
NORDICTECH
WEBINARS
Developing Bluetooth mesh
products: Introduction

Today's hosts

Robin Saltnes

Product Marketing Engineer

Omkar Kulkarni

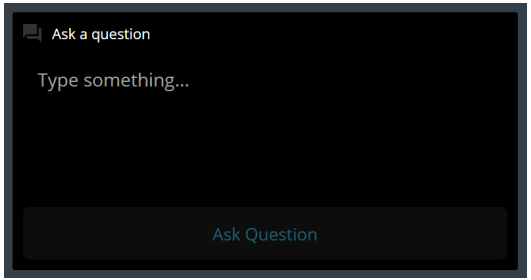
Senior R&D Engineer

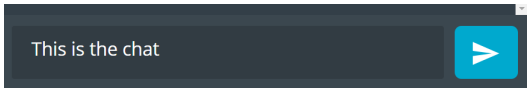
Terje Schjelderup

Application Engineer

Practicalities

- Duration: about 60 minutes
- Questions are encouraged!
  - Please type questions in the top of the right sidebar
  - All questions are anonymous
  - Try to keep them relevant to the topic
  - We will answer towards the end
- The chat is not anonymous, and should not be used for questions
- Go to DevZone if you have more questions after the webinar





{ DevZone

A recording of the webinar will be available together with the presentation at webinars.nordicsemi.com

Pre-requisites

- Webinar: Introduction to Bluetooth mesh
  - https://webinars.nordicsemi.com/introduction-to-bluetooth-mesh-4
- Webinar: Introducing nRF Connect for VS Code
  - https://webinars.nordicsemi.com/introducing-nrf-connect-for-vs-code-5

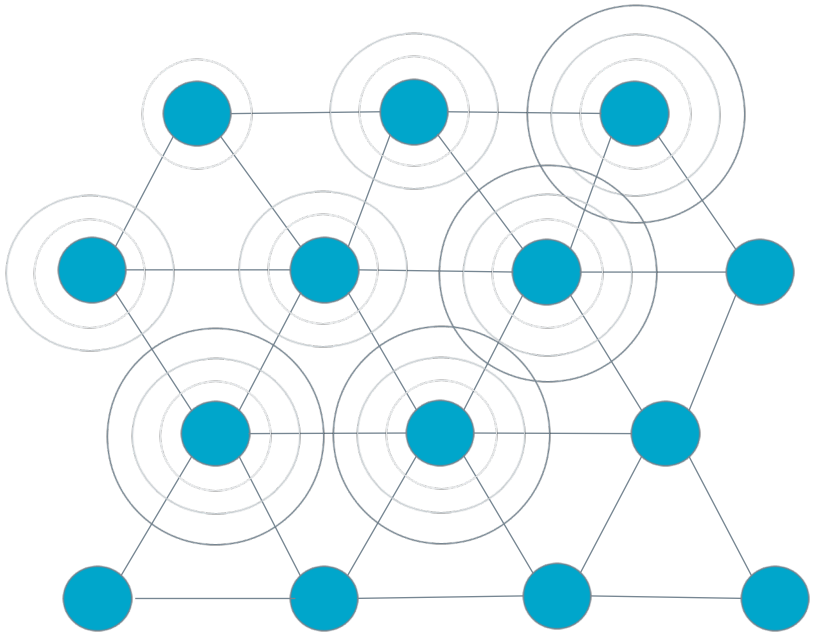
Agenda

- Bluetooth mesh: Recap
- Bluetooth mesh product: Concepts
- Samples deep dive: Code walkthrough – Light and Switch samples
  - Architecture
  - Composition data
  - Model API and handlers
- Samples: Demo and understanding mesh networking
  - Addressing
  - Node to Node communication
  - Security

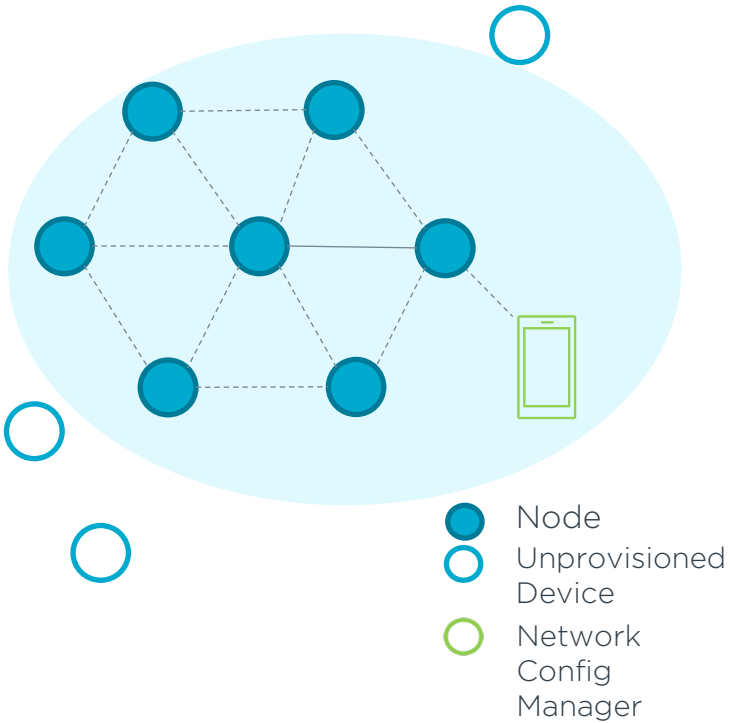

Bluetooth mesh: Recap

What is Bluetooth Mesh?

