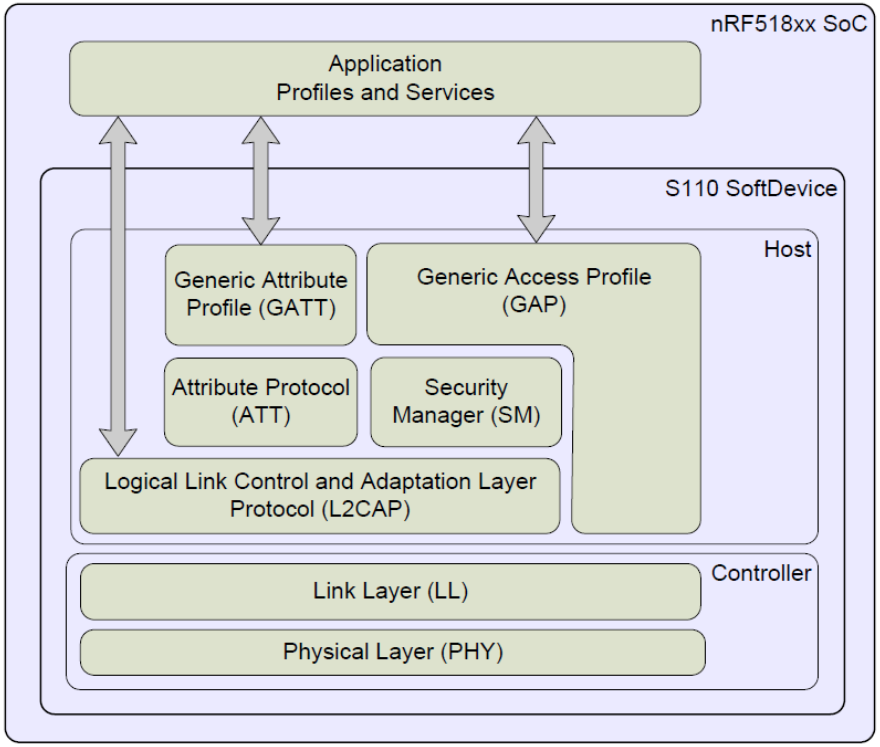
Building a GATT/GAP Profile

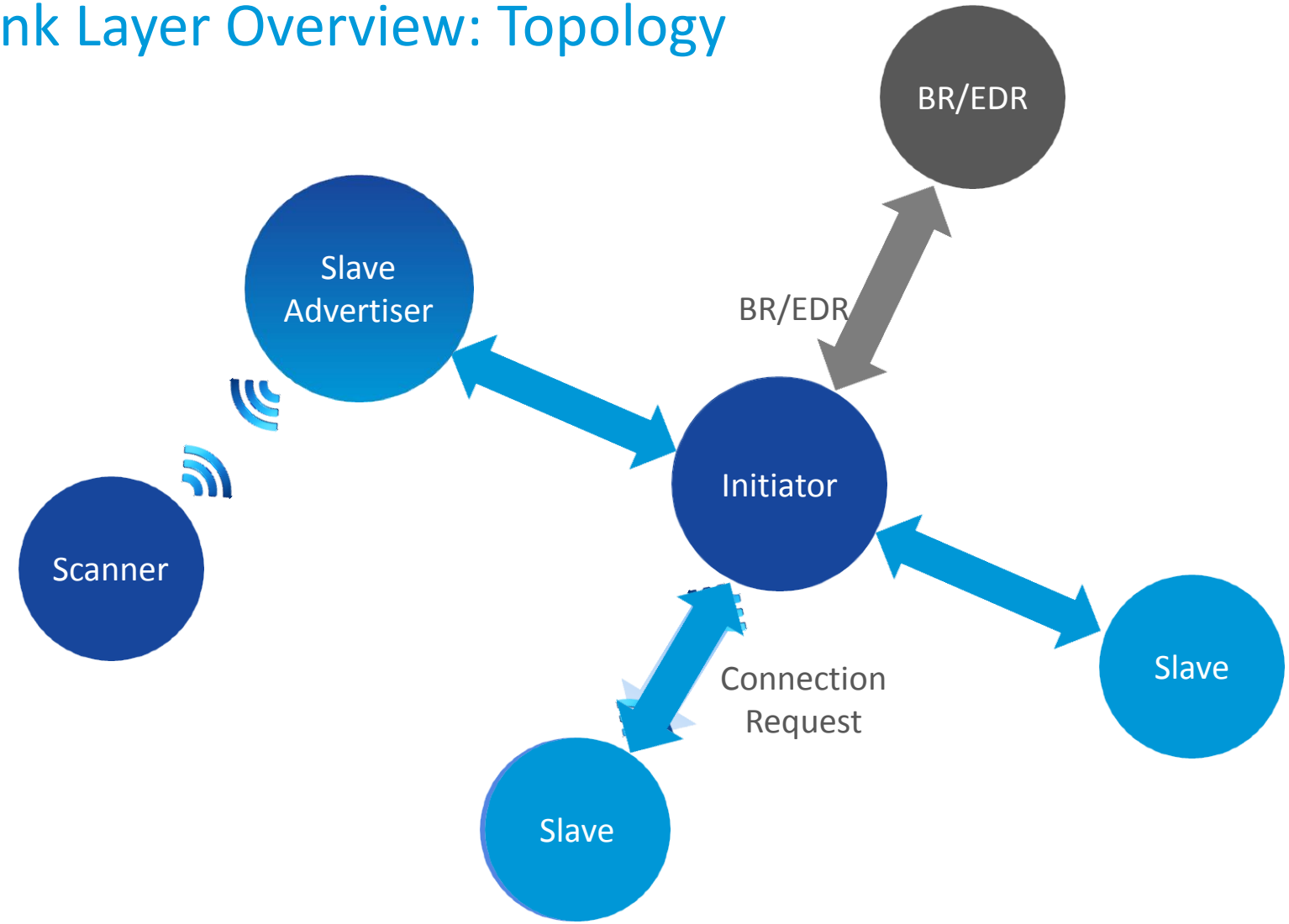
nRF52 Global Tech Tour

How a BLE application works

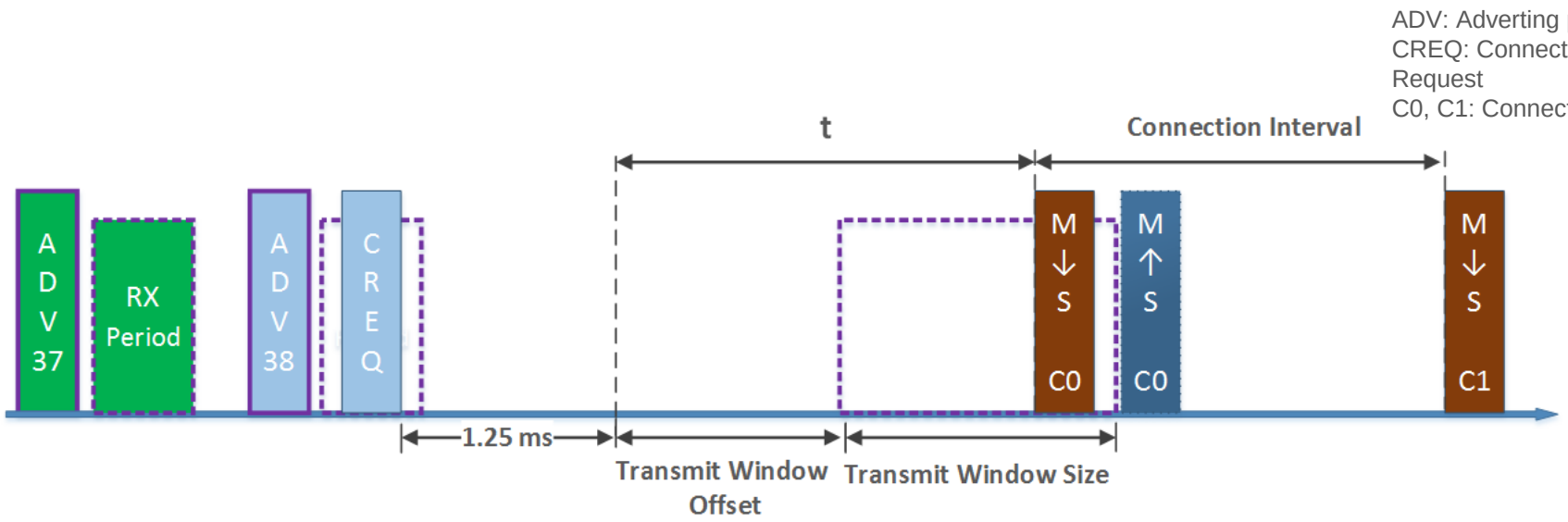
- How a BLE application works?
- SoftDevice is a protocol stack solution
  - that runs in a protected code area
  - Accompanying protected RAM area.
- SoftDevice is a precompiled and pre-linked HEX file
  - Independent from the application
  - Can be programmed separately.



