BLE Service Discovery with Zephyr

Insights and Pitfalls

Florian Limberger

FOSDEM 2025 Brussels

inovex

Florian Limberger

Senior Embedded Systems Engineer

- Embedded Linux
- Real-time systems
- Zephyr Neophyte

Florian Limberger

@flimberger

inovex

Agenda

- BLE Basics
  - Protocol Stack
  - GATT Protocol
- BLE in Zephyr
  - API
  - Tools

inovex

Terminology Clarification

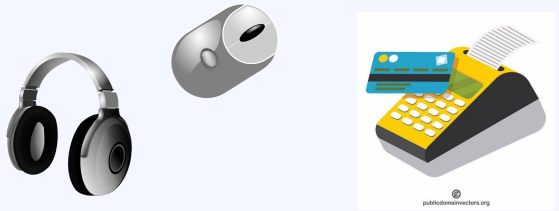
Bluetooth does not use *Host* and *Client* terminology

- *Host → Central Device*
- *Client → Peripheral Device*

inovex

Bluetooth Low Energy: General Terminology

- *Central Device*
  - Controls the connection
  - (Comparatively) powerful device
  - Less resource constrained
  - Mobile phones, Laptops, …
- *Peripheral Device*
  - resource constrained
  - potentially low-cost, high-volume



Images: https://publicdomainvectors.org/

inovex

How does BLE work?

The spec is huge: *Core v5.4* alone has 3112 pages!

Protocol Stack:

- L2CAP (Logical Link Control and Adaptation Layer Protocol)
- HCI (Host Controller Interface)
- GAP (Generic Access Profile)
- ATT (Attribute Protocol)
- GATT (Generic Attribute Protocol)

Focus on the Generic Attribute (GATT) protocol:

- “Application Layer”
- Usual abstraction layer for application developer
- Peripheral Device is usually the *server*
- Central Device is usually the *client*

The Bluetooth SIG *loves* UUIDs

- UUIDs are used everywhere
  - Type information
  - Object identity
  - ...
- 128-bit are kind of big

Solution: Objects defined in the standard use a 16-bit or 32-bit portion to represent a full UUID

`xxxxxxxx-0000-1000-8000-00805F9B34FB`

inovex

ATT: Attribute Protocol

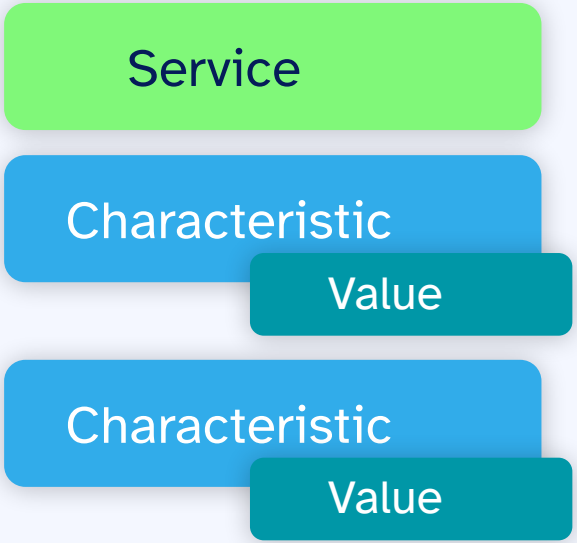
- Client-server protocol
- Provides access to values via 16-bit *handle*
- Attribute data:

- Basic operations
  - Read: client receives a value
  - Write: client sends a value
  - Notify: client instructs server to send updates
  - Indicate: client instructs server to send updates, and requires each update to be acknowledged by the client

inovex

GATT: Generic Attribute Protocol

- Services
  - Identified by UUID
  - Protected by permissions
  - Potentially multiple characteristics providing data
- Characteristics
  - “Container” for values
  - Identified by UUID
  - Metadata: Client Characteristic Configuration Descriptor (CCCD)



inovex

GATT Service Discovery

- Built on ATT attributes
  - Services and Characteristics are stored as big list of attributes
- Data is accessed via ATT operations

A mapping between ATT handles and GATT UUIDs is required

- Mapping may differ between devices or sessions
- Mapping may be cached for efficiency
  - Not implemented by Zephyr

inovex

BLE Central Devices with Zephyr

Just enable the BLE Central module in `prj.conf`:

```
CONFIG_BT=y
CONFIG_BT_CENTRAL=y
CONFIG_BT_GATT_CLIENT=y
```

That’s it, we’re ready to go!