- A short-range wireless networking technology using Bluetooth Low Energy protocol as its “bearer” layer
- Mesh packets are encapsulated in advertisements or GATT packets
- Managed flooding

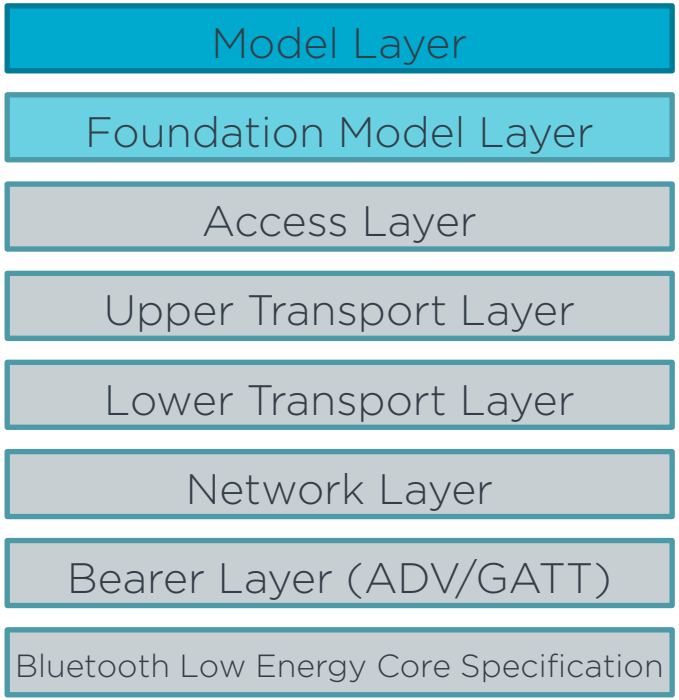


Important Terminology



- Unprovisioned Device
  - Can be provisioned into a Bluetooth mesh network
- Provisioning
  - A process of adding an unprovisioned device to a network
- Node
  - Provisioned device
- Provisioner
  - A device capable of performing provisioning
- Network Configuration Manager
  - A node that is a provisioner and has the capability to configure other mesh nodes. Typically, a phone or a tablet
- Element
  - Addressable entity within a device

Bluetooth Mesh - Layered Architecture

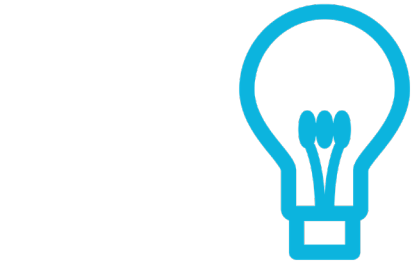


- Model layer (represents application layer)
  - Server Models holds the state and executes defined behavior
  - Client Models communicate with peer Servers to GET or SET state values or to cause action
  - Models are identified by Model IDs
- Foundation model layer handles node configuration
- Access, Upper Transport, Lower Transport, Network, Bearer
  - Networking stack layers implementing mesh networking

Model layer - Application functionality

- Bluetooth Mesh Model specification
- Standardizes typical user application scenarios with a defined set of mesh models
- Vendors can implement their own models to enhance the functionality of existing Bluetooth mesh models, or add new functionality