Link Layer Overview: Topology



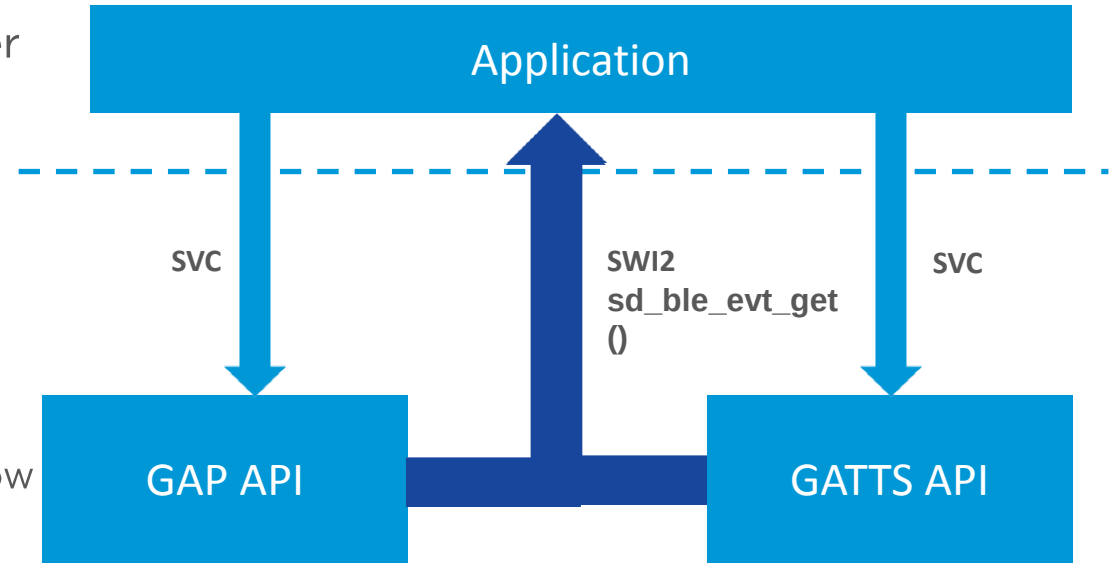
Link Layer Overview: Connection establishing



- Slave advertises in all three channels, there is RX period on each channel
- Slave stops advertising on consecutive channel when receives a request from master
- A Transmit Window is opened for first connection event

Bluetooth® low energy API

- Generic Access Profile (GAP)
- Generic Attribute Profile Server
  - (GATTS)
- API calls as SuperVisor Calls
  - Switches Core to SV priority
  - Each SV Call numbered
- Events as SoftWare Interrupts
  - Always through SWI2
  - Interrupt priority: Application Low
  - `sd_ble_evt_get()`
    - For all BLE events
    - From ISR or main context



BLE API: GAP SVC

- GAP Service (built in to the stack)
  - sd_ble_gap_device_name_set(security, name)
    - Sets the device name and security mode for the device name characteristic
  - sd_ble_gap_appearance_set(appearance)
    - Describes what our device does to central peers
  - sd_ble_gap_ppcp_set(ppcp)
    - Defines the connection parameters
  - …
- GAP Advertisement
  - sd_ble_gap_adv_data_set(adv_data, ad_len, sr_data, sr_len)
    - Sets the advertisement data that central peers will receive
  - sd_ble_gap_adv_start(adv_params)
    - Starts sending advertisement packets over the air
  - …

BLE API: GAP Events

- BLE_GAP_EVT_CONNECTED {conn_handle, peer_addr, conn_params}
  - A central peer has established a physical connection
- BLE_GAP_EVT_DISCONNECTED {conn_handle, reason}
  - The connection has been terminated, locally or remotely
- BLE_GAP_EVT_CONN_PARAM_UPDATE {conn_params}
  - A connection parameter update procedure has completed
- BLE_GAP_EVT_TIMEOUT {source}
  - A procedure has timed out (advertisement, security, …)
- …

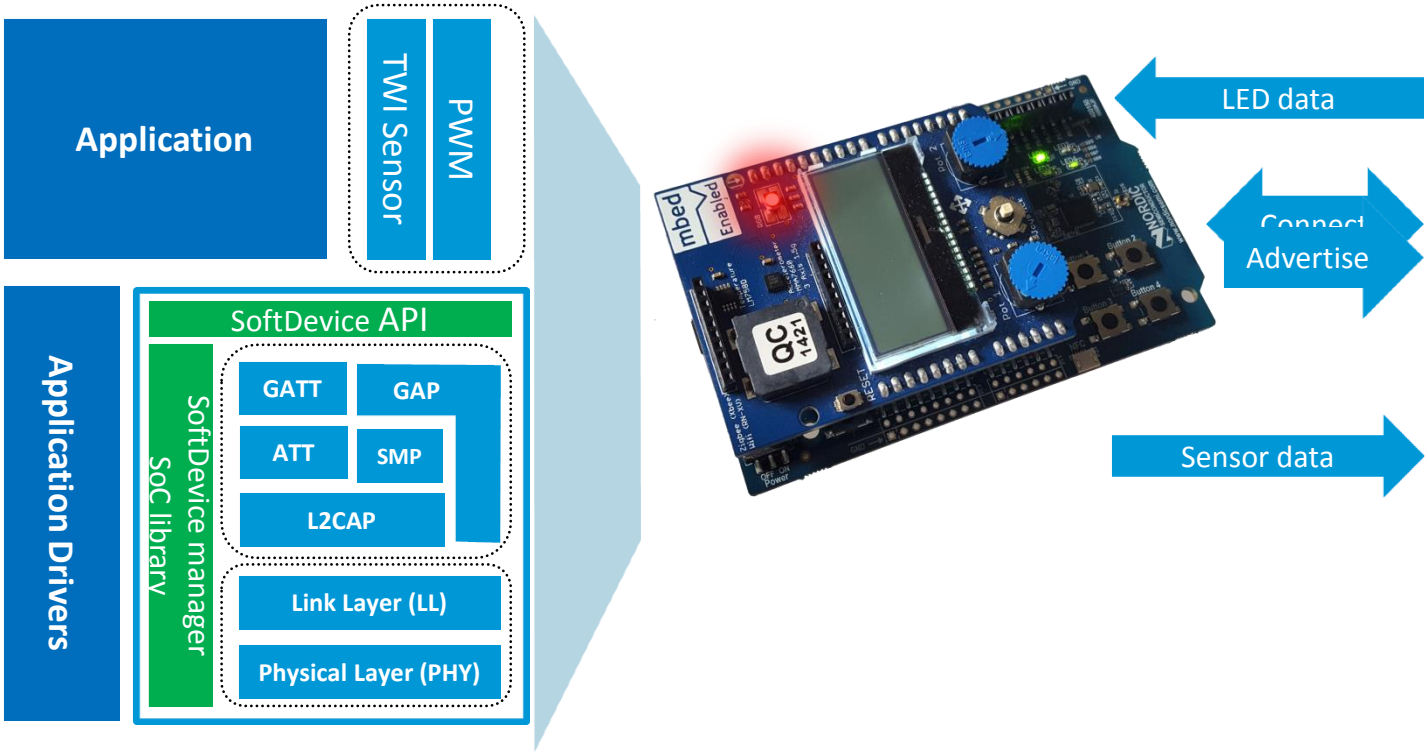
BLE API: GATTS SVC

- ATT Table Population
  - sd_ble_gatts_service_add(type, UUID, out_handle)
    - Adds an empty Service to the ATT Table
  - sd_ble_gatts_characteristic_add(svc_handle, md, value, out_handles)
    - Adds a Characteristic to the referenced service
- ATT Table Local Access
  - sd_ble_gatts_value_set(handle, offset, len, value)
    - Sets the value of any particular attribute
  - sd_ble_gatts_value_get(handle, offset, len, value)
    - Gets the value of any particular attribute
- Server Initiate
  - sd_ble_gatts_hvx(conn_handle, params)
    - Sends an ATT Notification or Indication

