Dialog SDK 5.0.x/6.0.x Tutorial

Usage of GATT Commands in Custom Profile

April 2019

...personal
...portable
...connected

BLE Custom profile

Let's build a demo together …

- **Before we start, we recommend you to …**
  - Install the latest Smartsnippets studio from Dialog customer support website
  - Download the SDK as well
  - Link:
    - https://support.dialog-semiconductor.com/connectivity
  - Require to look at Dialog Tutorial 1 and Tutorial 2

- **Consideration …**
  - All the changes are applicable in both the SDK 5.0.x (DA14580/1/2/3) and SDK 6.0.x (DA14585/6) if it is not mentioned specifically for a particular application

BLE Custom profile

Let's build a demo together …

- **What are you going to learn from this tutorial …**
  - Basic understanding of Generic ATT profile
  - GATT custom profile application message flow
  - Basic understanding of custom database creation process
  - Small assignment to add a characteristic in the custom service database that will be used to change the LED state from on to off or vice versa

Contents

BLE profile

Custom profile service wrt GATT Source code discussion

What would you see as output

BLE profile

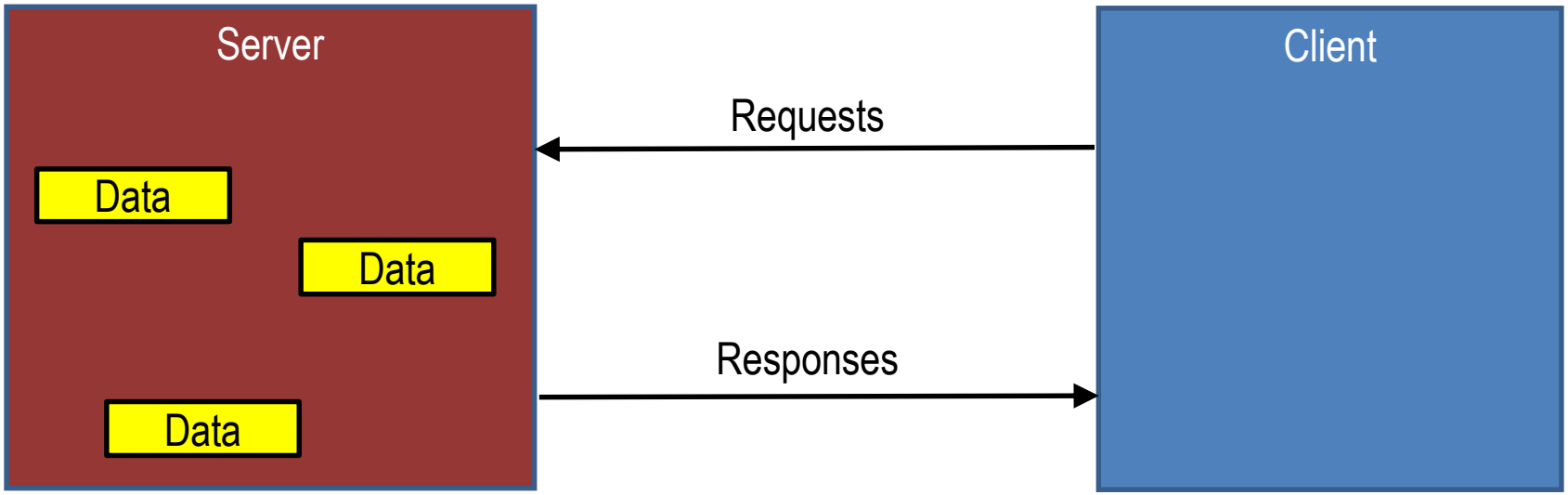
Overview

- Bluetooth Low Power (BLE) profile is a formal definition of the behaviour of a Bluetooth application which is based on Generic Attribute Profile (GATT).
- BLE profile follows a **structured approach** to help a device (**server/peripheral**) to expose information to other devices (**client/central**) about its capabilities and how to access its information.
- **The server** is the owner of the data and in most cases is the peripheral device.
- **The client** is the consumer of the data and is typically the central device (Smart phone/tab).
- **https://developer.bluetooth.org/gatt/services/Pages/ServicesHome.aspx**

BLE profile

Overview

- Client Server Architecture
  - Servers have data, this is known as the **peripheral** in GAP Protocol
  - Clients request data to/from servers, this is known as **central** in GAP
- Servers expose data using Attributes



BLE profile

Overview

- A BLE **Profile** can have one or more **services**.
- **Services** are used to break data into logic entities and contain specific chunks of data called **characteristics**.
- A service can have one or more **characteristics**, and each service distinguishes itself from other services by means of a unique numeric ID called a **UUID**, which can be either 16-bit (for officially adopted BLE Services) or **128-bit (for custom services)**.
- A **characteristic** is the lowest level concept in GATT transactions, which contains a single data point.
- Similarly to services, each characteristic **distinguishes** itself via a pre-defined **16-bit or 128-bit UUID,** and you're free to use the SIG standard characteristics (which ensures interoperability across and BLE-enabled HW/SW) or define your own custom characteristics which only your peripheral and SW understands.

Contents

Custom profile service and source code discussion

What would you see as output

Custom service

Custom service profile example

- **This example demonstrates:**
  - 128 bit UUID custom service implementation
  - How to access custom profile database
  - This training covers a step by step procedure of creating a characteristic, advertise the new characteristic, send and receive GATT CMD between Central and Peripheral devices.

- **Software you need:**
  - Dialog Smartsnippets studio
  - Dialog SDK
  - Project location:
    - ..\projects\target_apps\ble_examples\ble_app_peripheral

Custom service

target_apps\ble_examples\ble_app_peripheral project covers

- Check **custom profile database** access.
- Check the **advertising device name**.
- Use the device information service (**DISS**).
- Inspect the Custom service user defined characteristic.
- Examples of creating user defined characteristics.