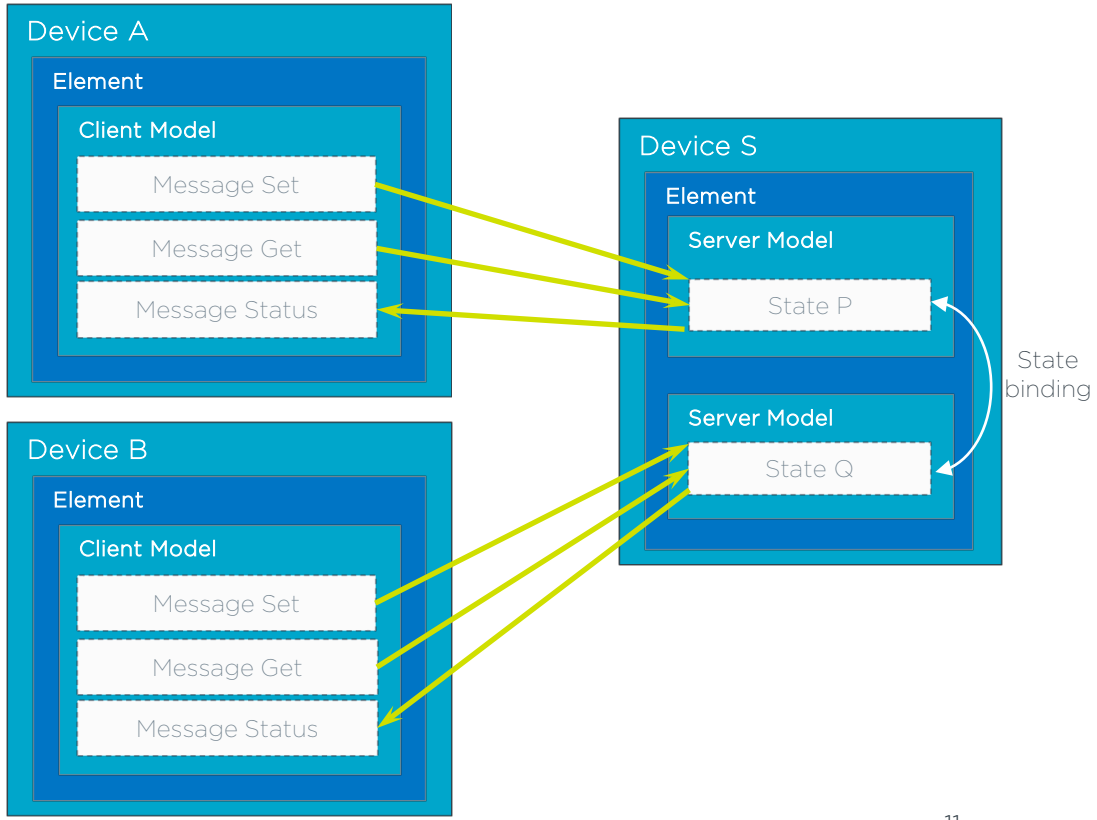
Generic OnOff Client model



Generic OnOff Server model

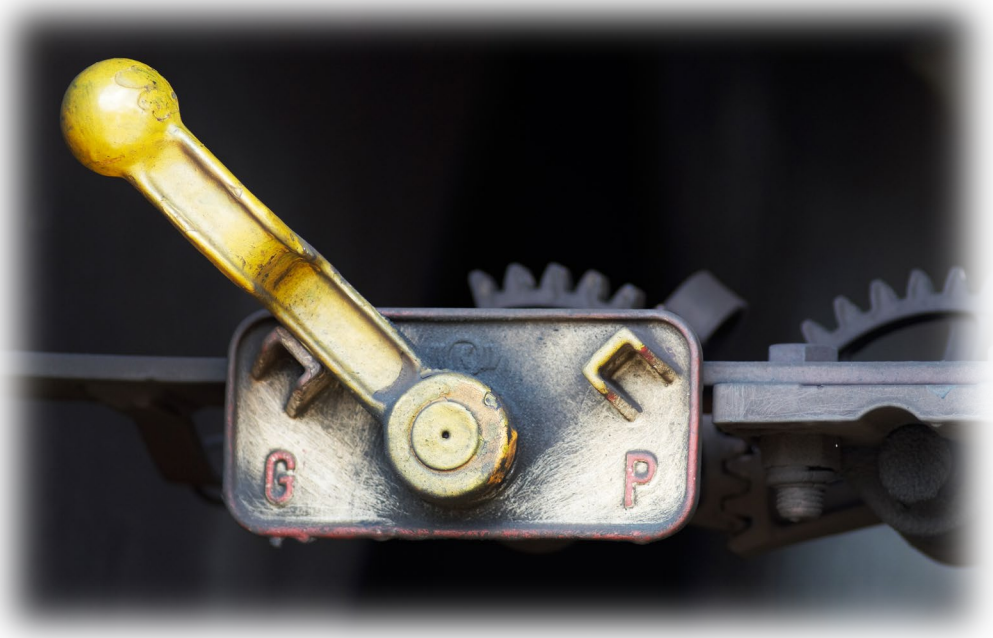
Models and States

- Server models hold states and client models set or query state values
- Change of state causes an action on the server models



Foundation model layer - Node settings

- Foundation models
  - Heart of the Node's settings and parameter control
- Config Server (and Client)
  - Subscriptions, Publications, Heartbeat, Node Features, Keys, Security procedures
- Health Server (and Client)
  - Fault monitoring, Attention timer



Bluetooth mesh product: Concepts

What constitutes a Bluetooth mesh product?

- A device that can be provisioned by any available provisioner on the market
- Once provisioned, it performs certain functionality in a mesh network. For example:
  - Relay
  - Light Fixture
  - Light Switch
  - Protocol Translator
    - Ex: Converting EnOcean Sw messages to mesh model messages
  - Proxy
  - Friend, Etc.

How provisioner ‘views’ the devices?

- All mesh nodes convey their functionality through a “Composition Data Page 0” (CDP0) state
- After device is provisioned, the Network Configuration Manager reads CDP0 to “understand” what device can do

Models and device types … [1]

- Presence of models on a device loosely identify the device functionality.
- Bluetooth Mesh Model specification provides many models for most common functionalities for networked lighting control applications.
- Of course, mesh models can be used for non-lighting applications as well, for example: controlling window-blinds, or ceiling fans
- Devices can instantiate several models for implementing complex functionality

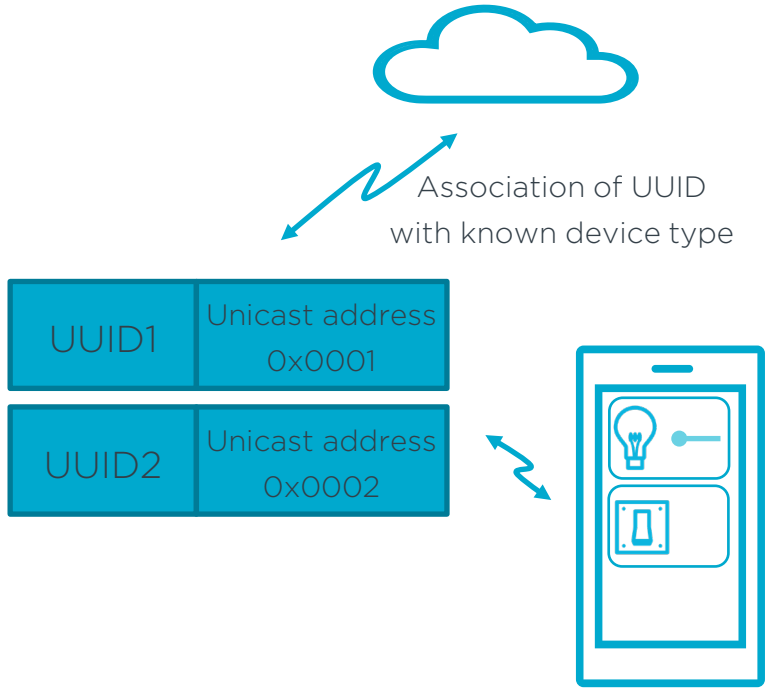
Models and device types … [2]

- Some examples

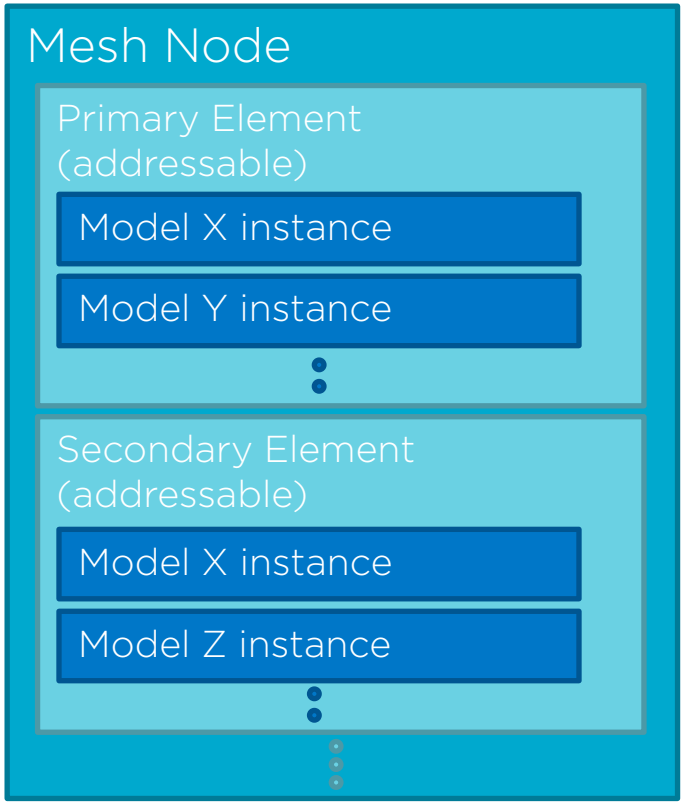
Model	Potential device type
Generic OnOff Client	On-Off switch
Generic OnOff Server	Simple light
Light LC Server	Complex light fixture with timeouts and sensor driven control
Light HSL Server	Coloured light
Light CTL Server	Light with colour temperature control
Generic Sensor Server	A sensor device

Methods inferring device type

- Device Composition Page 0 state, Or
  - Certain commonly used device types can be ‘inferred’ from the Device Composition Page 0 information
- Out of Band association of the device UUID with specific type of device, Or
- Combination of both, Or
- Custom vendor specific methods