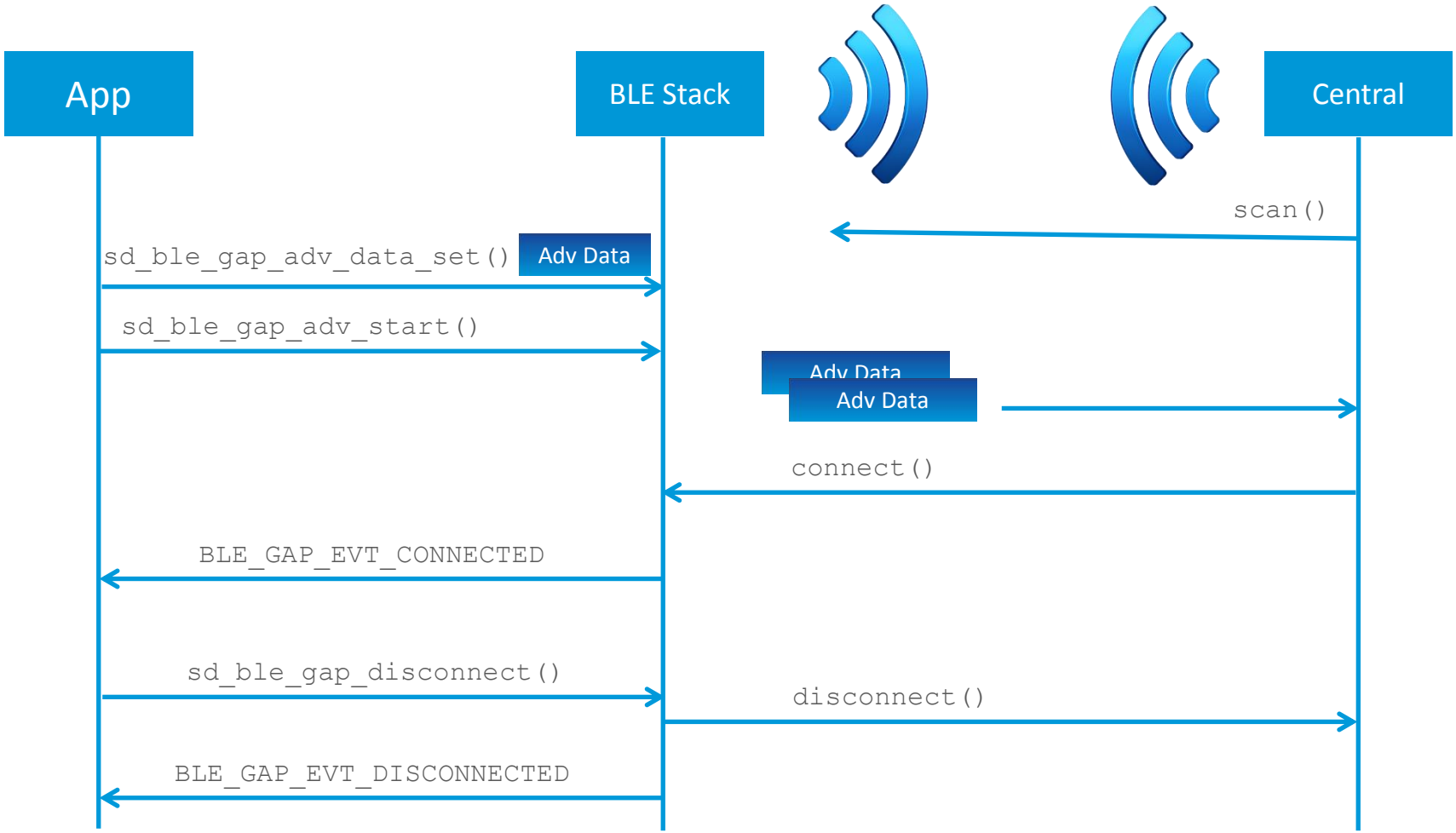
BLE API: GATTS Events

- BLE_GATTS_EVT_WRITE {conn_handle, handle, data}
  - An incoming client ATT Write operation has received and executed
- BLE_GATTS_EVT_HVC {conn_handle, handle}
  - A Handle Value Confirmation has been received from the peer
- . . .