Custom service

Custom service profile basic message flow

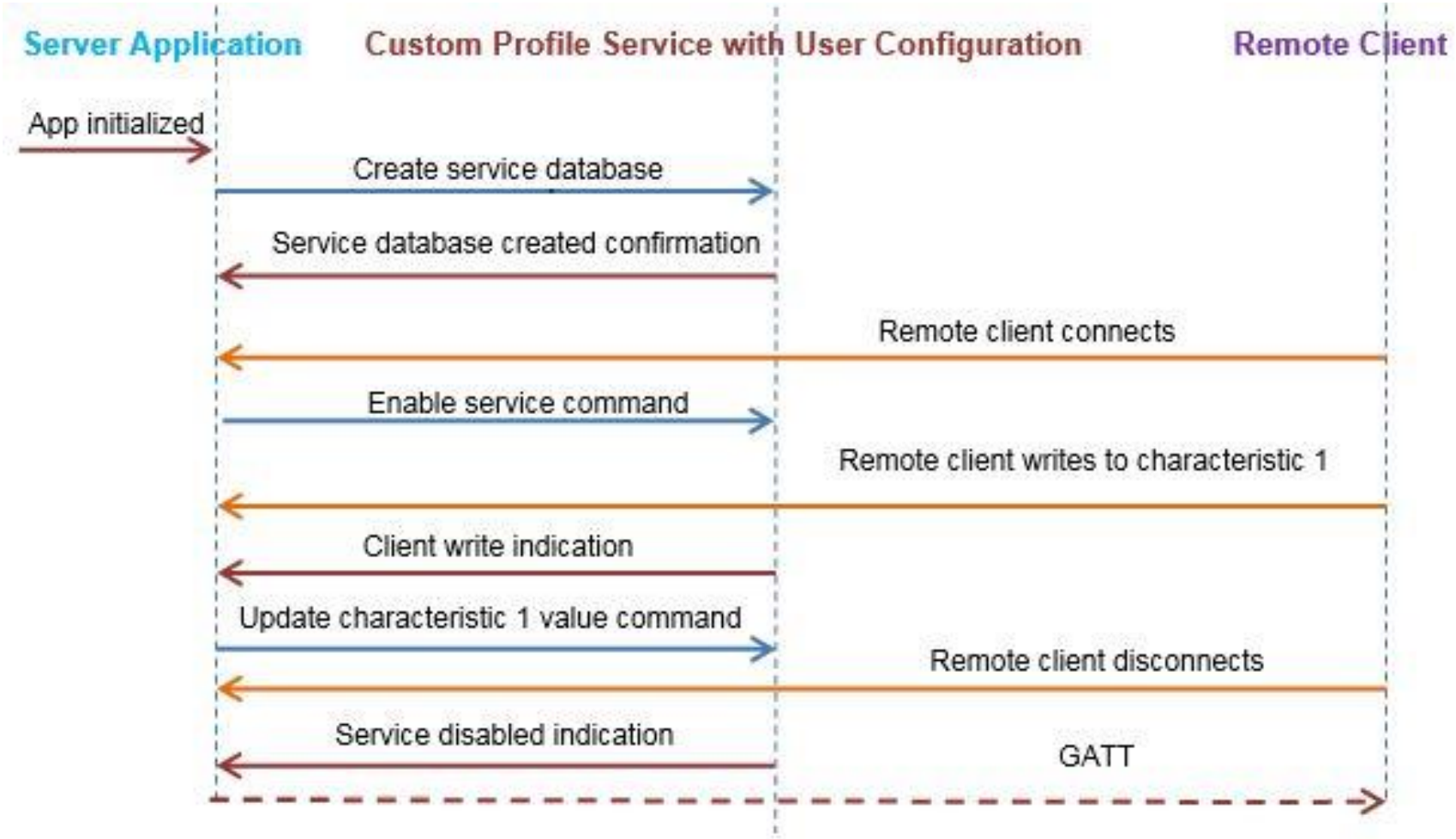
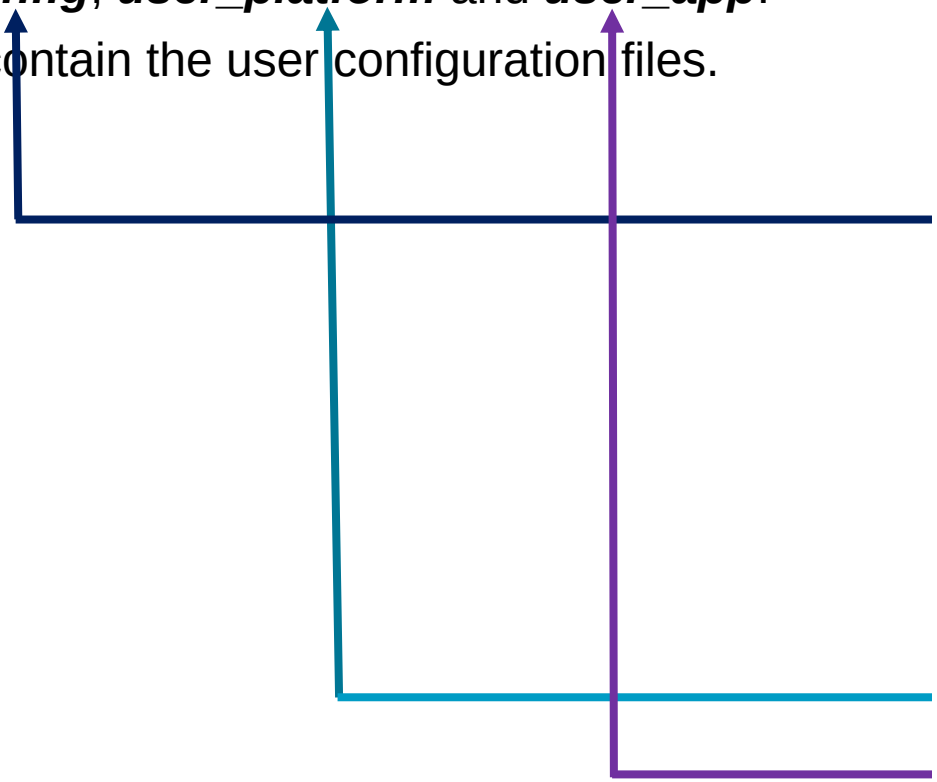


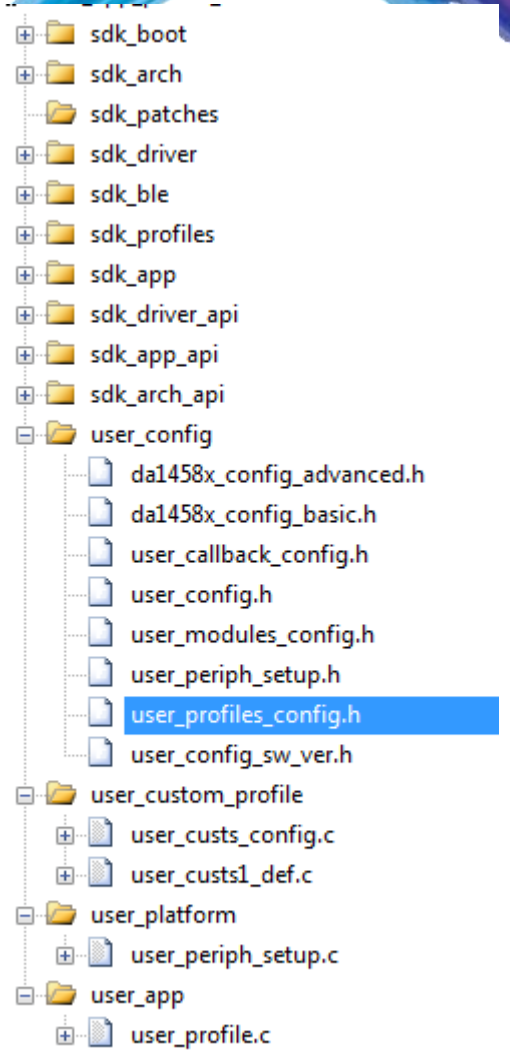
Figure: Message flow diagram

Custom service

ble_app_peripheral.uvprojx project layout

- Group ***user_config***, ***user_platform*** and ***user_app***.
- These groups contain the user configuration files.





