 Health Status

(void) healthModel:(STMeshHealthModel*)healthModel didReceiveHealthStatusFromAddress:
(uint16_t)peerAddr withPresentsFaults:(NSArray*)faultsArray;
This callback is called when register fault status is received from the peer node. All the present fault will be present in faultsArray.

4.17.8 Health Period Status

(void) healthModel:(STMeshHealthModel *)healthModel
didReceiveHealthPeriodStatusResponseFromAddress:(uint16_t)peerAddr WithFastPeriodDivisor:
(uint8_t)fastPeriodDivisor;
This callback is called when the Health Period Status is received from the peer node.
4.17.9 Health Attention Status

(void) healthModel:(STMeshHealthModel *)healthModel
didReceiveHealthAttentionStatusResponseFromAddress:(uint16_t)peerAddr WithAttention:
(uint8_t)attention;
This callback is called when Health Attention Status is received from the peer node.

4.18 Heartbeat model methods

4.18.1 Get Heartbeat Publication

(STMeshStatus)heartbeatPublicationGet:(uint16_t)peerAddress;
This method is used to get the Heartbeat Publication from a peer node.

4.18.2 Set Heartbeat Publication

(STMeshStatus)heartbeatPublicationSet:(uint16_t) peerAddr destinationAddress:(uint16_t)
destinationAddress countLog:(uint8_t)countLog periodLog:(uint8_t)periodLog TTL:(uint8_t)TTL features:
(uint16_t)features netKeyIndex:(uint16_t)netKeyIndex;
This method is used to set the Heartbeat Publication from a peer node.

Note: The response to Heartbeat Publication get/set message is Heartbeat Publication status message.

4.18.3 Get Heartbeat Subscription

(STMeshStatus)heartbeatSubscriptionGet:(uint16_t) peerAddr;
This method is used to get the Heartbeat Subscription from a peer node.

4.18.4 Set Heartbeat Subscription

(STMeshStatus)heartbeatSubscriptionSet:(uint16_t) peerAddr sourceAddress:(uint16_t)sourcAddress
destinationAddress:(uint16_t)destinationAddress periodLog:(uint8_t) periodLog;
This method is used to set the Heartbeat Subscription from a peer node.

Note: The response to the Heartbeat Subscription get/set message is Heartbeat Subscription status message.

@property(weak , nonatomic) id<STMeshHeartBeatModelDelegate> delegate;
This delegate must be confirmed to get the callbacks of the methods.

4.18.5 Heartbeat Publication Status

(void) heartbeatModel:(STMeshHeartbeatModel *)heartbeatModel
didRecieveHeartbeatPublicationsStatusFromAddress:(uint16_t)peerAddress statusCode:(uint8_t)status
destinationAddress:(uint16_t)destinationAddress countLog:(uint8_t)countLog periodLog:
(uint8_t)periodLog TTL:(uint8_t)TTL features:(uint16_t)features netKeyIndex:(uint16_t)netKeyIndex;
This callback is called when Heartbeat Publication Status is received from a peer node.

4.18.6 Heartbeat Subscription Status

(void) heartbeatModel:(STMeshHeartbeatModel *)heartbeatModel
didRecieveHeartbeatSubscriptionStatusFromAddress:(uint16_t)peerAddress statusCode:(uint8_t)status
destinationAddress:(uint16_t)destinationAddress periodLog:(uint8_t)periodLog countLog:
(uint8_t)countLog minHops:(uint8_t)minHops maxHops:(uint8_t)maxHops
This callback is called when Heartbeat Subscription Status is received from a peer node.
NetworkSettings object contains the provisioning data required by provisioner to manage and control the nodes on the network. It can also be used by user application to display a list of nodes in UI.