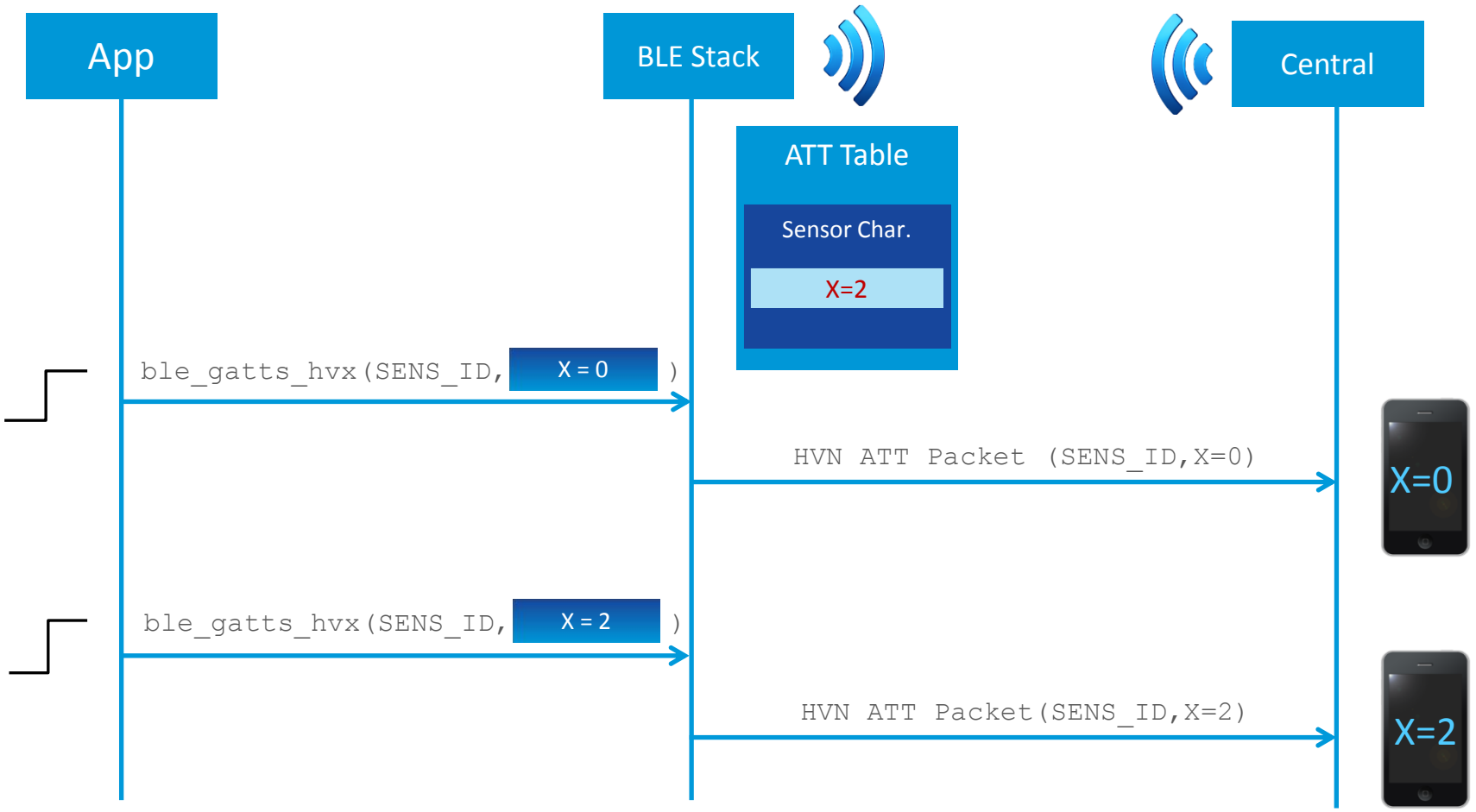
LED Sensor application



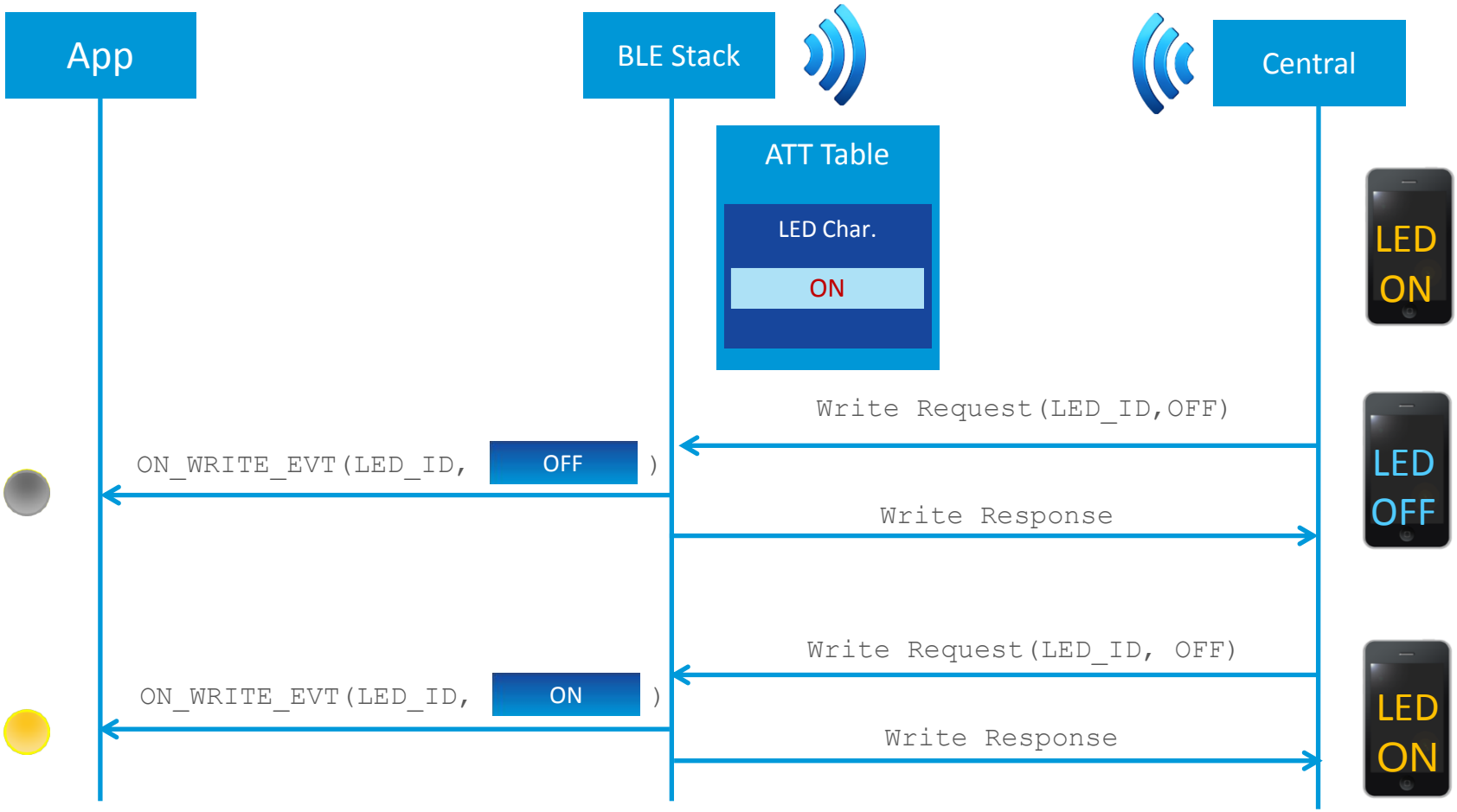
BLE API: Connection Sequence



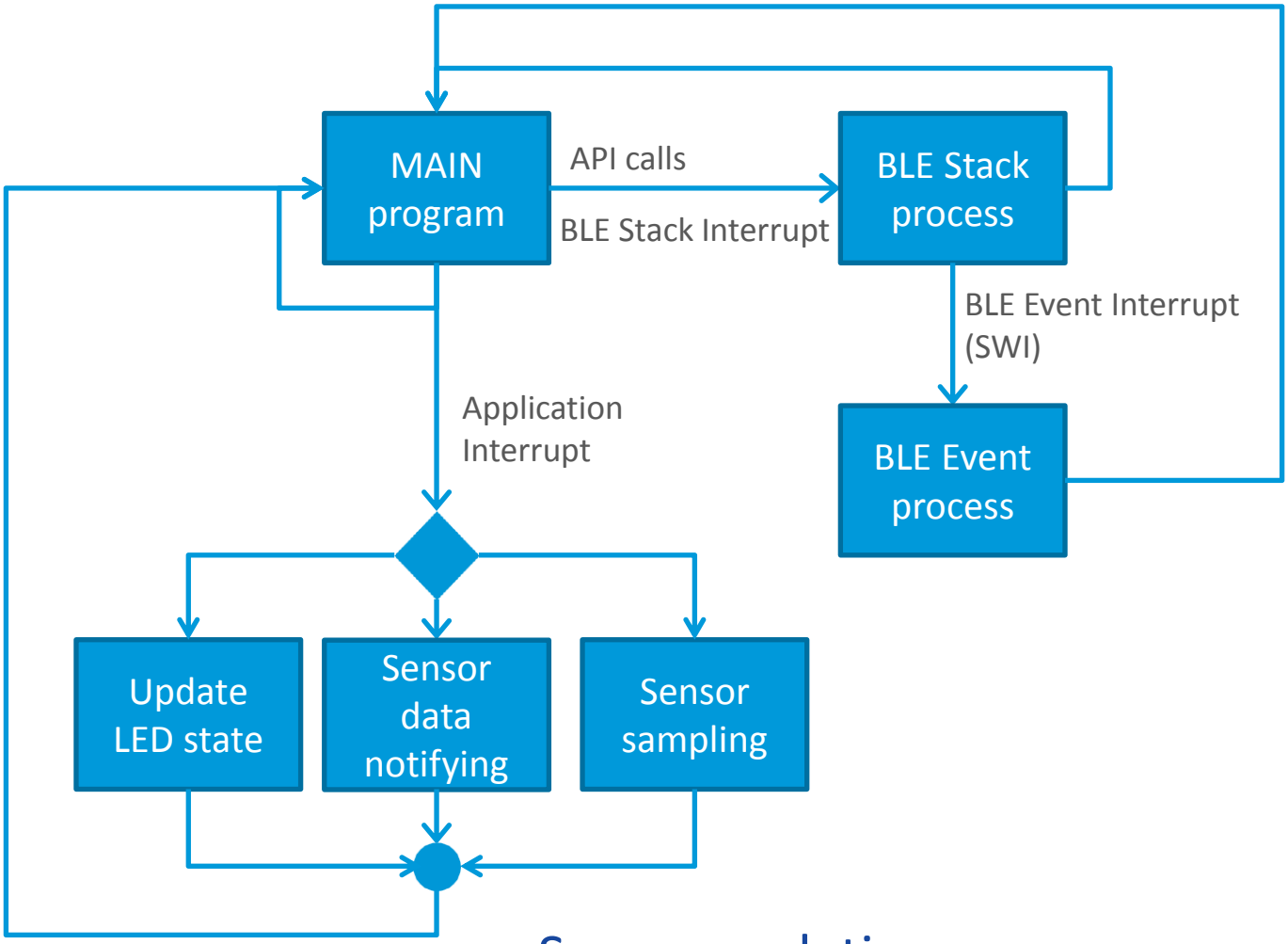
BLE API: Handle Value Notification



BLE API: Handle Write Commands

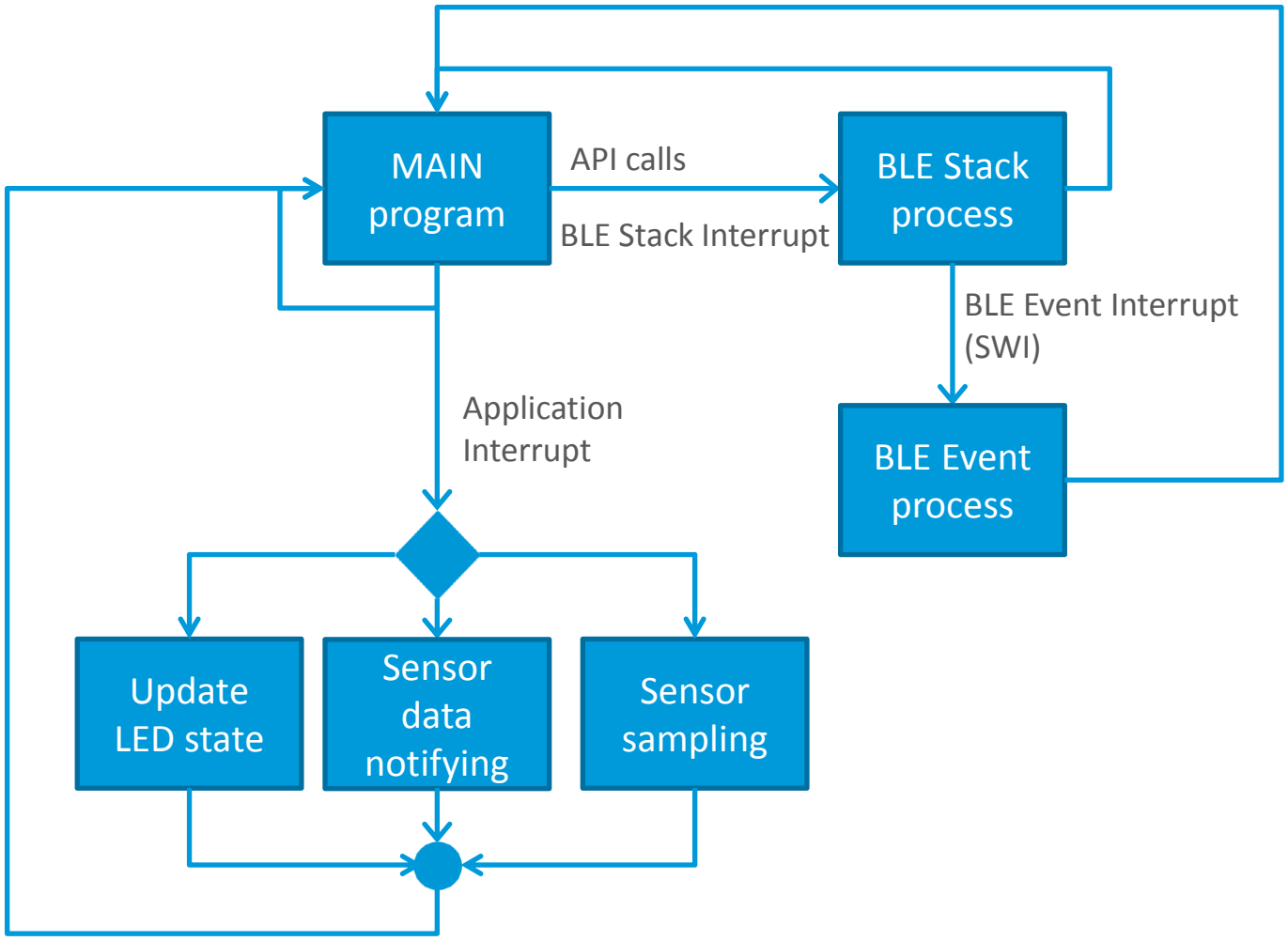


Application block diagram



Sensor updating

Application block diagram



LED updating

Building the data structure

- Sensor characteristic
  - 3 bytes to notify the sensor state.
    - X: -128 to 127
    - Y: -128 to 127
    - Z: -128 to 127
  - Properties:
    - Read
    - Notification
  - Permission:
    - Read

- LED characteristic
  - Six bytes to update the LED state
    - Color 1 - Red: 0 to 255
    - Color 1 - Green: 0 to 255
    - Color 1 - Blue: 0 to 255
    - Color 2 - Red: 0 to 255
    - Color 2 - Green: 0 to 255
    - Color 2 - Blue: 0 to 255
  - Properties:
    - Read
    - Write
  - Permission:
    - Read
    - Write

Standard versus custom services and characteristics

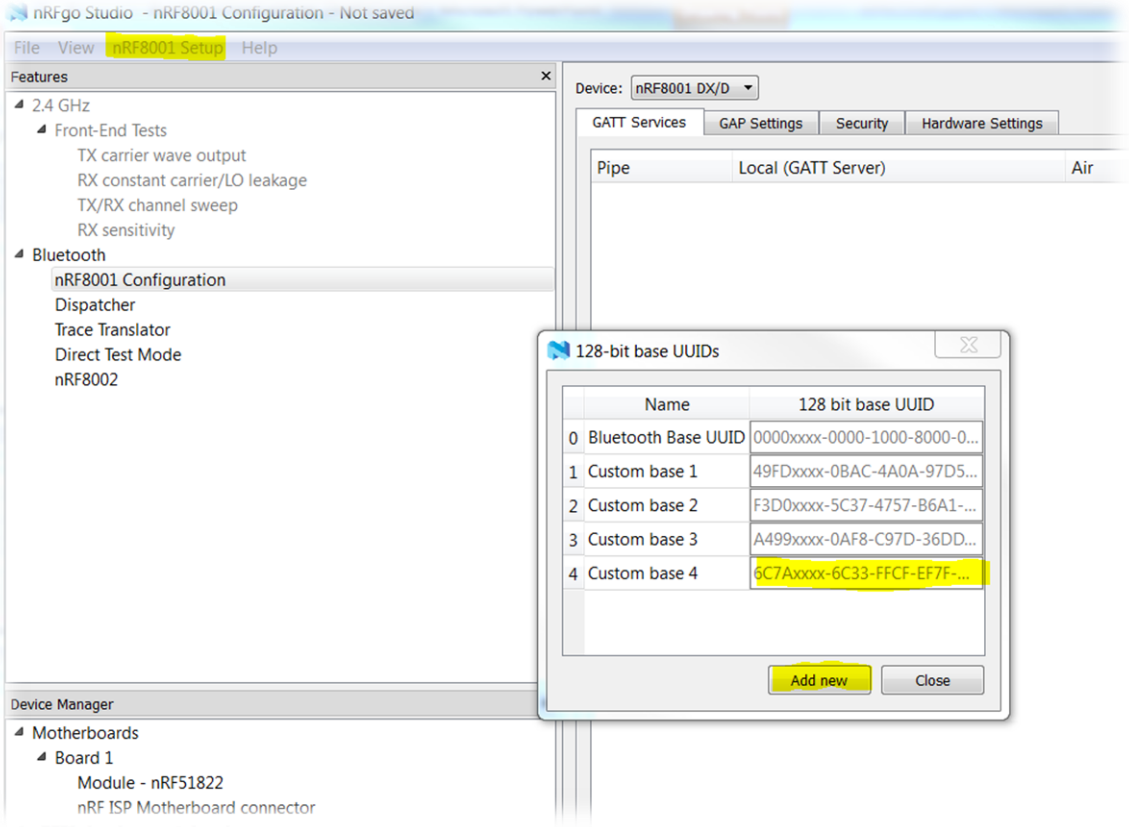
Description	UUID	Properties
Led Sensor Service	0x6E400001-B5A3-F393-E0A9-E50E24DCCA9E	
Sensor Characteristic	0x6E400002-B5A3-F393-E0A9-E50E24DCCA9E	Read, Notify
LED Characteristic	0x6E400003-B5A3-F393-E0A9-E50E24DCCA9E	Write

- A U
  sep
- Bas
  - E
    - 0x0000xxxx-0000-1000-8000-00805F9B34FB
  - For LED Sensor example, a base UUID is generated:
    - 0x6E40xxxx-B5A3-F393-E0A9-E50E24DCCA9E

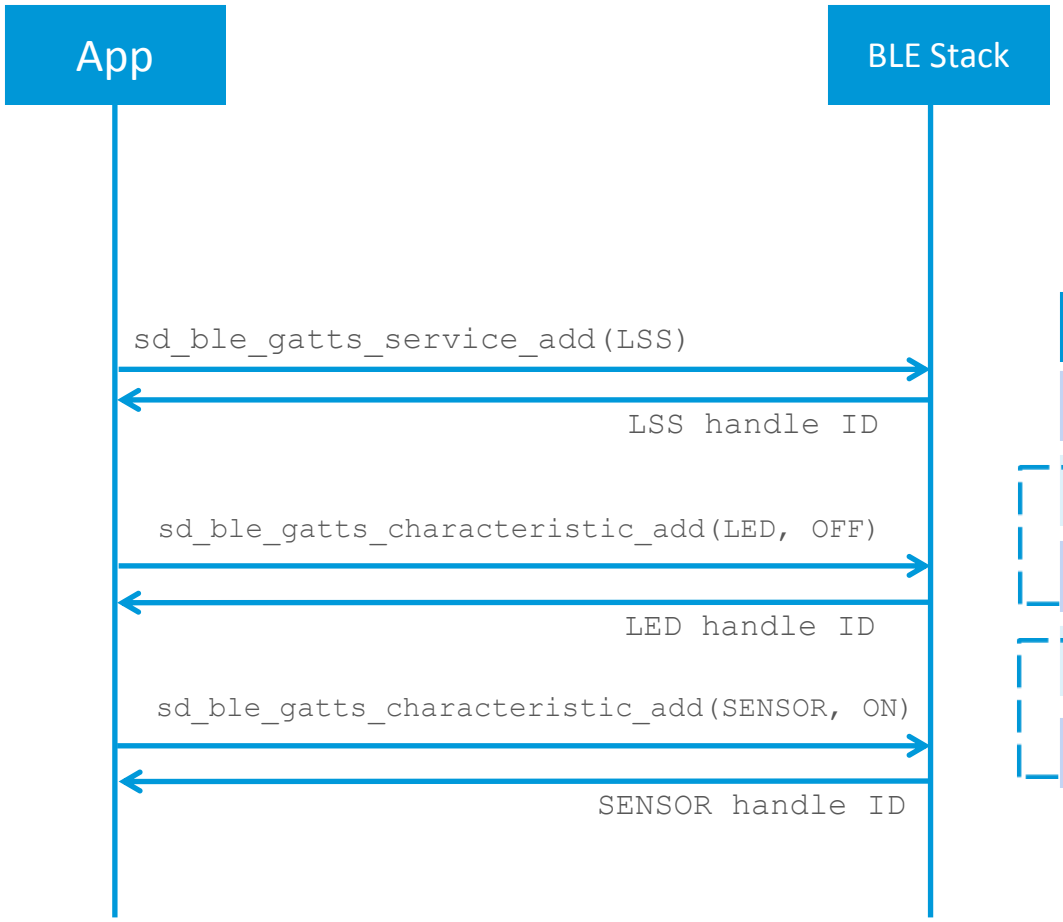
- Alias is the remaining 16 bits not defined by the Base UUID.
  - For example 0x180F for the Battery Service UUID, 0x2A19 for the Battery Level Characteristic UUID, etc
  - For LED Sensor example
    - Service: 0x0001
    - Sensor Characteristic: 0x0002
    - LED Characteristic: 0x0003