Custom service

Description of some important files

```
/* Holds DA1458x basic configuration settings.    */
da1458x_config_basic.h

/* Holds DA1458x advanced configuration settings. */
da1458x_config_advanced.h

/* Holds user specific information about software version. */
user_config_sw_ver.h

/* Defines which application modules are included or excluded from the user's application. */
user_modules_config.h

            /* The Device information application profile is excluded. */
            #define EXCLUDE_DLG_PROXR         (1)
            /* The Device information application profile is included. */
            #define EXCLUDE_DLG_CUSTS1        (0)

            /* Note:                                                                    */
            /*      This setting has no effect if the respective module is a BLE Profile */
            /*      that is not used in the user's application.                         */


/* Callback functions that handle various events or operations. */
user_callback_config.h


/* Holds advertising parameters, connection parameters, etc. */
user_config.h
```

Custom service

Description of some important files

```
/* Defines which BLE profiles (Bluetooth SIG adopted or custom ones) will be included in user's application.
   each header file denotes the respective BLE profile*/
user_profiles_config.h

            #inlucde "diss.h"       // Includes Device Information Service.
            #include "custs1.h"     // Includes Custom service.
```

Note: SDK6 has provided a robust interface so the above implementation is done by MACRO flags

```
            #define CFG_PRF_DISS
            #define CFG_PRF_CUST1
```

```
/* Defines the structure of the Custom profile database structure and
   cust_prf_funcs[] array, which contains the Custom profile API functions calls.*/
user_custs_config.h
```

Note: SDK6 uses the following file for the same purpose

```
user_custs_config.c
```

```
/* Holds hardware related settings relative to the used Development Kit. */
user_periph_setup.h
```

```
/* Source code file that handles peripheral (GPIO, UART, SPI, etc.)
   configuration and initialization relative to the Development Kit.*/
user_periph_setup.c
```

Custom service

Adding a characteristic step by step

TODO 1 - Change the default **BD_ADDRESS, this address has to be unique in a BLE network.**

/* @file **da1458x_config_advanced.h** */

```
/* copy and paste in code step 1 change the BLE device address */
#define CFG_NVDS_TAG_BD_ADDRESS                {0x01, 0x01, 0x01, 0x01, 0x01, 0x01}
```

TODO 2 – Check and define **DLG_CUST1** module in your application code

/* @file **user_modules_config.h** */

```
#define EXCLUDE_DLG_SPOTAR          (1)          /* excluded */
/* copy and paste in code step 2 define DLG_CUST1 module in your application code */
#define EXCLUDE_DLG_CUSTS1          (0)          /* included */
```

TODO 3 – Check and include **cust1.h** in your application code to activate custom profile

/* @file **user_profiles_config.h** */

```
#include "diss.h"
/* copy and paste in code step 3 add custs1.h NOTE: For SDK6 check the MACRO flags mentioned in slide 14 */
#include "custs1.h"
```

Custom service

Adding a characteristic step by step

TODO 4 - Information and change your advertising device name

/* @file **user_config.h** */

```
/* default sleep mode. Possible values ARCH_SLEEP_OFF, ARCH_EXT_SLEEP_ON, ARCH_DEEP_SLEEP_ON
   ARCH_EXT_SLEEP_ON, ARCH_DEEP_SLEEP_ON - You cannot debug in these modes
*/
const static sleep_state_t app_default_sleep_mode = ARCH_SLEEP_OFF;
//-------------NON-CONNECTABLE & UNDIRECTED ADVERTISE RELATED COMMON -- //
/// Advertising service data
/// dev step 5 explanation of the following 3 items

#define USER_ADVERTISE_DATA ("\x03"\
            ADV_TYPE_COMPLETE_LIST_16BIT_SERVICE_IDS\
            ADV_UUID_DEVICE_INFORMATION_SERVICE\
            "\x11"\                                  /// The next section takes hex x11 = decimal 17 bytes
            ADV_TYPE_COMPLETE_LIST_128BIT_SERVICE_IDS\   /// Shows complete list of 128 bit Service IDs
            "\x2F\x2A\x93\xA6\xBD\xD8\x41\x52\xAC\x0B\x10\x99\x2E\xC6\xFE\xED") /// Your Custom Service UUID
/// Note- Custom service UUID is shown from right to left <-- EDFEC6...2F in the client LightBlue iOS app GUI
/* copy and paste in code step 4 change your advertising device name */
#define USER_DEVICE_NAME    ("B-CUST1")
```

Custom service

Adding a characteristic step by step

TODO 5 – Overview of existing BLE Profile custom service characteristic values and properties

NAME	PROPERTIES	LENGTH	DESCRIPTION
Control Point	WRITE	1	Accept commands from peer
LED State	WRITE NO RESPONSE	1	Toggles a LED connected to a GPIO
ADC Value 1	READ, NOTIFY	2	Reads sample from an ADC channel
ADC Value 2	READ	2	Reads sample from an ADC channel
Button State	READ, NOTIFY	1	Reads the current state of a push button connected a GPIO
Indicate able	READ, INDICATE	20	Demonstrate indications
Long Value	READ, WRITE. NOTIFY	50	Demonstrate writes to long characteristic value

Custom service

Adding a characteristic step by step

- Characteristics have ***names***
  - Name that will be displayed on the client scanner application.
- Characteristics have ***values***
  - Array of up to 512 octets, fixed or variable length data mostly in hexadecimal format.
- Characteristics have ***handlers***
  - Used to address an individual attribute by a client, this will be discussed more in Training 3.
- Characteristics have ***description***
  - <<UUID>>, determines what does the value mean
  - Defined by GAP, GATT, or “User defined Custom Characteristic Specifications”
  - Example “`Accept commands from peer`” is a description for Control point characteristic
- Characteristics have ***properties***
  - Read, Write, Notify etc.

Custom service

Adding a characteristic step by step

TODO 6 - Information

```
/* @file user_custs_config.h Note: SDK6 uses user_custs1_def.h */

/* step 5 and step 6 info:: 128 bit Service UUID this is displayed from Right to Left in the client scanner device */
#define DEF_CUST1_SVC_UUID_128 {0x2F, 0x2A, 0x93, 0xA6, 0xBD, 0xD8, 0x41, 0x52, 0xAC, 0x0B, 0x10, 0x99, 0x2E, 0xC6, 0xFE, 0xED} /* Displayed as EDFEC62E99100BAC5241D8BDA6932A2F */
```

TODO 7 - Add your control point

```
/* @file user_custs_config.h Note: SDK6 uses user_custs1_def.h */

#define DEF_CUST1_LONG_VALUE_UUID_128      {0x8C, 0x09, 0xE0, 0xD1, 0x81, 0x54, 0x42, 0x40, 0x8E, 0x4F, 0xD2, 0xB3, 0x77, 0xE3, 0x2A, 0x77}
/* copy and paste in code step 7 define your control point */
#define DEF_USER_LED_STATE_UUID_128 {0x33, 0x32, 0x31, 0x30, 0x29, 0x28, 0x27, 0x26, 0x25, 0x24, 0x23, 0x22, 0x21, 0x20, 0x19, 0x18}
```

- **NOTE 1:** A service can have one or more characteristics, and each service distinguishes itself from other services by means of a unique numeric ID called a UUID, which can be either 16-bit (for officially adopted BLE Services) or 128-bit (for custom services).
- **NOTE 2: This tutorial provides an example of a 128bit UUID number.** Before releasing a product to the market the **user will need to define a different 128bit number than used in the example** to avoid **conflicts.** The user can select any number and this does not need to be registered at the Bluetooth SIG .