inovex

General Structure: Startup

```
1  int main(void) {
2          bt_enable(NULL); /* synchronous for simplicity */
3          bt_le_scan_start(BT_LE_SCAN_PASSIVE, on_device_scanned);
4          k_sleep(K_FOREVER);
5  }
6
7  void on_device_scanned(const bt_le_addr_t *addr, /* omitted */) {
8          bt_le_scan_stop();
9          struct bt_conn *conn;
10         bt_conn_le_create(addr, BT_CONN_LE_CREATE_CONN,
11                           BT_LE_CONN_PARAM_DEFAULT, &conn);
12 }
```


inovex

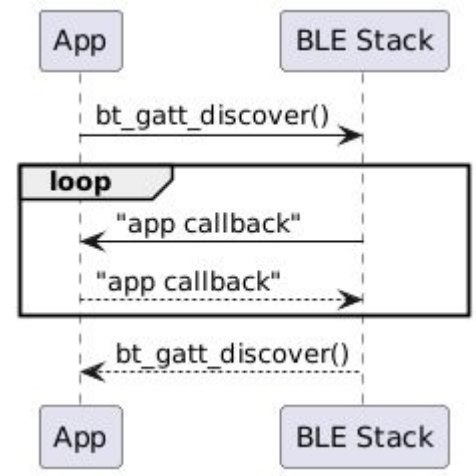
General Structure: Callbacks

```
1  BT_CONN_CB_DEFINE(conn_cbs) = {
2          .connected = on_connected,
3          .disconnected = on_disconnected,
4  };
5
6  void on_connected(struct bt_conn *conn, uint8_t err) {
7          /* start service discovery */
8  }
9
10 void on_connected(struct bt_conn *conn, uint8_t err) {
11         /* clean up */
12 }
```


inovex

Service Discovery in Zephyr

- Set up parameters and callback
- Call the Zephyr BLE stack
- Callback is called from the BLE RX thread for matching conditions



While it sound simple and straightforward, the samples are lacking in this regard.

inovex

The Zephyr GATT API

```
1  int bt_gatt_discover(struct bt_conn *conn,
2                       struct bt_gatt_discover_params *params);
3
4  struct bt_gatt_discover_params {
5          const struct bt_uuid *uuid;
6          bt_gatt_discover_func_t func;
7          union {
8                  struct { /* 3 fields omitted */ } _included;
9                  uint16_t start_handle;
10         };
11         uint16_t end_handle;
12         uint8_t type;
13         /* 2 more omitted fields */
14 };
```


inovex

The Zephyr GATT API (cont'd)

```
typedef uint8_t (*bt_gatt_discover_func_t)(struct bt_conn *conn,
                                           struct bt_gatt_attr *attr,
                                           struct bt_gatt_discover_params
                                               *params);

struct bt_gatt_attr {
        const struct bt_uuid *uuid;
        bt_gatt_attr_read_func_t read;
        bt_gatt_attr_write_func_t write;
        void *user_data;
        uint16_t handle;
        uint16_t perm;
};
```


inovex

Apparently linear code may be actually asynchronous

```c
int discover_primary_services(struct bt_conn *conn) {
        /* set up svc_discover_params */
        return bt_gatt_discover(conn,
            &svc_discover_params);
}

int discover_characteristics(struct bt_conn *conn) {
        /* set up chrc_discover_params */
        return bt_gatt_discover(conn,
            &chrc_discover_params);
}

void discover_services(struct bt_conn *conn) {
        if (!discover_primary_services(conn)) {
        discover_characteristics(conn);
        }
}
```

```
<inf> central_main: connecting to peripheral...
<inf> central_main: device connected: 0
<inf> service_discovery: starting service discovery...
<inf> service_discovery: discovering primary services...
<inf> service_discovery: primary service discovery done
<inf> service_discovery: discovering characteristics...
<inf> service_discovery: characteristic discovery done
<inf> service_discovery: service discovery done
<inf> service_discovery: [SVC] discovering attribute handle 1
<inf> service_discovery: discovered uninteresting service 1801
<inf> service_discovery: [SVC] discovering attribute handle 9
<inf> service_discovery: discovered uninteresting service 1800
<inf> service_discovery: [CHRC] discovering attribute handle 2
<inf> service_discovery: discovered uninteresting service 2a05
<inf> service_discovery: [CHRC] discovering attribute handle 5
<inf> service_discovery: discovered uninteresting service 2b29
<inf> service_discovery: [SVC] discovering attribute handle 16
<inf> service_discovery: discovered LED service
<inf> service_discovery: [CHRC] discovering attribute handle 7
<inf> service_discovery: discovered uninteresting service 2b2a
<inf> service_discovery: [CHRC] discovering attribute handle 10
<inf> service_discovery: discovered uninteresting service 2a00
<inf> service_discovery: [SVC] discovering attribute handle 19
<inf> service_discovery: discovered uptime service
<inf> service_discovery: [CHRC] discovering attribute handle 12
<inf> service_discovery: discovered uninteresting service 2a01
<inf> service_discovery: [CHRC] discovering attribute handle 14
<inf> service_discovery: discovered uninteresting service 2a04
<inf> service_discovery: primary service discovery completed
<inf> service_discovery: [CHRC] discovering attribute handle 17
<inf> service_discovery: discovered LED characteristic
<inf> service_discovery: [CHRC] discovering attribute handle 20
<inf> service_discovery: discovered uptime characteristic
<inf> service_discovery: characteristic discovery completed
<inf> central_main: services discovered
<inf> service_user: uptime: 3018612
```