Standard versus custom services and Characteristics

How to generate a 128 bit UUID using 8001 configuration tool:



BLE API: Service population



Handle	UUID	Value
0x0001	*SERVICE*	*LSS*
0x0002	*CHAR*	*LED*
0x0003	*LED*	0, 0, 0
0x0004	*CHAR*	*SENSOR*
0x0005	*SENSOR*	*0, 0, 0*

Designing the API

The LED Sensor service needs to:

- Notify the central device when the sensor state is changed. So, we need to add a method to be called when the sensor state changes.

```
uint32_t ble_lss_on_sensor_change(ble_lss_t * p_lss, uint8_t *sensor_state, uint16_t length)
{
        params.type = BLE_GATT_HVX_NOTIFICATION;
        params.handle = p_lss->sensor_char_handles.value_handle;
        params.p_data = sensor_state;
        params.p_len = &len;

        return sd_ble_gatts_hvx(p_lss->conn_handle, &params);
}
```

Designing the API

The LED Sensor service needs to:

- Update the LED state when a write command is received from the central device. The service need to handle the write event and other events received from the stack.

```
void ble_lss_on_ble_evt(ble_lss_t * p_lss, ble_evt_t * p_ble_evt);
{
        switch (p_ble_evt->header.evt_id)
            case BLE_GAP_EVT_CONNECTED:
               on_connect(p_lss, p_ble_evt);
               break;

            case BLE_GAP_EVT_DISCONNECTED:
               on_disconnect(p_lss, p_ble_evt);

            case BLE_GATTS_EVT_WRITE:
               on_write(p_lss, p_ble_evt);
}
```

Designing the API

The LED Sensor service needs to know:

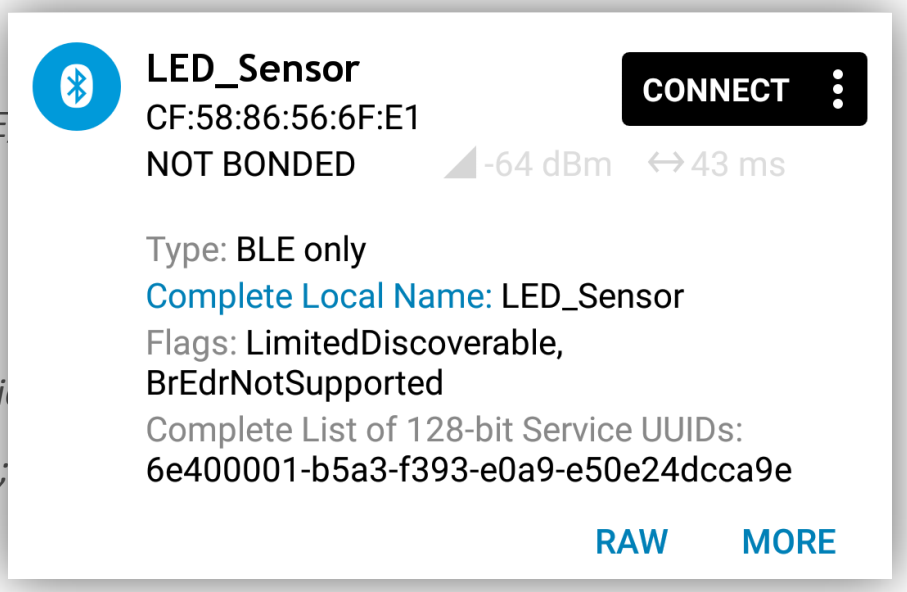
- When a write command is received from the central device to update the LED state. The service need to handle the write event received from the stack.

```
static void on_write(ble_lss_t * p_lss, ble_evt_t * p_ble_evt)
{
    ble_gatts_evt_write_t * p_evt_write = &p_ble_evt->evt.gatts_evt.params.write;

    if ((p_evt_write->handle == p_lss->led_char_handles.value_handle) &&
        (p_evt_write->len == 6) && (p_lss->led_write_handler != NULL))
    {
            p_lss->led_write_handler(p_lss, p_evt_write->data[0]);
    }

}
```

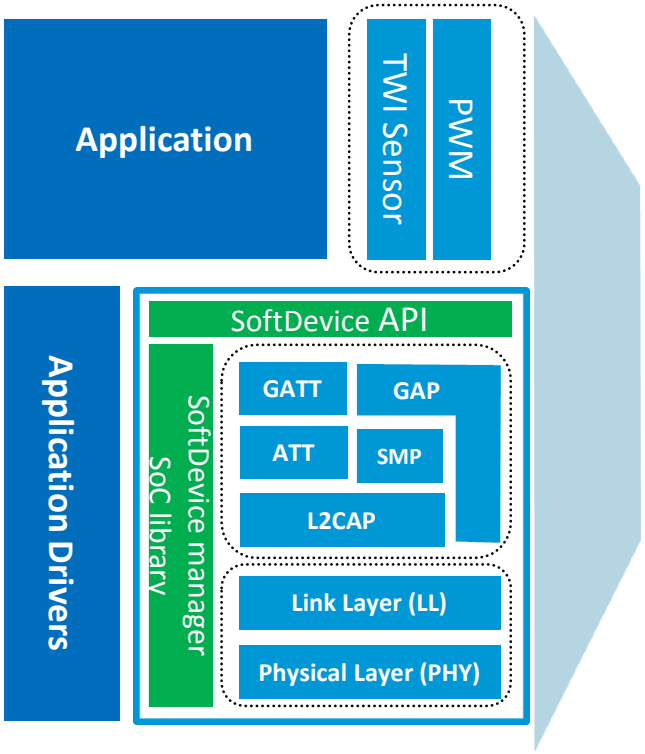
Last step: setting up Advertising packet

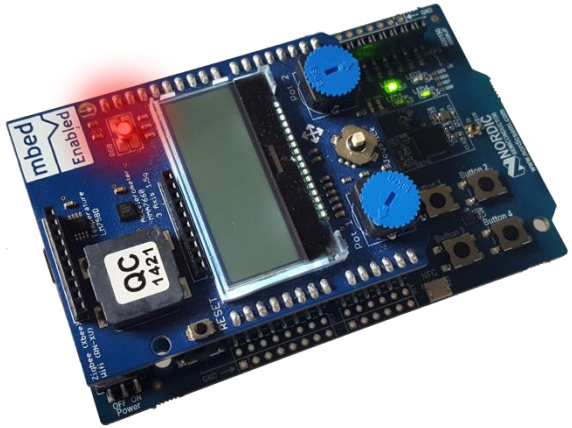
We will try to add the LSS service UUID into the advertising data. The advertising packet can contain 31 bytes, and if additional data needed a scan response data packet can be used.

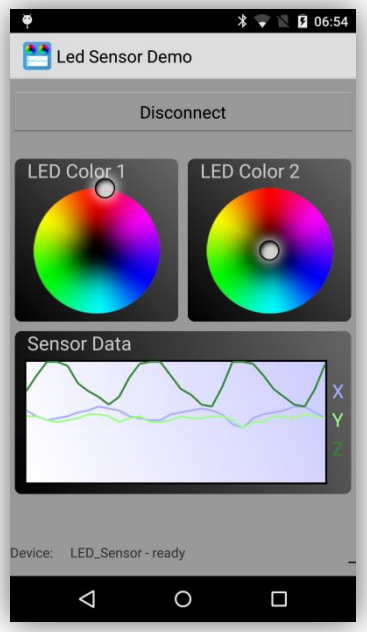
```
static void advertising_init(void)
{
    uint8_t flags = BLE_GAP_ADV_FLAGS_LE_ONLY_LIMITED_DISC_MODE;

    ble_uuid_t adv_uuids[] = {{LSS_UUID_SERVICE, m_lss.uuid_type}};

    // Build and set advertising packet data
    memset(&advdata, 0, sizeof(advdata));
    advdata.name_type = BLE_ADVDATA_FULL_NAME
    advdata.include_appearance = true;
    advdata.flags.size = sizeof(flags);
    advdata.flags.p_data = &flags;

    // Build and set scan response packet data
    memset(&scanrsp, 0, sizeof(scanrsp));
    scanrsp.uuids_complete.uuid_cnt = sizeof(adv_uui
    scanrsp.uuids_complete.p_uuids = adv_uuids;
    err_code = ble_advdata_set(&advdata, &scanrsp);
}
```