Custom service

Adding a characteristic step by step

TODO 8 - Add your control point data length

```
/* @file user_custs_config.h Note: SDK6 uses user_custs1_def.h */
```

```
#define DEF_CUST1_LONG_VALUE_CHAR_LEN       50
/* copy and paste in code step 8 define your control point data length */
#define DEF_USER_LED_STATE_CHAR_LEN  1
```

TODO 9 - Add your characteristic description name as string

```
/* @file user_custs_config.h Note: SDK6 uses user_custs1_def.h */
```

```
#define CUST1_LONG_VALUE_CHAR_USER_DESC     "Long Value"
/* copy and paste in code step 9 define your characteristic description name */
#define USER_LED_STATE_USER_DESC            "Your LED Characteristic"
```

Custom service

Adding a characteristic step by step

TODO 10 - Add your custom1 service database control point characteristic enumeration

```
/* @file user_custs_config.h Note: SDK6 uses user_custs1_def.h */

enum
{
    …
    CUST1_IDX_LONG_VALUE_CHAR,
    CUST1_IDX_LONG_VALUE_VAL,
    CUST1_IDX_LONG_VALUE_NTF_CFG,
    CUST1_IDX_LONG_VALUE_USER_DESC,

    /* copy and paste in code step 10 add your characteristic */
    USER_IDX_LED_STATE_CHAR,
    USER_IDX_LED_STATE_VAL,
    USER_IDX_LED_STATE_USER_DESC,
    CUST1_IDX_NB
};
```

Custom service

Adding a characteristic step by step

TODO 11 - Declare and assign custom server attribute value

```
/* @file user_custs_config.h Note: SDK6 uses user_custs1_def.c */
static uint8_t CUST1_LONG_VALUE_UUID_128[ATT_UUID_128_LEN]       = DEF_CUST1_LONG_VALUE_UUID_128;
/* copy and paste in code step 11 declare and assign custom server attribute value */
static uint8_t USER_LED_STATE_UUID_128[ATT_UUID_128_LEN]         = DEF_USER_LED_STATE_UUID_128;
```

TODO 12 - Add your characteristic description with permission properties, handler and UUID

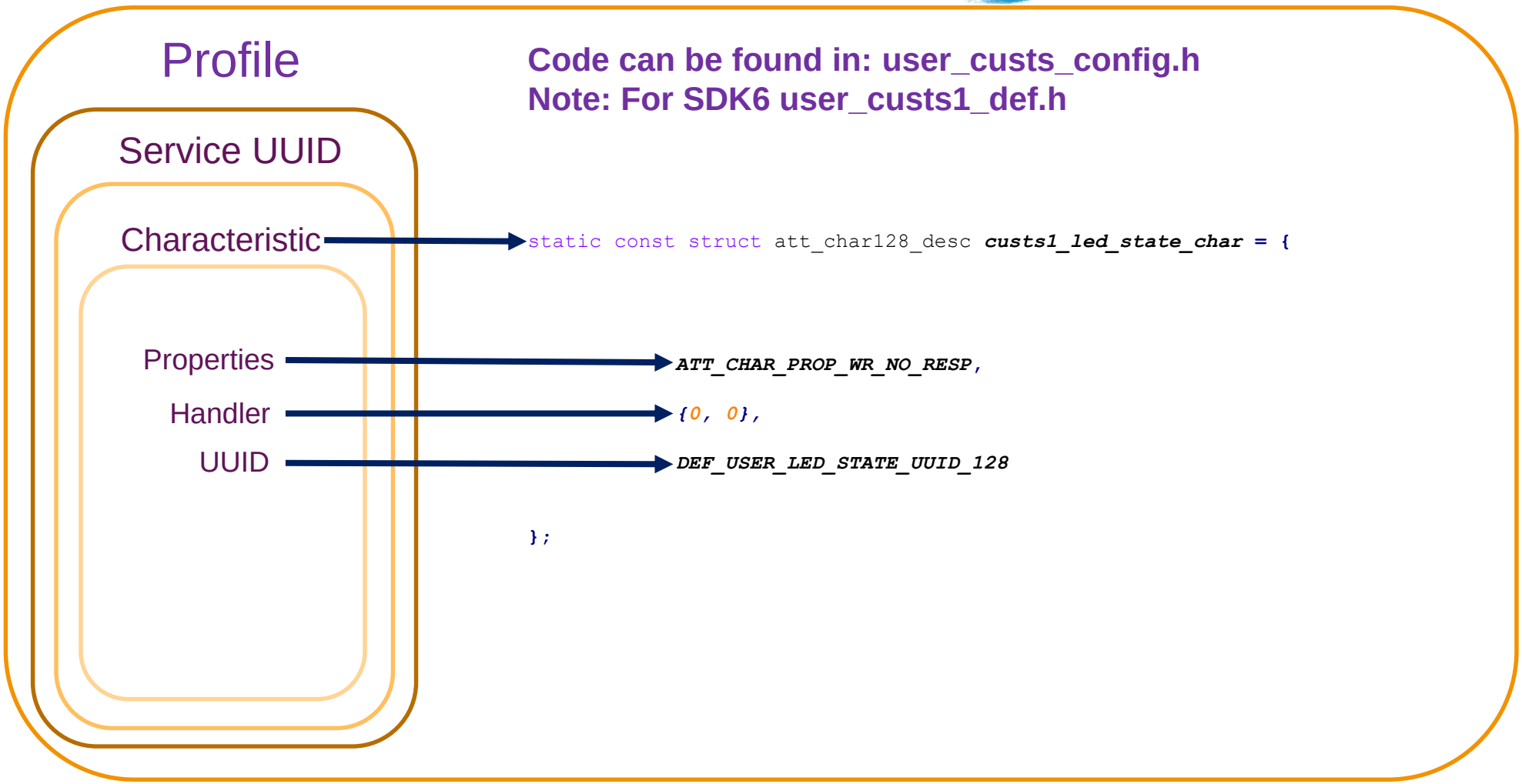
```
/* @file user_custs_config.h Note: SDK6 uses user_custs1_def.c */
static const struct att_char128_desc custs1_long_value_char       = {ATT_CHAR_PROP_RD | ATT_CHAR_PROP_WR |
ATT_CHAR_PROP_NTF,
                                                                    {0, 0},
                                                                    DEF_CUST1_LONG_VALUE_UUID_128};

/* copy and paste in code step 12 */
/* Add your characteristic description with permission properties, handler and UUID */

static const struct att_char128_desc user_led_state_char          = {ATT_CHAR_PROP_WR_NO_RESP,
                                                                    {0, 0},
                                                                    DEF_USER_LED_STATE_UUID_128};
```

Custom service

Adding a characteristic step by step



Code can be found in: user_custs_config.h
Note: For SDK6 user_custs1_def.h

```
static const struct att_char128_desc custs1_led_state_char = {

    ATT_CHAR_PROP_WR_NO_RESP,
    {0, 0},
    DEF_USER_LED_STATE_UUID_128

};
```

Custom service

Adding a characteristic step by step

TODO 13 - Add your characteristic declaration, value and description in custom server database attributes, please go to next slide to copy the code, to large code to fit in one slide

```
/* @file user_custs_config.h Note: SDK6 uses user_custs1_def.c */
```

```
/// Full CUSTOM1 Database Description - Used to add attributes into the database
static const struct attm_desc_128 custs1_att_db[CUST1_IDX_NB] =
{
    ...
    // Long Value Characteristic Declaration
    [CUST1_IDX_LONG_VALUE_CHAR]         = {(uint8_t*)&att_decl_char, ATT_UUID_16_LEN, PERM(RD, ENABLE),
                                           sizeof(custs1_long_value_char), sizeof(custs1_long_value_char),
                                          (uint8_t*)&custs1_long_value_char},
    // Long Value Characteristic Value
    [CUST1_IDX_LONG_VALUE_VAL]          = {CUST1_LONG_VALUE_UUID_128, ATT_UUID_128_LEN, PERM(RD, ENABLE) | PERM(WR,
ENABLE) | PERM(NTF, ENABLE),
                                           DEF_CUST1_LONG_VALUE_CHAR_LEN, 0, NULL},
    // Long Value Client Characteristic Configuration Descriptor
    [CUST1_IDX_LONG_VALUE_NTF_CFG]      = {(uint8_t*)&att_decl_cfg, ATT_UUID_16_LEN, PERM(RD, ENABLE) | PERM(WR,
ENABLE),
                                           sizeof(uint16_t), 0, NULL},
    // Long Value Characteristic User Description
    [CUST1_IDX_LONG_VALUE_USER_DESC]    = { (uint8_t*)&att_decl_user_desc, ATT_UUID_16_LEN, PERM(RD, ENABLE),
                                           sizeof(CUST1_LONG_VALUE_CHAR_USER_DESC) - 1,
sizeof(CUST1_LONG_VALUE_CHAR_USER_DESC) - 1, CUST1_LONG_VALUE_CHAR_USER_DESC},
```