Types used in the API are defined as below.

```objc
@interface STMeshGroup : NSObject
@property (nonatomic) NSString *groupName;
@property (nonatomic) uint16_t groupAddress;
@property (nonatomic) BOOL switchState;
@property (nonatomic) NSMutableArray<STMeshElement*> *subscribersElem;
@end

@interface STMeshNode : NSObject
@property (nonatomic) NSString *nodeName;
@property (nonatomic) NSString *nodeUUID;
@property (nonatomic) NSString *deviceKey;
@property (nonatomic) BOOL isProxyNode;
@property (nonatomic) BOOL switchState;
@property (nonatomic) uint16_t publishAddress;
@property (nonatomic) uint16_t unicastAddress;
@property (nonatomic, strong) id publishTarget;
@property (nonatomic) BOOL configComplete;
@property (nonatomic) BOOL blacklisted;
@property (nonatomic) NSString *cid;
@property (nonatomic) NSString *pid;
@property (nonatomic) NSString *nodeUUID;
@property (nonatomic) NSString *deviceId;
@property (nonatomic) STMeshNodeFeatures *features;
@property (nonatomic) NSMutableArray<STMeshElement*> *elementList;
@property (nonatomic) NSMutableArray<STMeshElement*> *subscribedGroups;
@property (nonatomic) STMeshNode *parentNode;
@end

@interface STMeshElement : NSObject
@property (nonatomic) uint8_t index;
@property (nonatomic) NSString *elementName;
@property (nonatomic, strong) id publishTarget;
@property (nonatomic) uint16_t unicastAddress;
@property (nonatomic) NSMutableArray<STMeshModel*> *modelList;
@property (nonatomic) BOOL configComplete;
@property (nonatomic) NSMutableArray<STMeshGroup*> *subscribedGroups;
@property (nonatomic) STMeshNode *parentNode;
@end

@interface ProvisioningRangeObjects : NSMutableDictionary
-(id)initRangeObjectsWithMinValue:(NSString*)lowRange maxValue:(NSString*)highRange;
@end

@interface STMeshProvisionerData : STMeshNode
@property (nonatomic) NSString *provisionerName;
@property (nonatomic) NSString *provisionerUUID;
@end
```
@property (nonatomic) NSMutableArray<ProvisioningRangeObjects*> *marrProvisionerAllocatedUnicastRange;
@property (nonatomic) NSMutableArray<ProvisioningRangeObjects*> *marrProvisionerAllocatedGroupRange;
@end
@interface STMeshNetworkSettings : NSObject
@property(nonatomic) uint32_t iVindex;
@property(nonatomic) NSString *netKey;
@property(nonatomic) NSString *appKey;
@property(nonatomic) NSString *devKey;
@property(nonatomic) NSString *meshName;
@property(nonatomic) NSString *meshUUID;
@property BOOL useDefaultSecuritiesCredential;
@property(nonatomic) NSMutableArray<STMeshNode*> *nodes;
@property(nonatomic) NSMutableArray<STMeshGroup*> *groups;
@property (nonatomic) NSMutableArray<STMeshProvisionerData*> *provisionerDataArray;
@property (nonatomic) NSMutableArray<STMeshProvisionerData*> *onlyProvisionerArray;
+ (instancetype) initAsNewNetwork;
- (void)reinitNetworkDataList;
@end
@interface STMeshModel: NSObject
@property(nonatomic) uint32_t modelId;
@property(nonatomic) NSString * modelName;
@property(nonatomic) NSMutableArray<STMeshGroup*> *subscribeList;
@property(nonatomic) id publish;
@end
@interface STMeshNodeFeatures :NSObject
@property(nonatomic) uint8_t relay;
@property(nonatomic) uint8_t proxy;
@property(nonatomic) uint8_t friendFeature;
@property(nonatomic) uint8_t lowPower;
@end
6 Notes for developers

1. The library is compiled with iOS 11 as the base SDK
2. The minimum deployment target is iOS 10.2
3. The library is provided as a (arm64+x86) fat binary.
4. The library includes a Dummy mode to debug app UI functionality with a simulator (where BLE is not available).
5. The client application should use –ObjC flag in “other linker settings” to allow XCode to import categories used by demo application. If you decide not to use these categories, skipping the flag may result in a slightly smaller app archive.
6. The client application should include a dummy/empty C++ file to indicate to XCode to use C++ bindings if you encounter linking errors.
Revision history