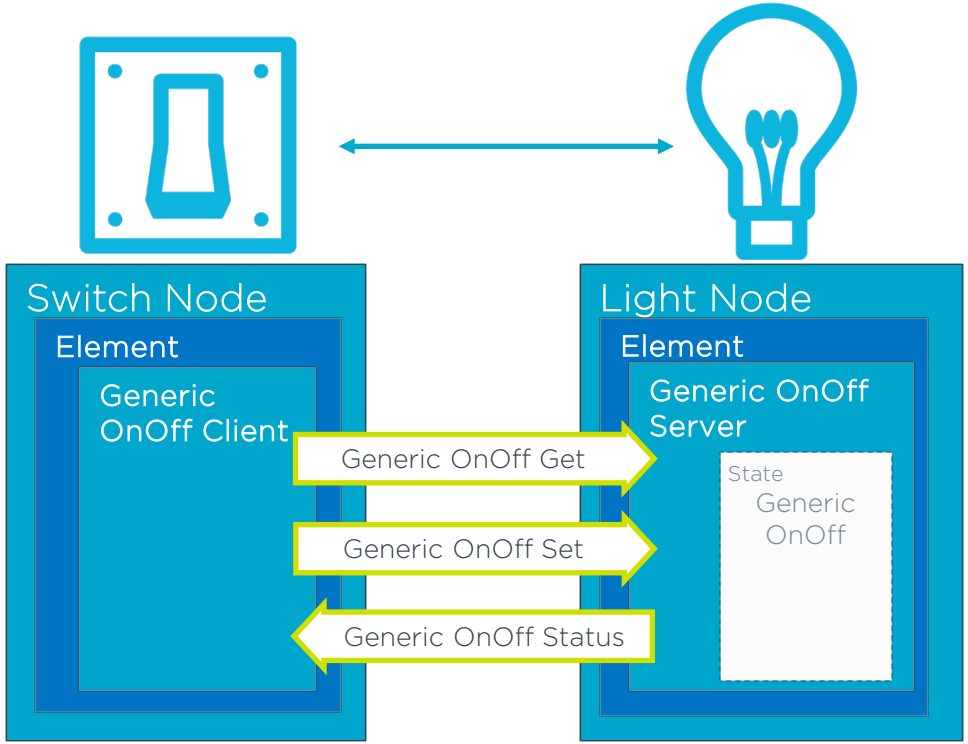


Device Composition



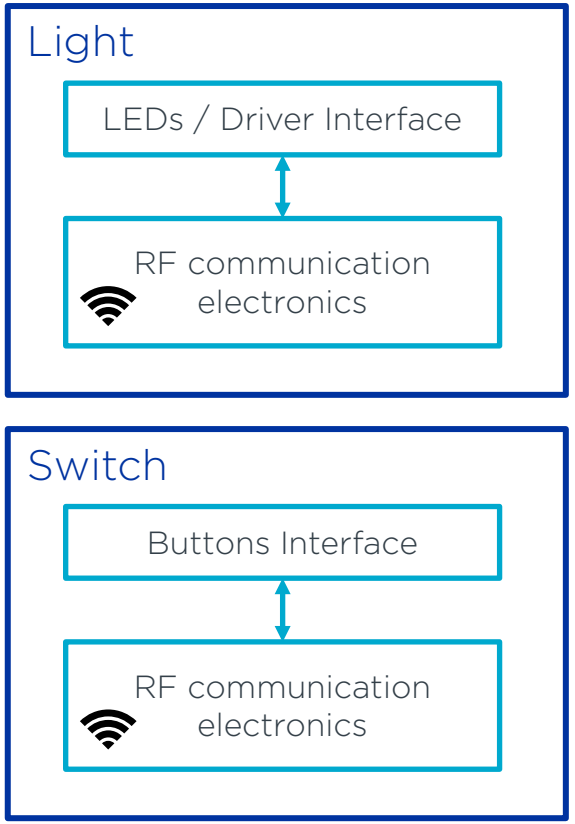
- Represents which functionality is implemented in the node
- Element is addressable entity on a node
- A single model cannot be instantiated twice on a same element
- Certain model can have associated models on several elements
- Can be used by Network Configuration Manager to inform users and populate GUI

A simplistic mesh lighting system



Light and a Switch devices

- Light
  - A lighting emitting component or an interface to such a component (driver)
  - A communication component to receive commands and generate output signals
- Switch