Custom service

Adding a characteristic step by step

TODO 13 - Add your characteristic declaration, value and description in custom server database attributes

```
/* @file user_custs_config.h Note: SDK6 uses user_custs1_def.c */
```

```
/* copy and paste in code step 13 add your characteristic declaration, value and description in database attributes
*/

    // user LED State Characteristic Declaration
    [USER_IDX_LED_STATE_CHAR]       = {(uint8_t*)&att_decl_char, ATT_UUID_16_LEN, PERM(RD, ENABLE),
                                       sizeof(user_led_state_char), sizeof(user_led_state_char), (uint8_t*)&user_led_state_char},

    // user LED State Characteristic Value
    [USER_IDX_LED_STATE_VAL]        = {USER_LED_STATE_UUID_128, ATT_UUID_128_LEN, PERM(WR, ENABLE),
                                       DEF_USER_LED_STATE_CHAR_LEN, 0, NULL},

    // user LED State Characteristic User Description
    [USER_IDX_LED_STATE_USER_DESC]    = {(uint8_t*)&att_decl_user_desc, ATT_UUID_16_LEN, PERM(RD, ENABLE),
                                         sizeof(USER_LED_STATE_USER_DESC) - 1, sizeof(USER_LED_STATE_USER_DESC) - 1,
                                         USER_LED_STATE_USER_DESC},
};
```

Custom service

Adding a GATT command step by step

TODO 14 - Add the following ENUM and GATT command handler declaration in **user_custs1_impl.h** file

/* @file **user_custs1_impl.h** */

```
/* user defined LED state */
enum
{
    LED_OFF = 0,
    LED_ON,
};

/**
 ****************************************************************************************
 * @brief User defined Led state value write indication handler.
 * @param[in] msgid   Id of the message received.
 * @param[in] param   Pointer to the parameters of the message.
 * @param[in] dest_id ID of the receiving task instance.
 * @param[in] src_id  ID of the sending task instance.
 * @return void
 ****************************************************************************************
*/
void user_led_wr_ind_handler(ke_msg_id_t const msgid,
                             struct custs1_val_write_ind const *param,
                             ke_task_id_t const dest_id,
                             ke_task_id_t const src_id);
```

Custom service

Adding a GATT command step by step

TODO 15 - Add the following GATT command handler definition in **user_custs1_impl.c** file

/* @file **user_custs1_impl.c** */

```
/**
 ****************************************************************************************
 * @brief User defined led state value write indication handler.
 * @param[in] msgid   Id of the message received.
 * @param[in] param   Pointer to the parameters of the message.
 * @param[in] dest_id ID of the receiving task instance.
 * @param[in] src_id  ID of the sending task instance.
 * @return void
 ****************************************************************************************
*/
void user_led_wr_ind_handler(ke_msg_id_t const msgid,
                             struct custs1_val_write_ind const *param,
                             ke_task_id_t const dest_id,
                             ke_task_id_t const src_id)
{
    uint8_t led_state = 0;
    memcpy(&led_state, &param->value[0], param->length);

    if (led_state == LED_ON)
        GPIO_SetActive(GPIO_LED_PORT, GPIO_LED_PIN);
    else if (led_state == LED_OFF)
        GPIO_SetInactive(GPIO_LED_PORT, GPIO_LED_PIN);
}
```

Custom service

Adding a GATT command step by step

TODO 16 - Add the following switch case in **user_catch_rest_hndl()** in **user_peripheral.c** file

```
/* @file user_peripheral.c */
```

```
void user_catch_rest_hndl(ke_msg_id_t const msgid,
                          void const *param,
                          ke_task_id_t const dest_id,
                          ke_task_id_t const src_id)
{
    switch(msgid)
    {
        case CUSTS1_VAL_WRITE_IND:
        {
            struct custs1_val_write_ind const *msg_param = (struct custs1_val_write_ind const *)(param);

            switch (msg_param->handle)
            {
                case USER_IDX_LED_STATE_VAL:
                    user_led_wr_ind_handler(msgid, msg_param, dest_id, src_id);
                    break;

                default:
                    break;
            }
        } break;
```

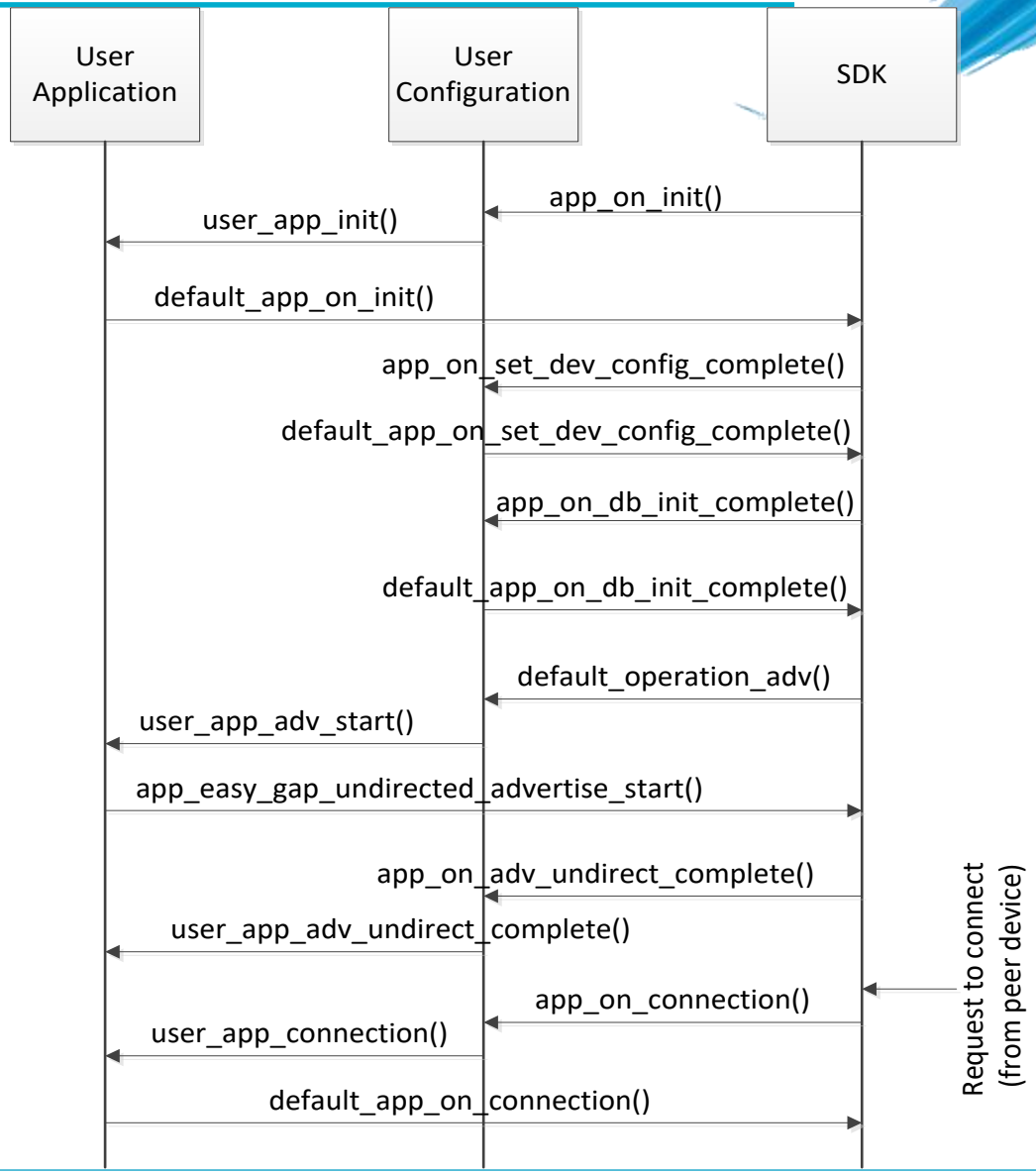
Custom service

How is it working?