Table 3. Document revision history

<table>
<thead>
<tr>
<th>Date</th>
<th>Version</th>
<th>Changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>21-May-2018</td>
<td>1</td>
<td>Initial release.</td>
</tr>
<tr>
<td>21-Sep-2018</td>
<td>2</td>
<td>Updated all content to reflect BlueNRG-Mesh version 1.05.000.</td>
</tr>
</tbody>
</table>
4.6.2 Key management ....................................................... 10
4.6.3 Subscription and group management ............................. 10
4.6.4 Publication .................................................................. 10
4.6.5 Composition ........................................................... 11
4.6.6 Relay ................................................................ 11
4.6.7 GATTProxy ............................................................ 11
4.6.8 Node Friend ................................................................ 11
4.6.9 Node Identity. ........................................................... 11
4.6.10 Beacon .................................................................. 11
4.6.11 TTL .................................................................. 11

4.7 Configuration method callbacks ...................................... 11

4.8 Arguments used by configuration model methods and callbacks .............................................. 12

4.9 Generic model methods .................................................. 13

4.10 Generic model callbacks ................................................ 13

4.11 Arguments used by generic model methods and callbacks ......................................................... 13

4.12 Vendor model methods. .................................................. 14
  4.12.1 Send command. ......................................................... 14
  4.12.2 Read data . ............................................................. 14
  4.12.3 Read version data. .................................................... 15

4.13 Vendor model callbacks ................................................ 15

4.14 Lighting model method ................................................... 15
  4.14.1 Lighting lightness ......................................................... 15
  4.14.2 Lighting CTL ........................................................... 16

4.15 Lighting model method callback ...................................... 16

4.16 Arguments used by configuration model methods and callbacks ......................................................... 17

4.17 Health model methods .................................................... 18
  4.17.1 Get Faults ............................................................. 18
  4.17.2 Clear Faults ........................................................... 18
  4.17.3 Get Health Period .................................................... 18
  4.17.4 Set Health Period .................................................... 18
  4.17.5 Get Health Attention ................................................ 18
  4.17.6 Set Health Attention ................................................ 18
List of figures

Figure 1. Low Energy mesh schematic .......................................................... 1
List of tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>enum STMeshStatus options</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>enum STMeshMode options</td>
<td>6</td>
</tr>
<tr>
<td>3</td>
<td>Document revision history</td>
<td>23</td>
</tr>
</tbody>
</table>
BlueNRG-Mesh iOS SDK glossary

(Mesh) Address
Mesh addresses are used to uniquely identify nodes or groups of nodes.

(Mesh) Element
An addressable unit within a node; e.g., a single socket on a switchboard with multiple sockets. Each element has its own unicast address.

(Mesh) Node
A device that is registered on a mesh network.

Generic attributes
Generic Attributes (GATT), define a hierarchical data structure used to expose data fields and their properties in BLE devices. In a mesh network, GATT is a bearer used to communicate in the mesh network. GATT bearers send and receive mesh packets via a proxy node over a BLE connection.

JavaScript Object Notation
JavaScript Object Notation is a lightweight data exchange format.

Local data
Data stored on a node is called local data.

Opcode
The vendor data consists of opcode and corresponding parameters.

Provisioner
An entity that adds an unprovisioned device to a network by allocating security information necessary for communicating in the mesh network. It also configures new nodes and sends commands to the mesh nodes.

Proxy node
Currently, nodes receive and transmit BLE Mesh messages in a mesh network through a special proxy node, and smartphones communicate with proxy node over a GATT connection.

Universally Unique ID
Universally Unique ID is the standard way to identify mesh devices on a mesh network.
IMPORTANT NOTICE – PLEASE READ CAREFULLY

STMicroelectronics NV and its subsidiaries (“ST”) reserve the right to make changes, corrections, enhancements, modifications, and improvements to ST products and/or to this document at any time without notice. Purchasers should obtain the latest relevant information on ST products before placing orders. ST products are sold pursuant to ST’s terms and conditions of sale in place at the time of order acknowledgement. Purchasers are solely responsible for the choice, selection, and use of ST products and ST assumes no liability for application assistance or the design of Purchasers’ products.

No license, express or implied, to any intellectual property right is granted by ST herein.

Resale of ST products with provisions different from the information set forth herein shall void any warranty granted by ST for such product.

ST and the ST logo are trademarks of ST. All other product or service names are the property of their respective owners.

Information in this document supersedes and replaces information previously supplied in any prior versions of this document.

© 2018 STMicroelectronics – All rights reserved