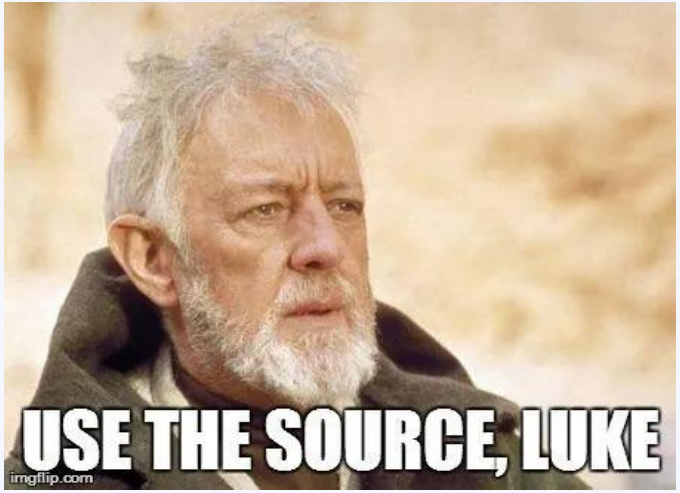

inovex

Pitfalls with BLE Central Devices in Zephyr

- Only API documentation and example code, no actual explanations
- Examples are few and rather simplistic
  - no multi-service discovery
  - no optional services
  - no interdependencies
- The Bluetooth subsystem is big and complex
  - Most of the Bluetooth APIs are asynchronous by default
  - Multithreading (e.g. callbacks called from RX thread)
  - Call-limitations, e.g. cannot call a function from interrupt, like a timer

Hints for being productive

- Use the the source, Luke!
  - The code describes what will happen
  - Doxygen rendering is suboptimal
- Use an event loop for interaction with other components
  - Simplifies concurrency issues
  - Prevents problems from call restrictions

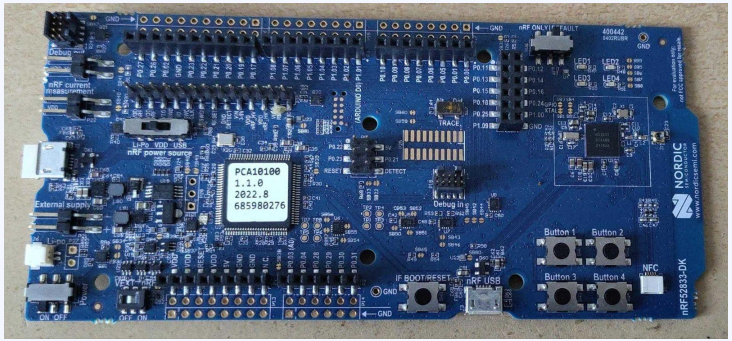


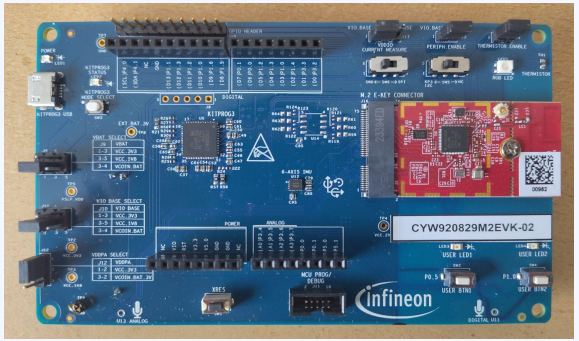
inovex

Portability

Demo code works out of the box without board-specific application code on:

- Nordic Semiconductor nRF52833 DK
- Infineon Technologies AIROC CYW920829M2EVK





Disclaimer: Tested only simple applications without power saving features

inovex

Tools

- Emulator support:
  - `qemu_x86` target
    - Needs the `btproxy` utility
    - Run with `west build -b qemu_x86 -t run`
  - native simulator (`native_sim`)
    - Run with `sudo ./build/zephyr/zephyr --bt-dev=hci0`

More information:
https://docs.zephyrproject.org/latest/connectivity/bluetooth/bluetooth-tools.html#running-on-qemu-or-native-sim

inovex

Where to go from here?

- Service Discovery Demo Project (contains a Central Device): https://github/com/inovex/talk-zephyr-ble-service-discovery
- Nordic Semiconductor has a great introduction: https://academy.nordicsemi.com/courses/bluetooth-low-energy-fundamentals/
- Check out the specification:
  - https://www.bluetooth.com/specifications/specs/
  - https://www.bluetooth.com/specifications/assigned-numbers/
- Tinker with BLE devices from your phone: https://www.nordicsemi.com/Products/Development-tools/nrf-connect-for-mobile

Thank you!

Visit the Zephyr table (Building K, Level 1) to chat!

Join us on Discord: https://chat.zephyrproject.org/

Zephyr®

Florian Limberger

Senior Embedded Engineer

florian.limberger@inovex.de

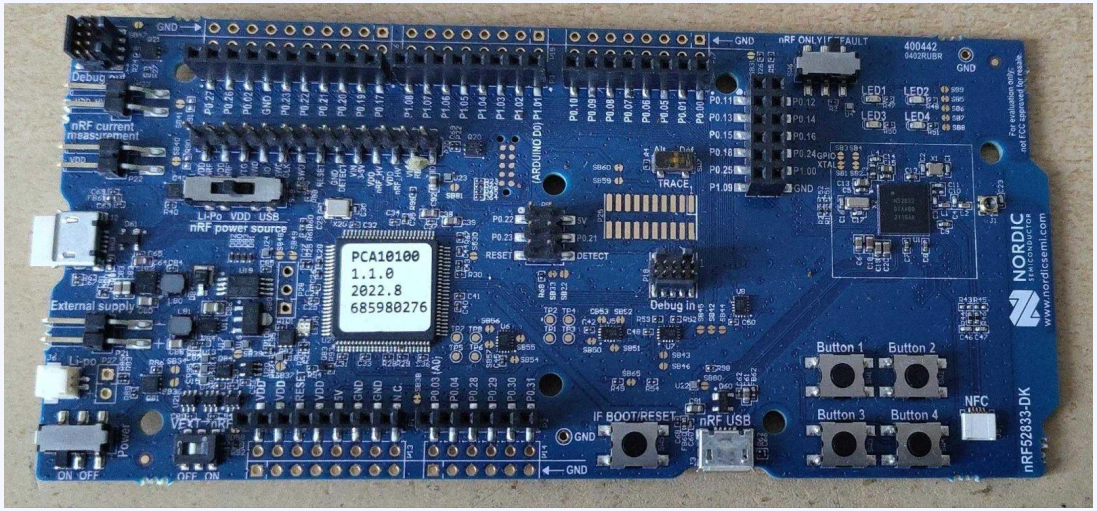
inovex is an IT project center driven by innovation and quality, focusing its services on ‘Digital Transformation’.

- founded in 1999
- 500+ employees
- 8 offices across Germany

www.inovex.de

inovex

Motivation



Recent project: Implement the Proximity Profile on nRF52833/nRF52840

- Link Loss Service
- optional, either both or none
  - Immediate Alert Service
  - Tx Power Service
- Battery Service

inovex

Using the Zephyr GATT API

```
1  static struct bt_gatt_discover_params global_params;
2  static uint16_t global_handle;
3
4  int discover(struct bt_conn *conn) {
5
6          global_params.uuid = MY_SERVICE_UUID;
7          global_params.func = my_discover_func;
8          global_params.start_handle = BT_ATT_FIRST_ATTRIBUTE_HANDLE;
9          global_params.end_handle = BT_ATT_LAST_ATTRIBUTE_HANDLE;
10         global_params.type = BT_GATT_DISCOVER_PRIMARY;
11
12         return bt_gatt_discover(conn, &global_params);
13 };
```