- Several events can occur during the lifetime of the BLE application and these events need to be handled in a specific manner.
- The SDK is flexible enough to either call a default handler or call the user's defined event or operation handler to handle specific events (**`user_catch_rest_hndl`**), you really need to understand this API.
- The SDK mechanism, which is provided to the user in order to take care of the above, is the registration of callback functions for every event or operation.
- The C header file *user_callback_config.h*, which resides in user space, contains the registration of the callback functions.

Custom service

Abstract code flow



Custom service

user_callback_config.h important function discussion

```
static const  struct arch_main_loop_callbacks user_app_main_loop_callbacks = {
    .app_on_init            = user_app_init,
    .app_on_ble_powered     = NULL,
    .app_on_sytem_powered   = NULL,
    .app_before_sleep       = NULL,
    .app_validate_sleep     = NULL,
    .app_going_to_sleep     = NULL,
    .app_resume_from_sleep  = NULL,
};
```

```
void user_app_init(void)
{
    // Initialize Manufacturer Specific Data
    mnf_data_init();
    // Initialize default services and set sleep mode
    default_app_on_init();
}
```

```
// Default Handler Operations
static const struct default_app_operations user_default_app_operations = {
    .default_operation_adv = user_app_adv_start,
};
```

Custom service

Overview user_callback_config.h

```
static const struct app_callbacks user_app_callbacks = {
    // Handle connection request indication, if no connection has been established restart advertising
    .app_on_connection              = user_app_connection,
    .app_on_disconnect              = user_app_disconnect, // Restart Advertising
    /* Add the first required service in the database
       if database initialized then
       No service to add in the DB -> Start Advertising */
    .app_on_set_dev_config_complete = default_app_on_set_dev_config_complete,
    /* If advertising was canceled for any reason other then connection establishment
       then update advertising data and start advertising again */
    .app_on_adv_undirect_complete   = user_app_adv_undirect_complete,
    // database initialization is completed, then set the initial values of service characteristics programmatically
    .app_on_db_init_complete        = default_app_on_db_init_complete,
    .app_on_scanning_completed      = NULL, // NULL indicated this indication will not be handled by Dialog SDK;
    .app_on_adv_report_ind          = NULL, // either implement it or use the existing code based on your requirement
};

// Handles the messages that are not handled by the SDK internal mechanisms.
static const catch_rest_event_func_t app_process_catch_rest_cb = (catch_rest_event_func_t)user_catch_rest_hndl;
```

Custom service

user_custs_config.h

Add custom1 server function callback table.

```
/// Custom1/2 server function callback table this is linking point of your database and DA1458x SDK5.x.x or SDK6.x.x
static const struct cust_prf_func_callbacks cust_prf_funcs[] =
{
#if (BLE_CUSTOM1_SERVER)
   {  TASK_CUSTS1,
      custs1_att_db,
      CUST1_IDX_NB,
      #if (BLE_APP_PRESENT)
      app_custs1_create_db, app_custs1_enable,
      #else
      NULL, NULL,
      #endif
      custs1_init, NULL
   },
#endif
#if (BLE_CUSTOM2_SERVER)
   {  TASK_CUSTS2,
      NULL,
      0,
      #if (BLE_APP_PRESENT)
      app_custs2_create_db, app_custs2_enable,
      #else
      NULL, NULL,
      #endif
      custs2_init, NULL
   },
#endif
   {TASK_NONE, NULL, 0, NULL, NULL, NULL, NULL},   // DO NOT MOVE. Must always be last
};
```

```
/// Structure of custom profile call back function table.
struct cust_prf_func_callbacks
{
    /// Profile Task ID.
    enum KE_TASK_TYPE       task_id;
    /// pointer to the custom database table defined by user
    const struct attm_desc_128 *att_db;
    /// max number of attributes in custom database
    const uint8_t max_nb_att;
    /// Pointer to the custom database create function defined by
user
    prf_func_void_t         db_create_func;
    /// Pointer to the custom profile enable function defined by user
    prf_func_uint16_t       enable_func;
    /// Pointer to the custom profile initialization function
    prf_func_void_t         init_func;
    /// Pointer to the validation function defined by user
    prf_func_validate_t     value_wr_validation_func;
};
```

Contents

What would you see as output

What would you see as output

Output

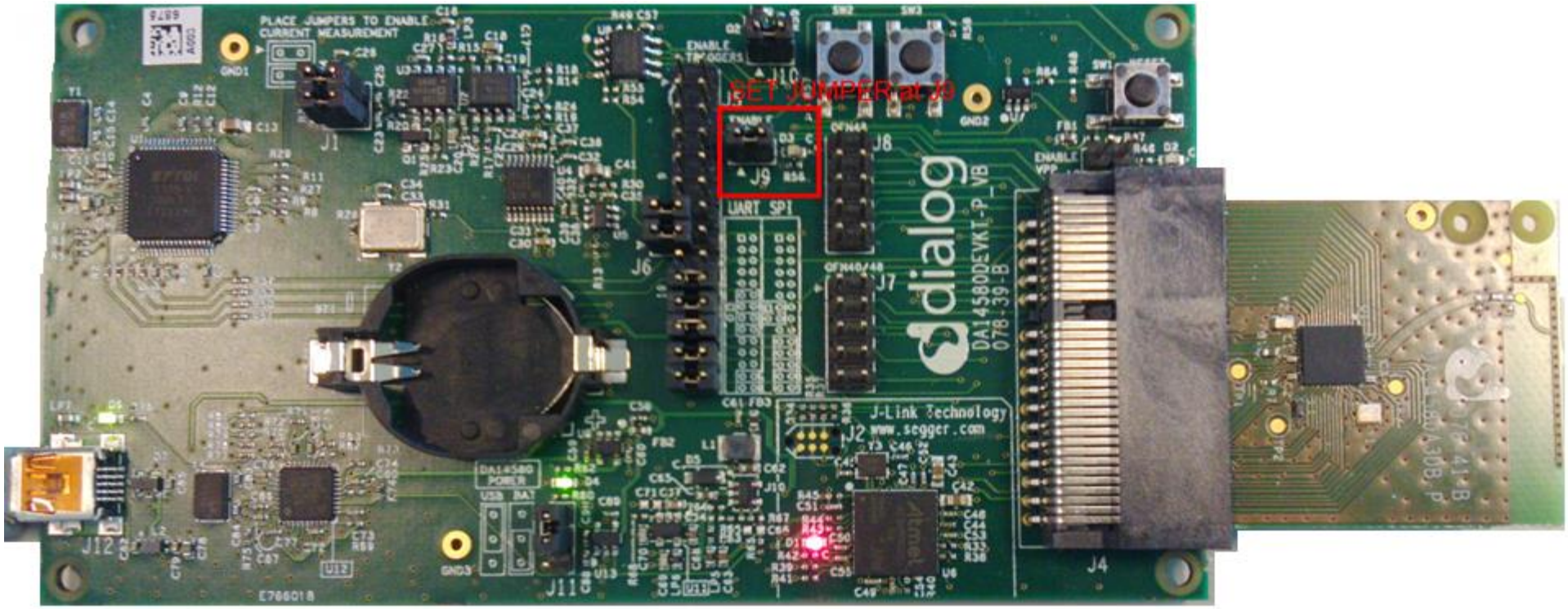
- The LightBlue iOS application can be used to connect an iPad/iPod/iPhone device to the application. In such a case the iPad/iPod/iPhone acts as a BLE Central and the application as a BLE Peripheral. It should be listed by the name given in the USER_DEVICE_NAME definition.