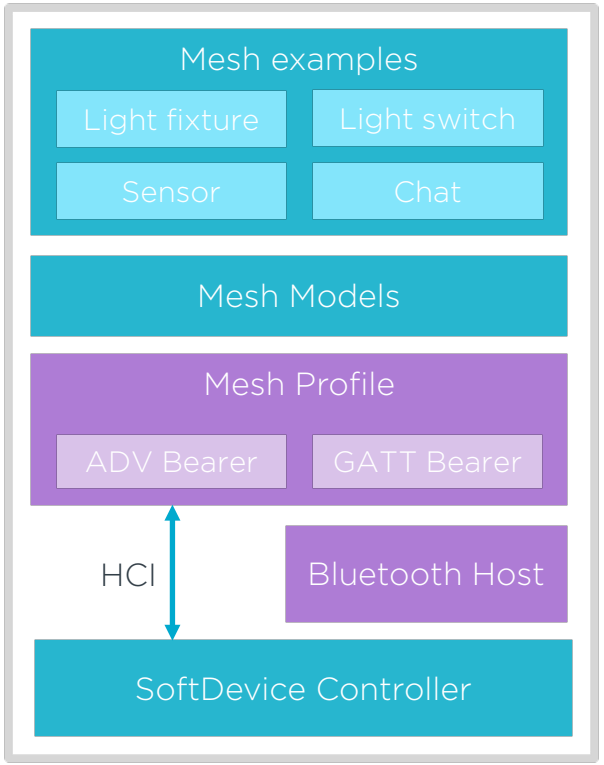


Samples deep dive …

Code walkthrough: Light and Switch samples

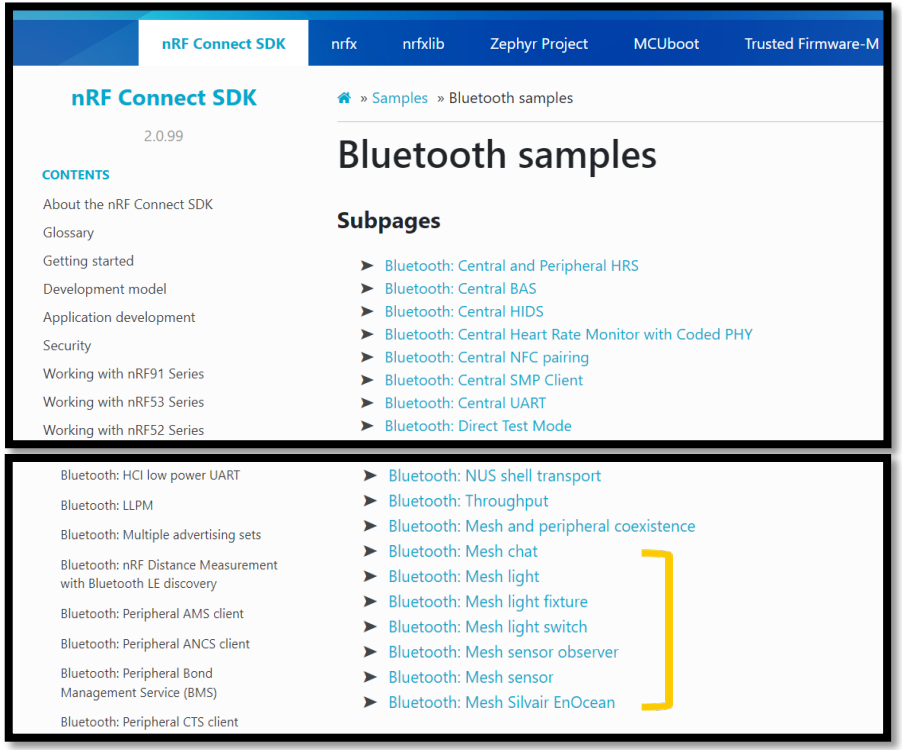
Software architecture for mesh applications

- Zephyr RTOS +
  Nordic SoftDevice controller +
  Zephyr Bluetooth mesh stack +
  Nordic Implementation of
  Bluetooth mesh models
- Various peripheral drivers, and libraries are part of nRF Connect SDK



Bluetooth mesh in nRF Connect SDK

- Uses Bluetooth Mesh Profile specification module implemented in Zephyr
- nRF Connect SDK additionally provides
  - All models from Bluetooth Mesh Model specification
  - Silvair EnOcean vendor model
  - A set of samples for evaluation
  - Comprehensive documentation
- Qualified



Light switch sample in nRF Connect SDK

Element 1	Element 2	Element 3	Element 4
Config Server	Gen. OnOff Client	Gen. OnOff Client	Gen. OnOff Client
Health Server			
Gen. OnOff Client			

- A multi-switch, that can control up to four light-fixtures using a same device
- The main model representing this functionality: Generic OnOff Client
- Four instances on four elements correspond to four buttons on DKs (where available)

Code overview – sample organization

- Common for all mesh samples
  - main.c => Initialize Bluetooth subsystem, mesh and hardware
  - model_handler.c => Bluetooth mesh specific code, device composition, model handlers

- main.c walkthrough

Code overview – composition data

- model_handler.c : walkthrough
  - How to specify composition data?
  - Model handlers and how they are used?

```
static struct bt_mesh_elem elements[] = {
#if DT_NODE_EXISTS(DT_ALIAS(sw0))
    BT_MESH_ELEM(1,
                 BT_MESH_MODEL_LIST(
                     BT_MESH_MODEL_CFG_SRV,
                     BT_MESH_MODEL_HEALTH_SRV(&health_srv, &health_pub),
                     BT_MESH_MODEL_ONOFF_CLI(&buttons[0].client)),
                 BT_MESH_MODEL_NONE),
#endif
#if DT_NODE_EXISTS(DT_ALIAS(sw1))
    BT_MESH_ELEM(2,
                 BT_MESH_MODEL_LIST(
                     BT_MESH_MODEL_ONOFF_CLI(&buttons[1].client)),
                 BT_MESH_MODEL_NONE),
#endif
#if DT_NODE_EXISTS(DT_ALIAS(sw2))
    BT_MESH_ELEM(3,
                 BT_MESH_MODEL_LIST(
                     BT_MESH_MODEL_ONOFF_CLI(&buttons[2].client)),
                 BT_MESH_MODEL_NONE),
#endif
#if DT_NODE_EXISTS(DT_ALIAS(sw3))
    BT_MESH_ELEM(4,
                 BT_MESH_MODEL_LIST(
                     BT_MESH_MODEL_ONOFF_CLI(&buttons[3].client)),
                 BT_MESH_MODEL_NONE),
#endif
};

static const struct bt_mesh_comp comp = {
    .cid = CONFIG_BT_COMPANY_ID,
    .elem = elements,
    .elem_count = ARRAY_SIZE(elements),
};
```

The light sample

- Simple light representing two states of operation
  - Either ON or OFF
- The main model representing this functionality: Generic OnOff Server

optionally present depending on availability of LEDs no DK

Element 1	Element 2	Element 3	Element 4
Config Server	Gen. OnOff Server	Gen. OnOff Server	Gen. OnOff Server
Health Server			
Gen. OnOff Server			

Code overview – composition data

- model_handler.c : walkthrough
  - How to specify composition data?
  - Model handlers and how they are used?

```
static const struct bt_mesh_lightness_srv_handlers lightness_srv_handlers = {
    .light_set = light_set,
    .light_get = light_get,
};

static struct lightness_ctx my_ctx = {
    .lightness_srv = BT_MESH_LIGHTNESS_SRV_INIT(&lightness_srv_handlers),

};

static struct bt_mesh_light_ctrl_srv light_ctrl_srv =
    BT_MESH_LIGHT_CTRL_SRV_INIT(&my_ctx.lightness_srv);

static struct bt_mesh_elem elements[] = {
    BT_MESH_ELEM(1,
                 BT_MESH_MODEL_LIST(
                     BT_MESH_MODEL_CFG_SRV,
                     BT_MESH_MODEL_HEALTH_SRV(&health_srv, &health_pub),
                     BT_MESH_MODEL_LIGHTNESS_SRV(
                         &my_ctx.lightness_srv)),
                 BT_MESH_MODEL_NONE),
    BT_MESH_ELEM(2,
                 BT_MESH_MODEL_LIST(
                     BT_MESH_MODEL_LIGHT_CTRL_SRV(&light_ctrl_srv)),
                 BT_MESH_MODEL_NONE),
};

static const struct bt_mesh_comp comp = {
    .cid = CONFIG_BT_COMPANY_ID,
    .elem = elements,
    .elem_count = ARRAY_SIZE(elements),
};
```