All example code is available on GitHub

inovex

Using the Zephyr GATT API (cont'd)

```
uint8_t my_discover_func(struct bt_conn *conn, struct bt_gatt_attr *attr,
                                        struct bt_gatt_discover_params *params) {

        if (attr == NULL) { return BT_GATT_ITER_STOP; } /* done */

        if (!bt_uuid_cmp(&global_params.uuid, MY_SERVICE_UUID) {

                global_params.uuid = MY_CHARACTERISTIC_UUID;
                global_params.start_handle = attr->handle + 1;
               global_params.type = BT_GATT_DISCOVER_CHARACTERISTIC;
               bt_gatt_discover(conn, &global_params);

        } else if (!bt_uuid_cmp(&global_params.uuid, MY_CHARACTERISTIC_UUID) {
                global_handle = bt_gatt_attr_value_handle(attr);
        }
        return BT_GATT_ITER_STOP;
};
```

All example code is available on GitHub

inovex

Handling Multiple Services Gracefully

```
static struct bt_gatt_discover_params global_params;
static uint16_t global_handle;

int discover(struct bt_conn *conn) {

-       global_params.uuid = MY_SERVICE_UUID;
        global_params.func = my_discover_func;
        global_params.start_handle = BT_ATT_FIRST_ATTRIBUTE_HANDLE;
        global_params.end_handle = BT_ATT_FIRST_ATTRIBUTE_HANDLE;
        global_params.type = BT_GATT_DISCOVER_PRIMARY;

        return bt_gatt_discover(conn, &global_params);
};
```

All example code is available on GitHub

inovex

Handling Multiple Services Gracefully (cont'd)

```
1  uint8_t my_discover_func(struct bt_conn *conn, struct bt_gatt_attr *attr,
2                                   struct bt_gatt_discover_params *params) {
3          if (attr == NULL) { return BT_GATT_ITER_STOP; } /* done */
4                  uint8_t ret = BT_GATT_ITER_CONTINUE;
5
6          if (params->type == BT_GATT_DISCOVER_PRIMARY) {
7                  struct bt_gatt_service_val *svc = (struct bt_gatt_service_val
*)attr->user_data;
8                  if (!bt_uuid_cmp(svc->uuid, MY_SERVICE_UUID) {
9                  ret = BT_GATT_ITER_STOP;
10         } else if (!bt_uuid_cmp(svc->uuid, OTHER_SERVICE_UUID) { /* … */ }
11         if (ret == BT_GATT_ITER_STOP) {
12                 global_params.start_handle = attr->handle + 1;
13                 global_params.end_handle = svc->end_handle;
14                 global_params.type = BT_GATT_DISCOVER_CHARACTERISTIC;
15         }
```

All example code is available on GitHub

inovex

Handling Multiple Services Gracefully (cont'd)

```
        } else if (params->type == BT_GATT_DISCOVER_CHARACTERISTIC) {
                struct bt_gatt_chrc *chrc = (struct bt_gatt_chrc *)attr->user_data;
                if (!bt_uuid_cmp(chrc->uuid, MY_CHARACTERISTIC_UUID) {
                        global_handle = bt_gatt_attr_value_handle(attr);
                        ret = BT_GATT_ITER_STOP;
                } else if (!bt_uuid_cmp(chrc->uuid, OTHER_CHARACTERISTIC_UUID) { /* ... */ }
                if (ret == BT_GATT_ITER_STOP) {
                        global_params.uuid = NULL;
                        global_params.start_handle = global_params.end_handle + 1;
                       global_params.end_handle = BT_ATT_LAST_ATTRIBUTE_HANDLE;
             global_params.type = BT_GATT_DISCOVER_PRIMARY;
                  }
        }

        if (ret == BT_GATT_ITER_STOP) { bt_gatt_discover(conn, &global_params); }
        return ret;
```

All example code is available on GitHub

inovex

Recap

- Discover known services by their UUID
  - Simple to implement
  - May be slow for multiple services, as the descriptors have to be traversed for each service
  - Unnecessary traversals for missing optional services
- Iterate over all primary services
  - Traverse all attributes only once
  - More complicated code and state tracking

inovex

How did we solve our issues?

A couple of observations:

- We monitor disconnection events already
- We looked only at RSSI, not the Tx Power
- Immediate Alert Service is very simple

Conclusion: Do not implement Proximity Profile, just put everything into our own non-standard service which is discovered simply by UUID ;)

Disclaimer: During creation of the demos for this presentation I learned how to do it properly!