- One service should be listed – the Device Information Service. On some scanners, this will be listed either as a named service, or as a set of hex numbers (0A 18) as part of a list of 16-bit Service class UUIDs.

- On connecting to the device, the Characteristics should be retrieved.

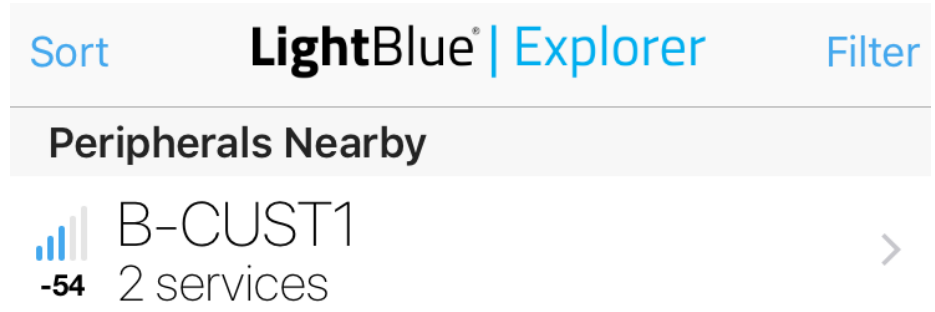
What would you see as output

DA1458x DK-Pro Configuration

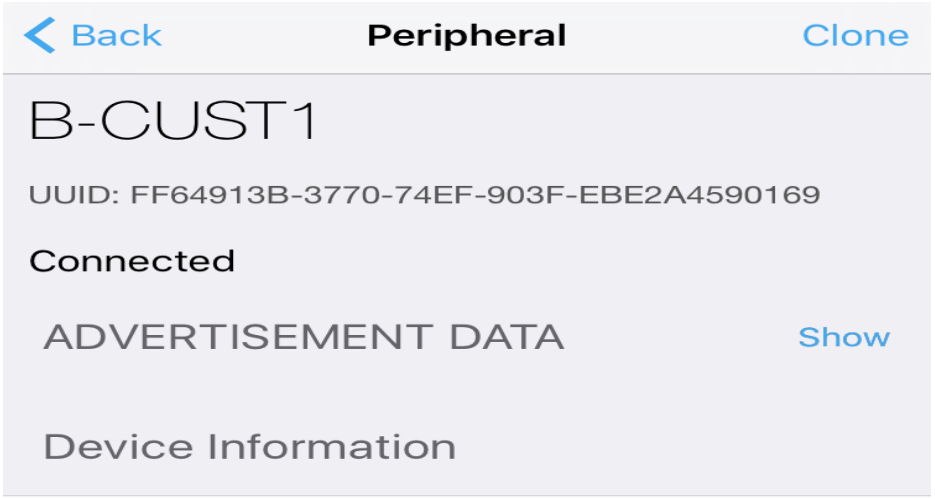


What would you see as output

What would you see as output



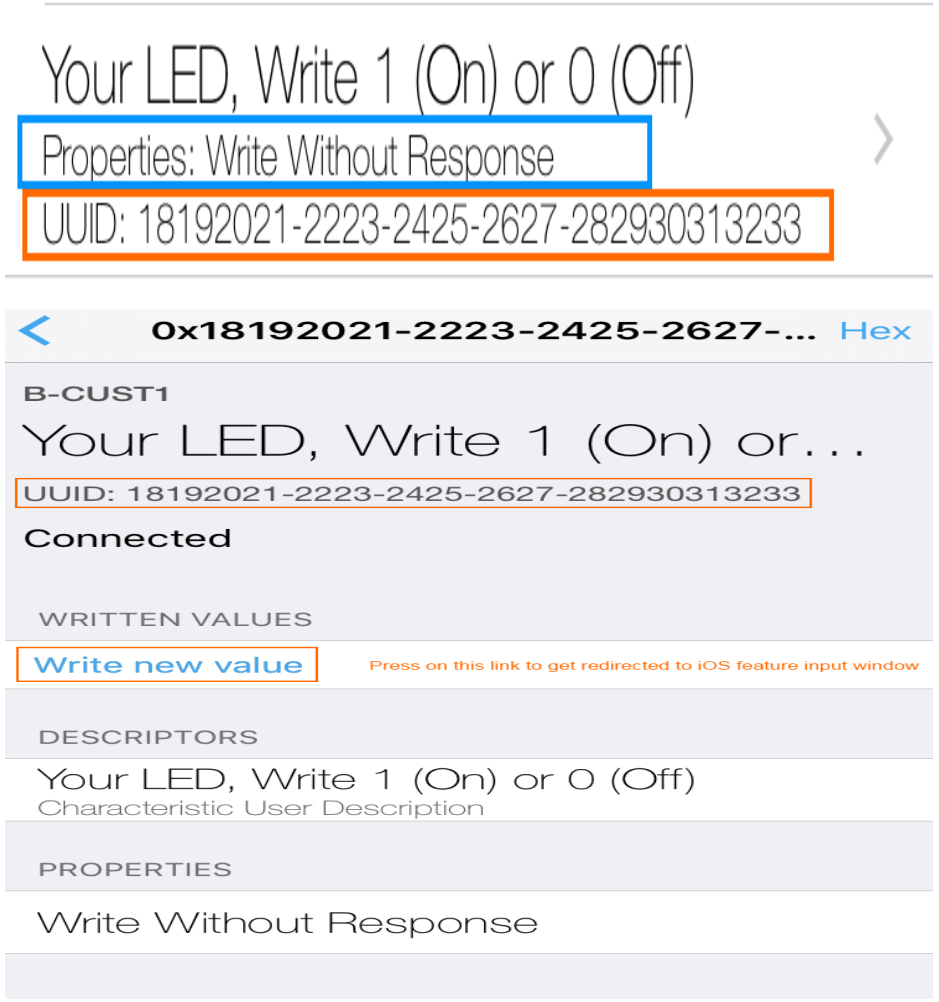
1. Your device is advertising



2. Your device is connected

What would you see as output

Output

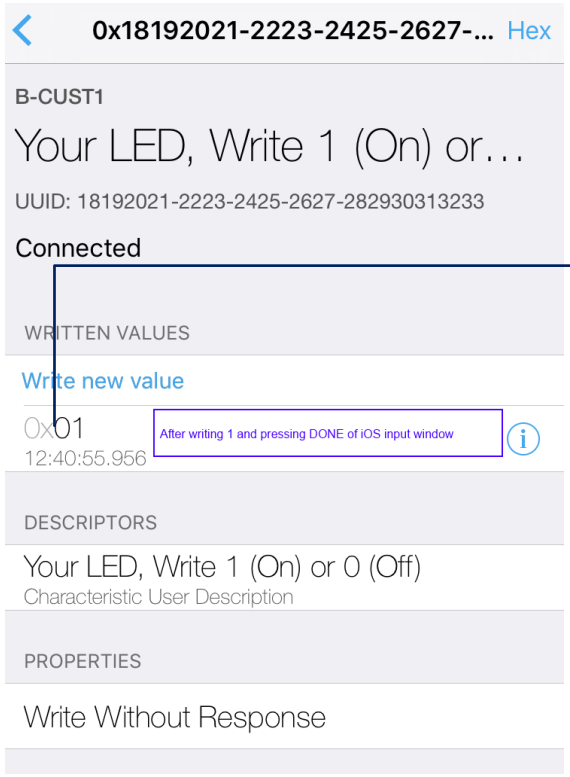


3. Your LED state characteristic

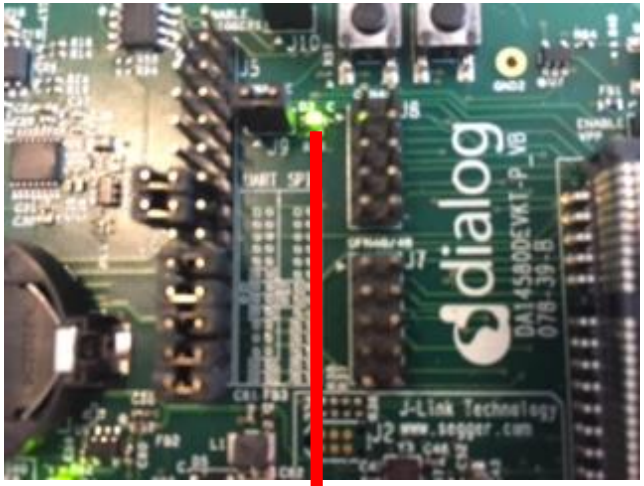
4. Follow the ORANGE instruction that is written beside Write new value

What would you see as output

Output



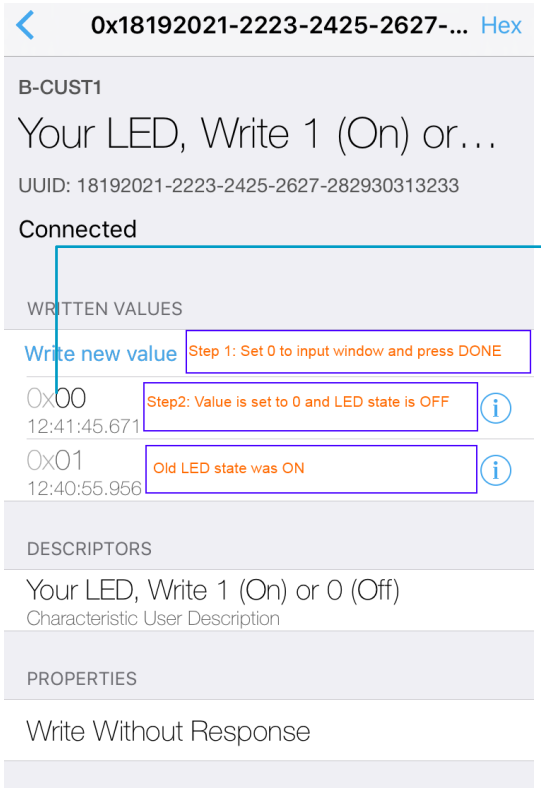
5. Verify 0x01 is written in iOS app



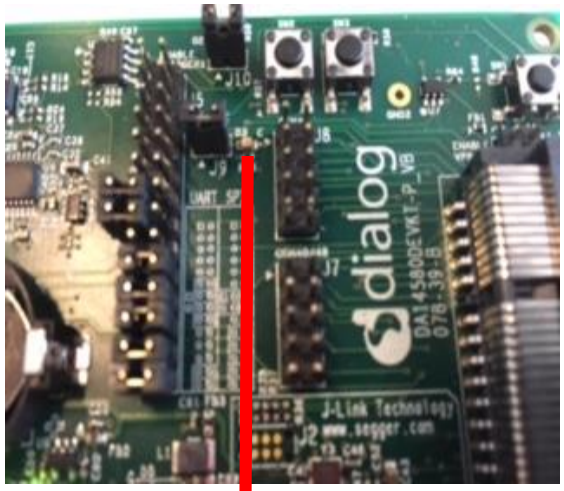
6. Check LED state on dev kit

What would you see as output

Output



7. Verify 0x00 is written in iOS app



8. Check LED state off dev kit

Information