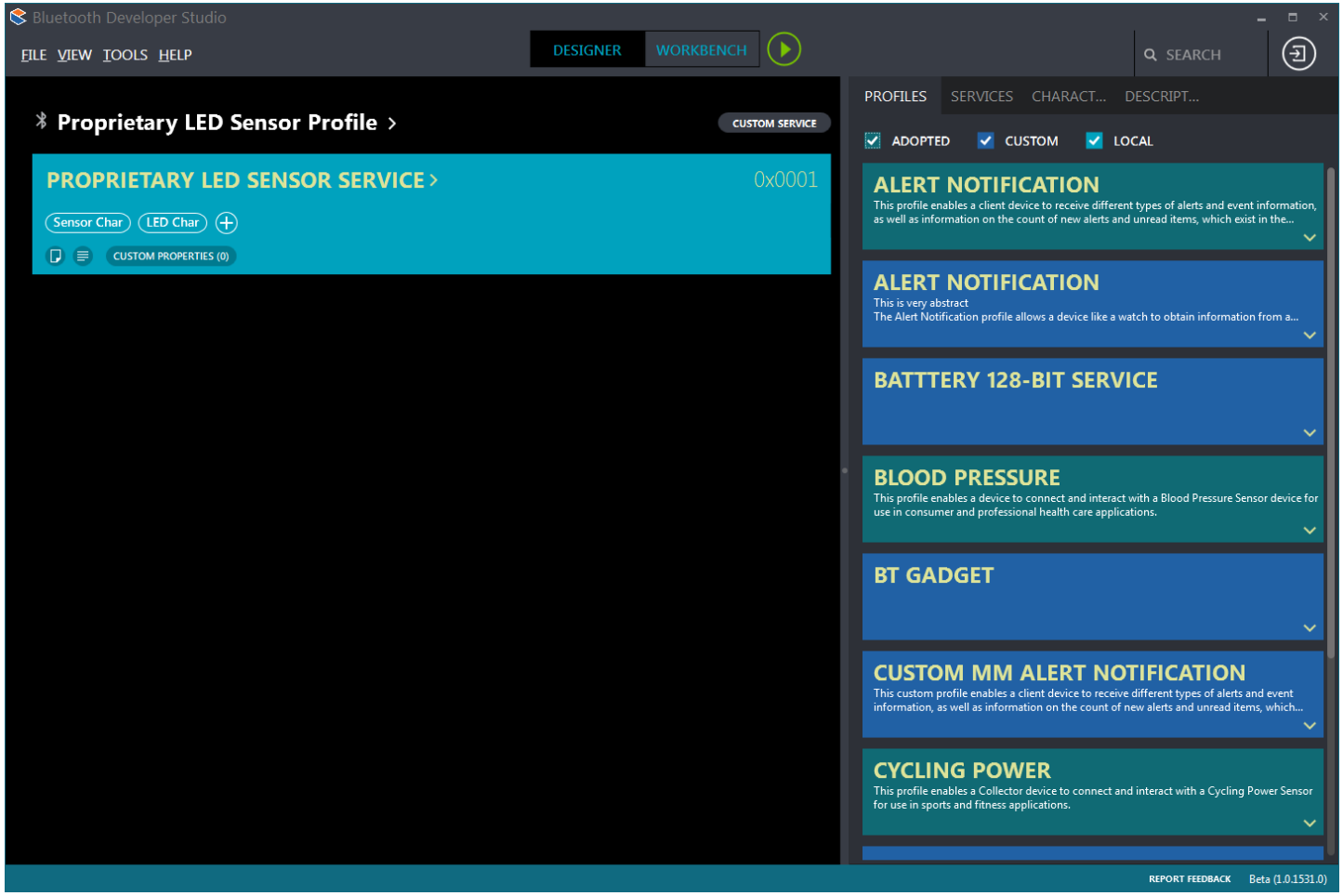
Demo



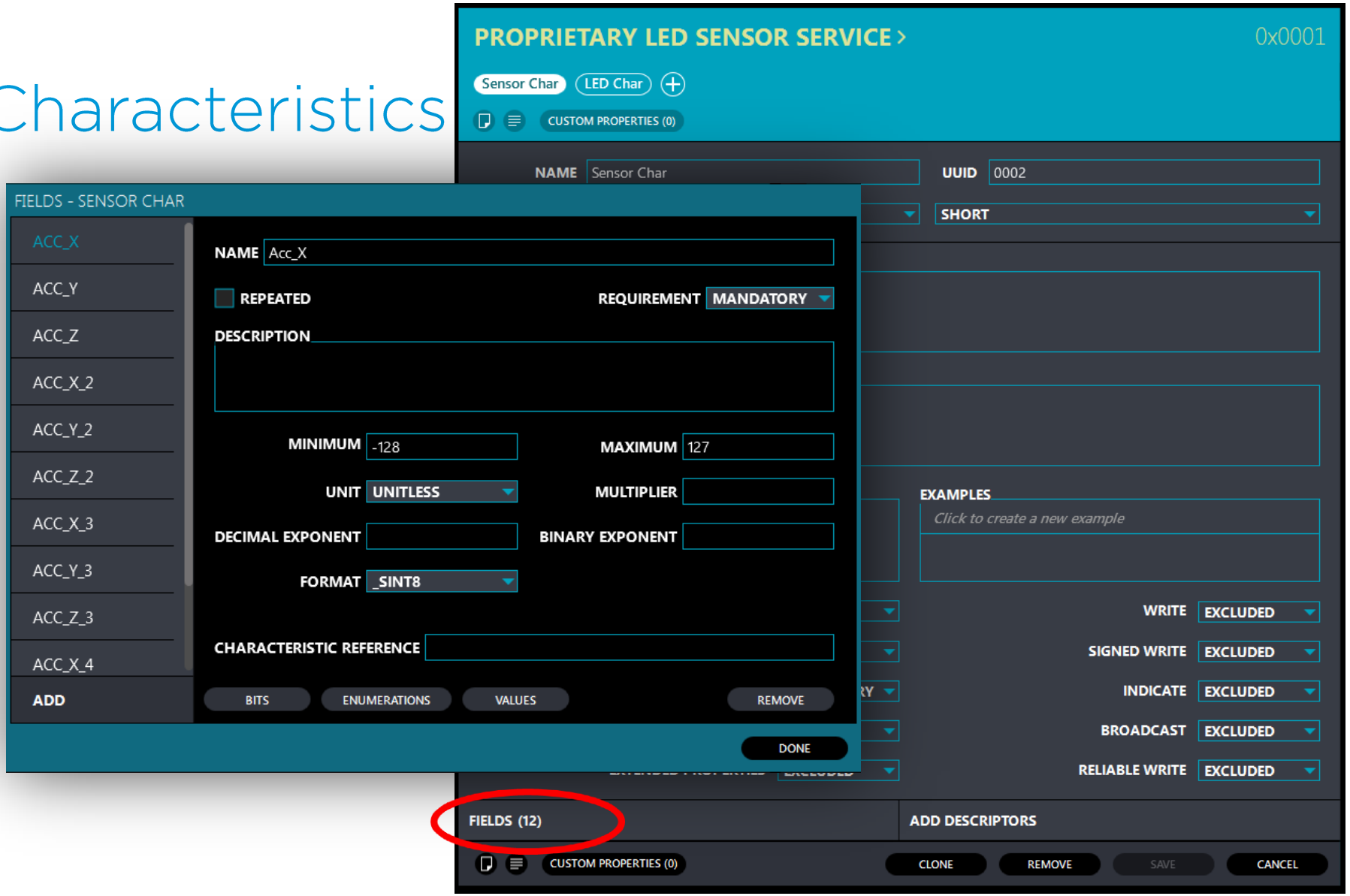




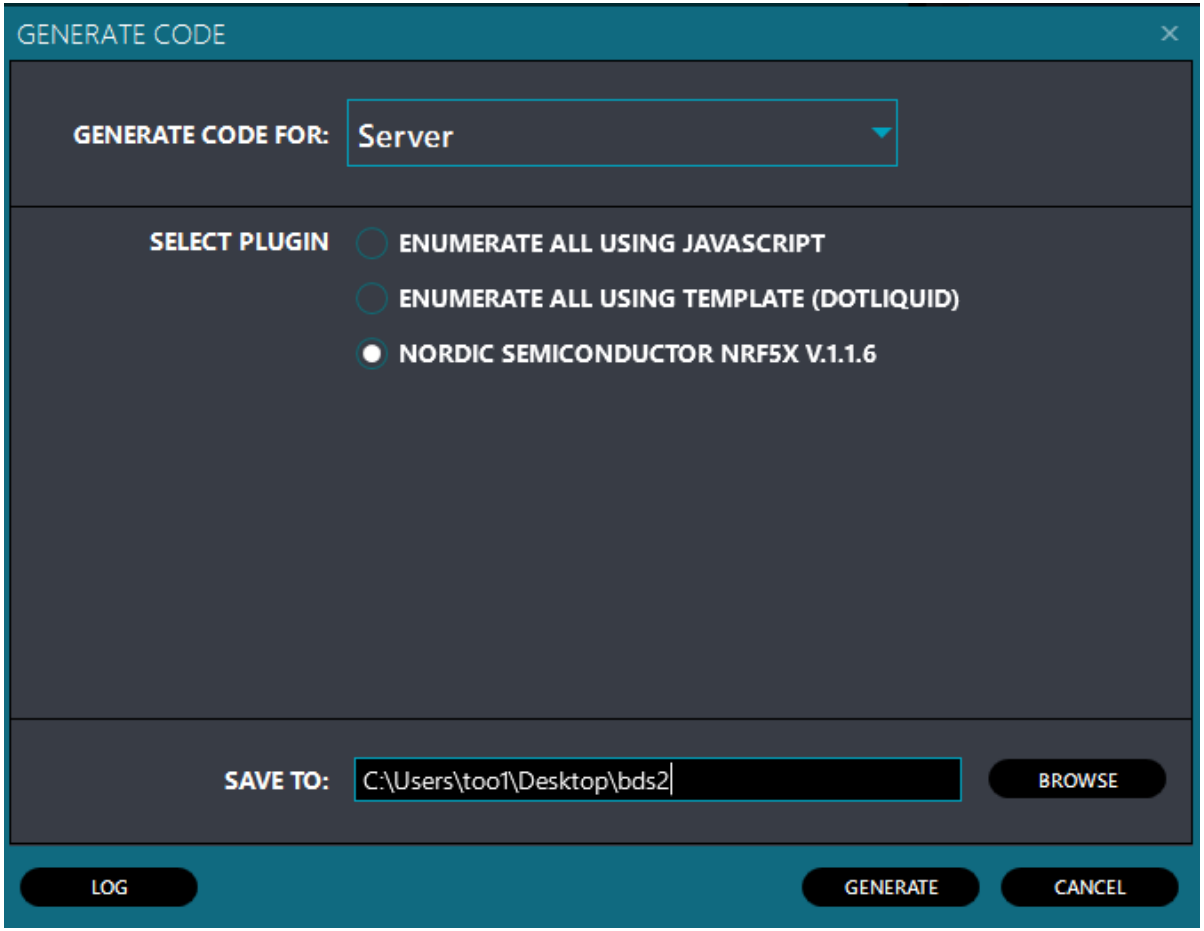
Bluetooth Developer Studio

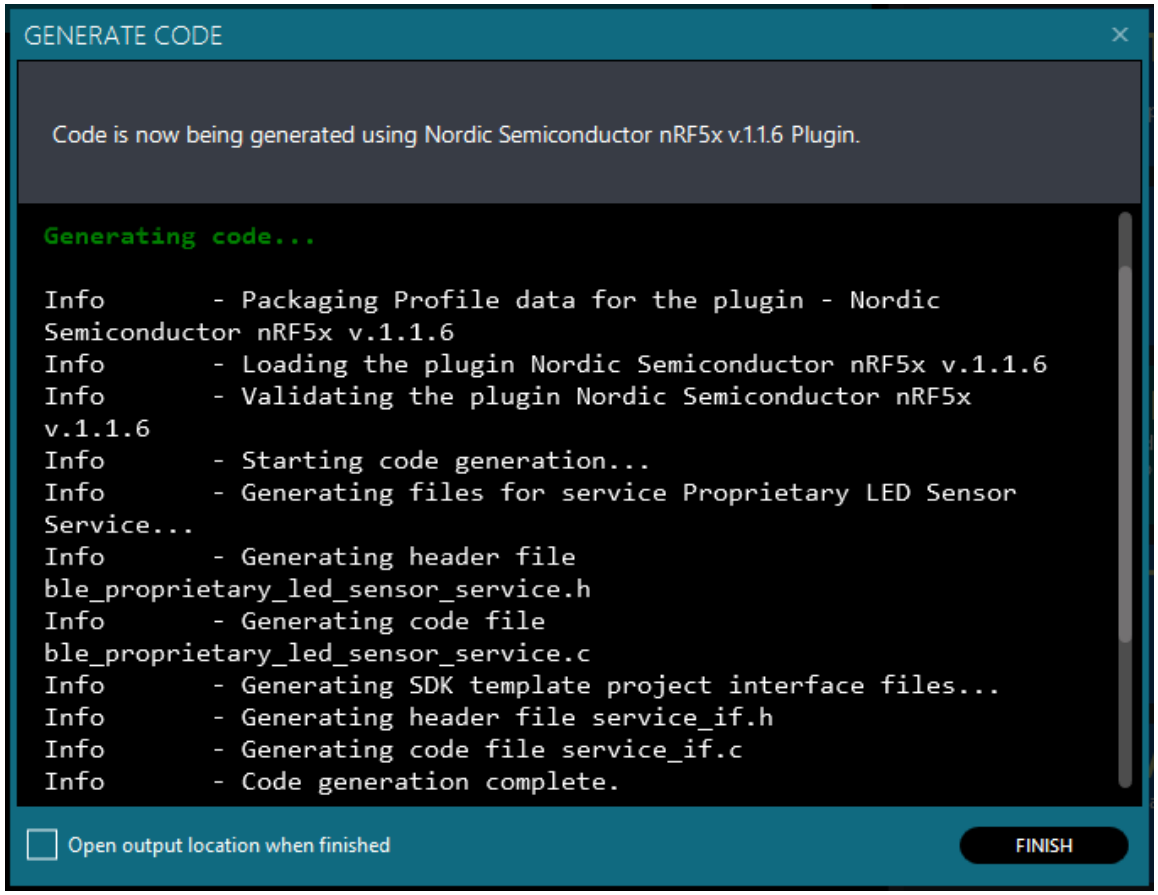


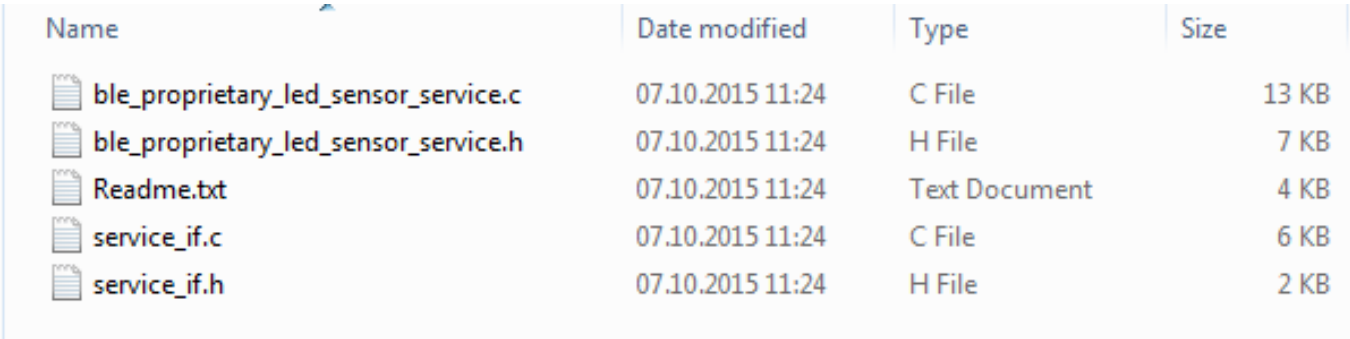
Characteristics



Code generation




GENERATE CODE
Code is now being generated using Nordic Semiconductor nRF5x v.1.1.6 Plugin.
Generating code...
Info - Packaging Profile data for the plugin - Nordic Semiconductor nRF5x v.1.1.6
Info - Loading the plugin Nordic Semiconductor nRF5x v.1.1.6
Info - Validating the plugin Nordic Semiconductor nRF5x v.1.1.6
Info - Starting code generation...
Info - Generating files for service Proprietary LED Sensor Service...
Info - Generating header file ble_proprietary_led_sensor_service.h
Info - Generating code file ble_proprietary_led_sensor_service.c
Info - Generating SDK template project interface files...
Info - Generating header file service_if.h
Info - Generating code file service_if.c
Info - Code generation complete.
Open output location when finished
FINISH


Name
Date modified
Type
Size
ble_proprietary_led_sensor_service.c 07.10.2015 11:24 C File 13 KB
ble_proprietary_led_sensor_service.h 07.10.2015 11:24 H File 7 KB
Readme.txt 07.10.2015 11:24 Text Document 4 KB
service_if.c 07.10.2015 11:24 C File 6 KB
service_if.h 07.10.2015 11:24 H File 2 KB

Merge with SDK

- Download the nRF51 v10 SDK from developer.nordicsemi.com
- Generate your profile in BDS
- Generate code
- Copy generated files to the BDS template project in the SDK
- Add service .c files to the project
- Compile and run

Limitations

- Changes to generated code can not be back ported to BDS
  - Treat generated code as read only
- Supports nRF51 only
  - Generated files still work for the nRF52