Code overview – model API

- Applicable for all Bluetooth mesh samples
- Model callbacks explanation
- <vs code>

Code overview – model handlers

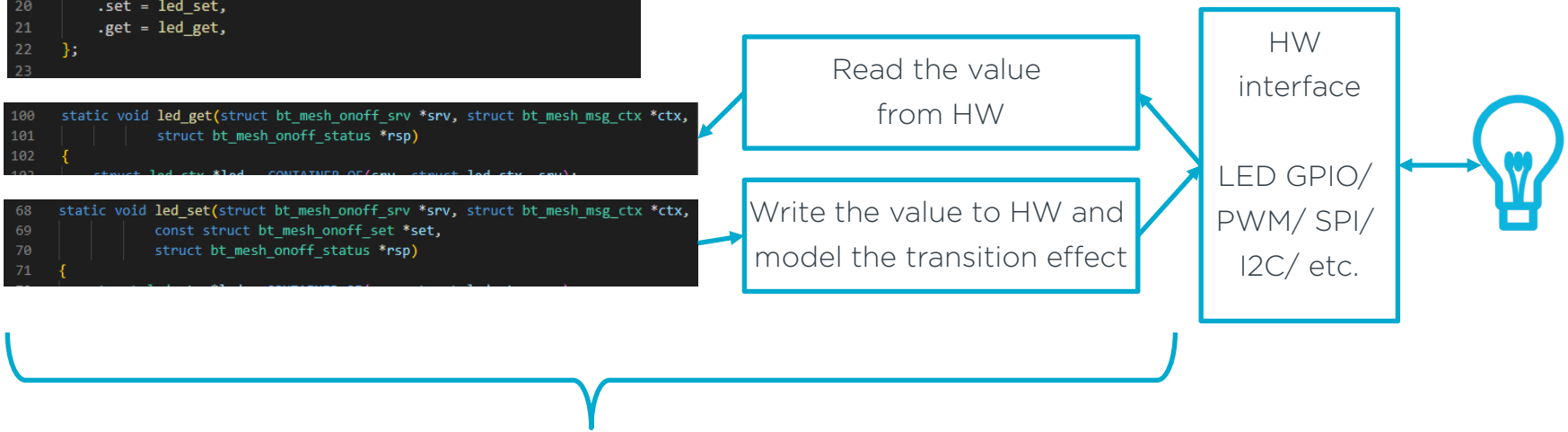
- Model handlers and how they are used?
- <vs code>

Get/Set handlers and driving LEDs

```
18
19  static const struct bt_mesh_onoff_srv_handlers onoff_handlers = {
20      .set = led_set,
21      .get = led_get,
22  };
23
```

```
100 static void led_get(struct bt_mesh_onoff_srv *srv, struct bt_mesh_msg_ctx *ctx,
101                   struct bt_mesh_onoff_status *rsp)
102 {
```

```
68  static void led_set(struct bt_mesh_onoff_srv *srv, struct bt_mesh_msg_ctx *ctx,
69                  const struct bt_mesh_onoff_set *set,
70                  struct bt_mesh_onoff_status *rsp)
71  {
```



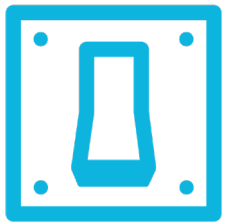
Transition behavior implementation in the application

Compliance with Bluetooth mesh model behaviors

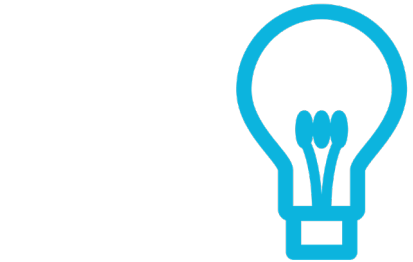
- Model interfaces are implemented in the SDK, transition behavior resides in the application.
- Mandatory to use provided callbacks
- Mandatory to implement transition behavior
- Respond to Get by fetching the current value of the LED/Light state
- Respond to Set by writing the given value of the LED/Light state
- If Transition time is specified => Model the transition