What would you see as output

- **Note**: The devices will be connectable in this and future examples. Connecting to a device will mean that other scanners won't be able to locate the device – it is recommended that you only connect to your own device.

- **Note**: Some scanners (notably Apple devices) may not update the name of device if it is changed – to correct this, it is necessary to disable then re-enable Bluetooth.

Reference

Reference

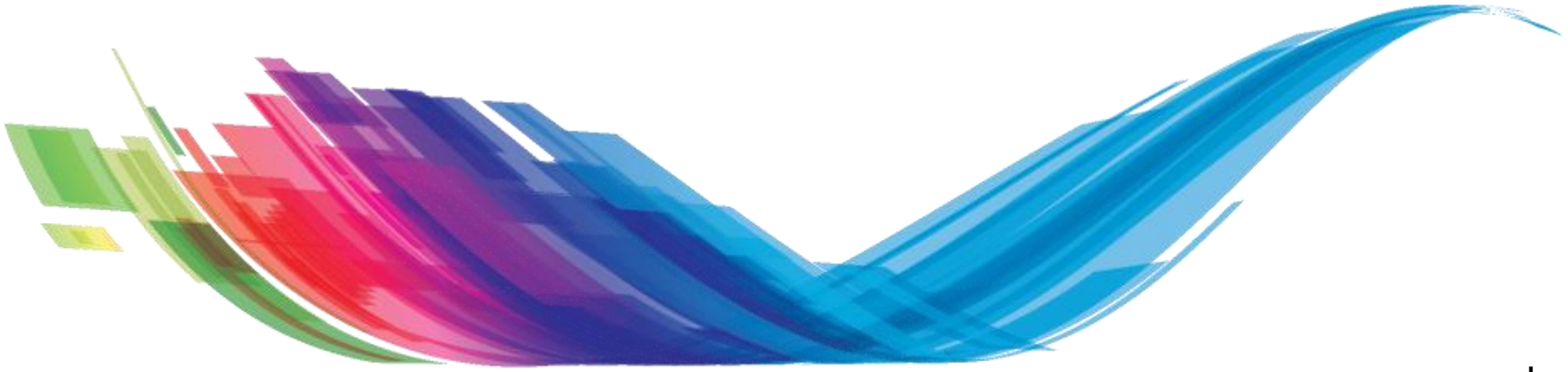
- http://support.dialog-semiconductor.com/connectivity
- https://developer.bluetooth.org/gatt/Pages/default.aspx
- https://www.bluetooth.com/specifications/adopted-specifications
- https://www.wikiwand.com/en/Universally_unique_identifier

What's next

For more …

- **What's next …**
  - Please follow the other tutorials based on –
    - **SDK 5.0.x** for **DA14580/1/2/3** development OR
    - **SDK 6.0.x** for **DA14585/6** development
  - See **Reference** section of this training slide
  - Learn about Dialog BLE chip **differences** at a glance from –
    https://support.dialog-semiconductor.com/connectivity/products

The Power To Be…



…personal
…portable
…connected