Building and Flashing

- <demo>
- Mini network scenario
- Before we run the samples …



Light Switch sample



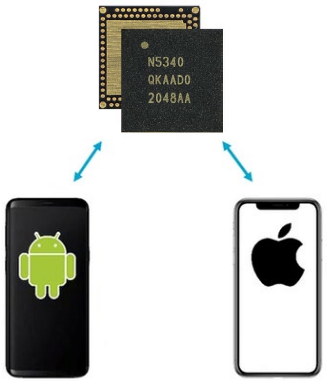
Light sample

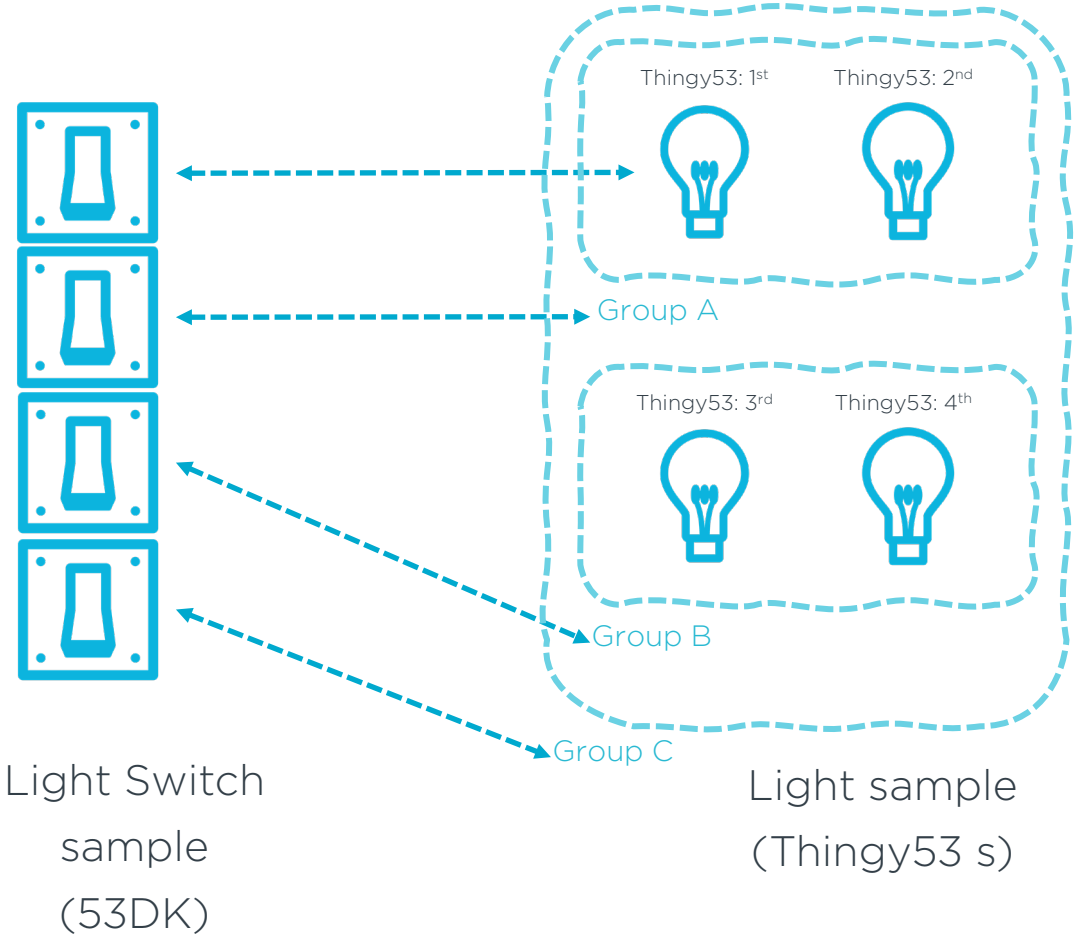
Samples: Demo

Understanding mesh networking through samples

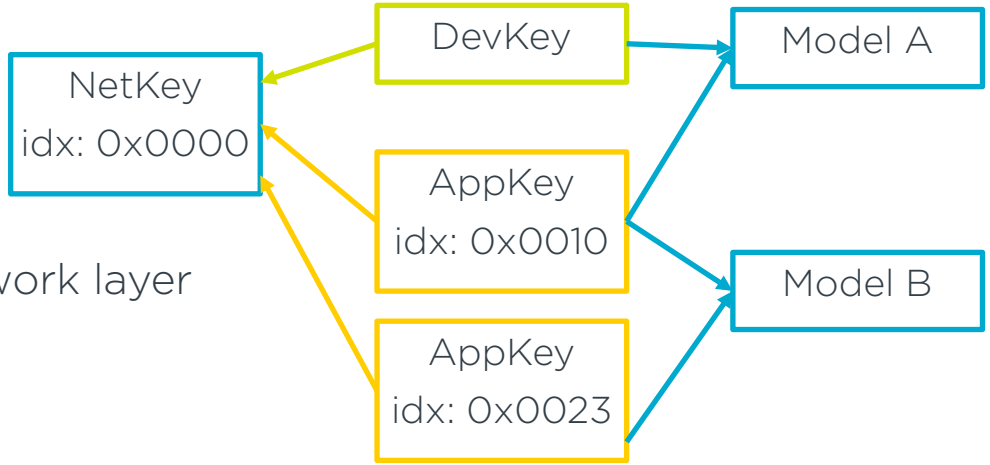
Demo

- Provisioning and Configuration
- <demo>





Message security



- Network key (NetKey):
  Encryption decryption at the network layer
- Application key (AppKey) or
  Device Key (DevKey):
  Encryption decryption at the upper transport layer
  - An AppKey is always associated with one NetKey
  - One Model instance can be bound to several AppKeys
  - Some models are explicitly bound to the DevKey
    (For example: Config Server model)

Keys and access privileges

- Keys and key-bindings can be used to give access privileges to the users.
- For example:
  - One Set of Application / Network keys => installers/managers – access all devices and all functionalities on the device.
  - Another Set of Application / Network keys => for users – access only specific devices or functionalities on the device.

Transmission/Reception and Keys

- Terms:
  - Binding a key: It is associated with ability to decrypt an incoming message with a given key and, optionally, send a response message.
  - Setting a publish key: It is associated with ability to generate outgoing messages (either periodically or upon certain event) encrypted with a given key.
- Both sender and receiver should have a correct configuration of keys
- If a sending model **sends/publishes** a message using key K => receiver model shall be **bound** to the same key K.

Addressing … [1]

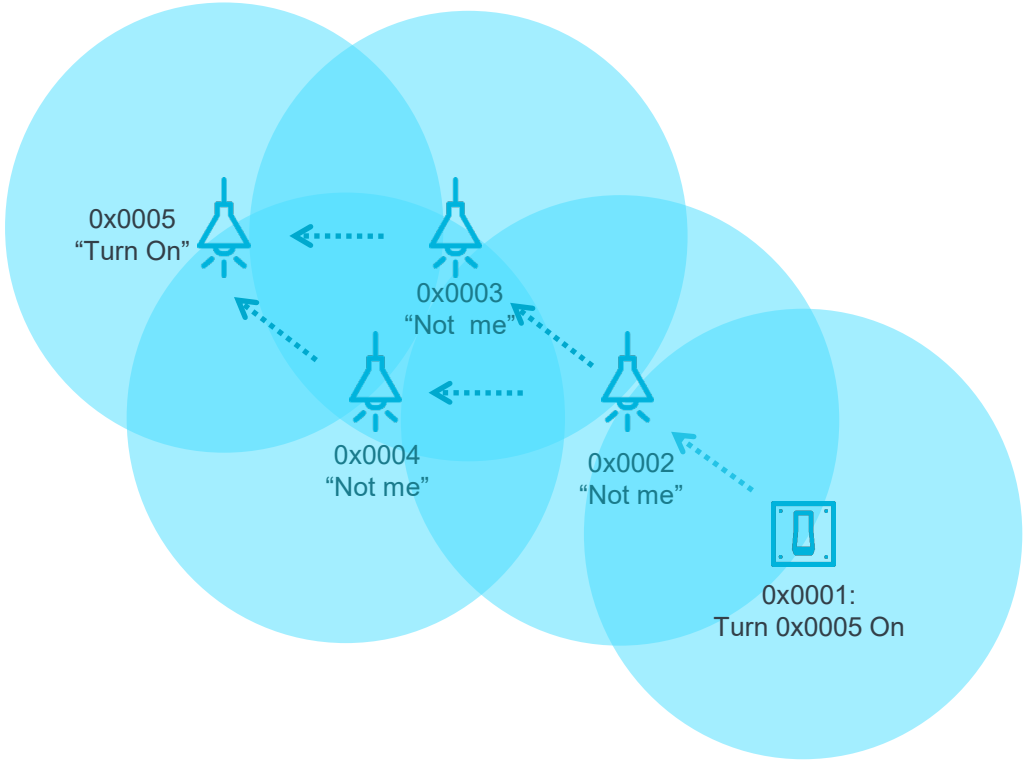
- Unassigned
  - 0x0000
  - Never used for addressing purpose
- Unicast
  - From 0x0001 to 0x7FFF
  - Allocated to device during provisioning
  - Primary element: provisioner supplied
  - Other elements: a sequential address starting from the address of the primary element
  - Maximum of 32767 addresses possible

Addressing … [2]

- Virtual
  - From 0x8000 to 0xBFFF
  - Represents a 128-bit label UUID
  - Inefficient to use as configuration messages for publication and subscriptions become longer
- Group
  - General purpose group addresses: 0xC000 to 0xFEFF
  - Fixed group addresses: 0xFF00 to 0xFFFF
    - 0xFFFF : broadcast
    - 0xFFFE : all-relays
    - 0xFFFD : all-friends
    - 0xFFFC : all-proxies

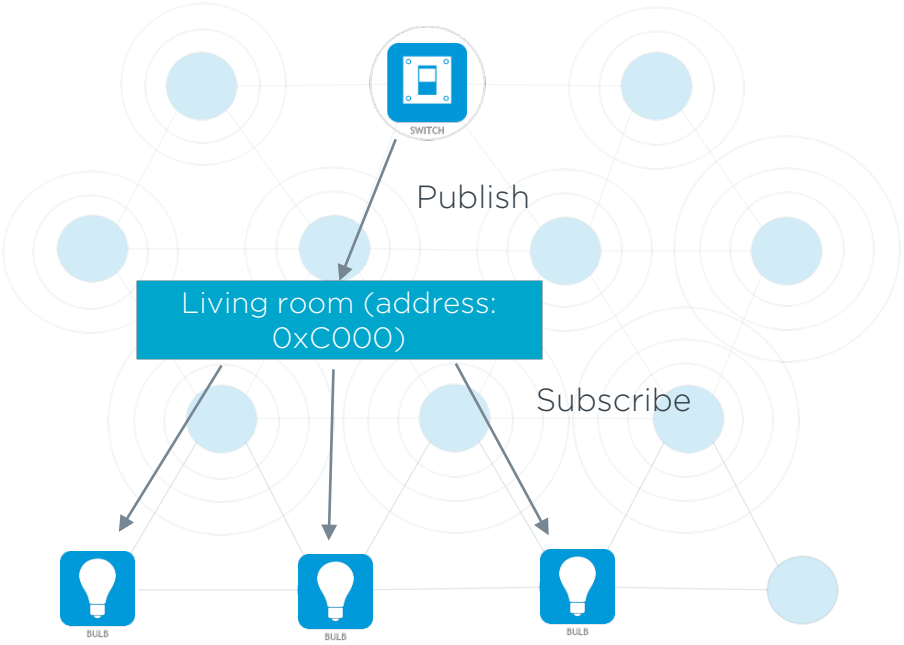
Node to Node communication: Unicast addressing

- Unicast addressing
  - Models can send/publish a message to a unicast destination address
  - one-to-one communication



Node to Node communication: Multicast addressing

- Publish <> Subscribe paradigm
  - Sender model ‘Publishes’ to a specific address
  - Receiver model ‘Subscribes’ to a specific address



Subscription

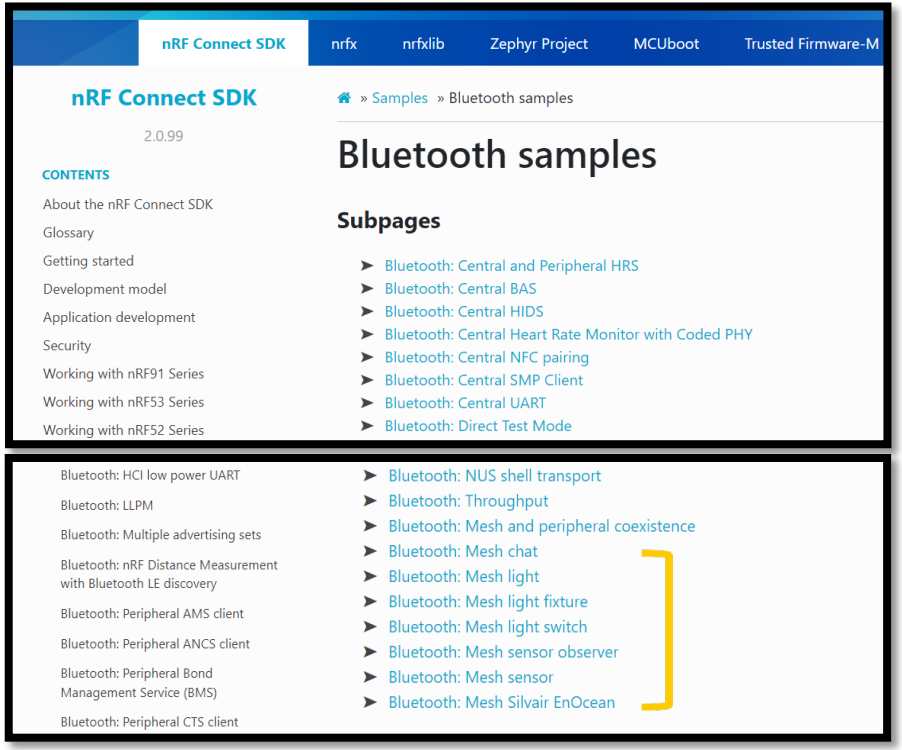
- Model subscription
  - Model can subscribe to group or virtual addresses
  - Model cannot subscribe to unicast address
    - Because sender model can publish a message directly to unicast address

Publication

- Model publication
  - Can be configured for unicast, group or virtual addresses
  - If configured for unassigned address publication is disabled.
    - Publish Address
    - Publish Period
    - Publish AppKey Index
    - Publish Friendship Credential
    - Publish TTL
    - Publish Retransmit Count
    - Publish Retransmit Interval Steps

Bluetooth mesh in nRF Connect SDK

- Uses Bluetooth Mesh Profile specification module implemented in Zephyr
- nRF Connect SDK additionally provides
  - All models from Bluetooth Mesh Model specification
  - Silvair EnOcean vendor model
  - A set of samples for evaluation
  - Comprehensive documentation
- Qualified



Thanks!