GATT格式

PRIMARY_SERVICE_UUID16 (0x0001, UUID_SERVICE_GATT),
首先定义一个服务，handle为0x0001，服务类型为UUID_SERVICE_GATT，实质是定义一个类型为UUID_ATTRIBUTE_PRIMARY_SERVICE的UUID
事实上，第一个字段handle好像没有太大作用。而且，虽然定义了服务，但是没有为这个服务定义特性。

PRIMARY_SERVICE_UUID16 (0x0014, UUID_SERVICE_GAP),
CHARACTERISTIC_UUID16 (0x0015, 0x0016, UUID_CHARACTERISTIC_DEVICE_NAME,
LEGATTDB_CHAR_PROP_READ, LEGATTDB_PERM_READABLE, 16),
'H','e','l','l','o',0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
CHARACTERISTIC_UUID16 (0x0017, 0x0018, UUID_CHARACTERISTIC_APPEARANCE,
LEGATTDB_CHAR_PROP_READ, LEGATTDB_PERM_READABLE, 2),
BIT16_TO_8(APPEARANCE_GENERIC_TAG),
和前面的服务定义方式类似，但是该服务会定义两个特性，分别是设备名称和可见性。
BIT16_TO_8，将一个16bit的short变量强制转化为两个8bit的char。
定义特性的各个字段：第一个，为handle，实际上这个handle字段貌似没有太大作用，有可能只是GATT内部使用。第二个，为handle_value，是handle的值。只有蓝牙的特性才有这个字段。第三个字段为特性的属性，包括BROADCAST, READ, WRITE_NO_RESPONSE, WRITE, NOTIFY, INDICATE, AUTHD_WRITES。第四个字段为特性的权限，包括READABLE，WRITE_CMD，WRITE_REQ等。需要参考Vol 3, Part F, 3.3.1.1。第五个字段为特性值的长度。最后为该特性的值。

PRIMARY_SERVICE_UUID128 (HANDLE_HELLO_SENSOR_SERVICE_UUID, UUID_HELLO_SERVICE),
新增一个自定义的服务。同样，这个handle好像也不知道咋用。UUID为自定义的16字节的字符。

CHARACTERISTIC_UUID128 (0x0029, HANDLE_HELLO_SENSOR_VALUE_NOTIFY, UUID_HELLO_CHARACTERISTIC_NOTIFY,
LEGATTDB_CHAR_PROP_READ | LEGATTDB_CHAR_PROP_NOTIFY | LEGATTDB_CHAR_PROP_INDICATE,
LEGATTDB_PERM_READABLE, 7),
'H','e','l','l','o',' ','0',
CHAR_DESCRIPTOR_UUID16_WRITABLE (HANDLE_HELLO_SENSOR_CLIENT_CONFIGURATION_DESCRIPTOR,
UUID_DESCRIPTOR_CLIENT_CHARACTERISTIC_CONFIGURATION,
LEGATTDB_PERM_READABLE | LEGATTDB_PERM_WRITE_REQ, 2),
0x00,0x00,
自定义服务的特性。UUID也是自定义的16字节的字符数组。
后面，还为这个特性定义了一个描述符。
对于服务和特性描述符，使用handle来查找，对于特性，使用handle_value来查找。

// Application initialization

```
APPLICATION_INIT()
{
    bleapp_set_cfg((UINT8 *)hello_sensor_gatt_database,
                   sizeof(hello_sensor_gatt_database),
                   (void *)&hello_sensor_cfg,
                   (void *)&hello_sensor_puart_cfg,
                   (void *)&hello_sensor_gpio_cfg,
                   hello_sensor_create);
}
```

hello_sensor_gatt_database为这个蓝牙设备的GATT数据库
hello_sensor_cfg 配置蓝牙设备的配置文件，包括广播方式，间隔。
hello_sensor_puart_cfg 配置波特率，
hello_sensor_gpio_cfg 配置GPIO，包括button，led，蜂鸣器，电池。gpio的定义取决于硬件设计。
hello_sensor_create 用于配置各种回调函数

void hello_sensor_create(void)

```
{
    BLEPROFILE_DB_PDU db_pdu;

    ble_trace0("\rhello_sensor_create()");
    ble_trace0(bleprofile_p_cfg->ver);

    bleprofile_Init(bleprofile_p_cfg);

    bleprofile_GPIOInit(bleprofile_gpio_p_cfg);

    hello_sensor_database_init(); //load handle number
```

```
    // register connection up and connection down handler.
    bleprofile_regAppEvtHandler(BLECM_APP_EVT_LINK_UP, hello_sensor_connection_up);
    bleprofile_regAppEvtHandler(BLECM_APP_EVT_LINK_DOWN, hello_sensor_connection_down);
    bleprofile_regAppEvtHandler(BLECM_APP_EVT_ADV_TIMEOUT, hello_sensor_advertisement_stopped);

    // handler for Encryption changed.
    blecm_regEncryptionChangedHandler(hello_sensor_encryption_changed);

    // handler for Bond result
    lesmp_regSMPResultCb((LESMP_SINGLE_PARAM_CB) hello_sensor_smp_bond_result);

    // register to process client writes
    legattdb_regWriteHandleCb((LEGATTDB_WRITE_CB)hello_sensor_write_handler);

    // register interrupt handler
    bleprofile_regIntCb((BLEPROFILE_SINGLE_PARAM_CB) hello_sensor_interrupt_handler);

    bleprofile_regTimerCb(hello_sensor_fine_timeout, hello_sensor_timeout);
    bleprofile_StartTimer();
```

调用通用函数，read data from peer gatt

```
    // Read value of the service from GATT DB.
    bleprofile_ReadHandle(HANDLE_HELLO_SENSOR_SERVICE_UUID, &db_pdu);
    ble_tracen((char *)db_pdu.pdu, db_pdu.len);

    if (db_pdu.len != 16)
    {
        ble_trace1("\rhello_sensor bad service UUID len: %d", db_pdu.len);
    }
    else
    {
     BLE_ADV_FIELD adv[3];

        // flags
        adv[0].len = 1 + 1;
        adv[0].val = ADV_FLAGS;
        adv[0].data[0] = LE_LIMITED_DISCOVERABLE | BR_EDR_NOT_SUPPORTED;

        adv[1].len = 16 + 1;
        adv[1].val = ADV_SERVICE_UUID128_COMP;
        memcpy(adv[1].data, db_pdu.pdu, 16);

        // name
        adv[2].len = strlen(bleprofile_p_cfg->local_name) + 1;
        adv[2].val = ADV_LOCAL_NAME_COMP;
        memcpy(adv[2].data, bleprofile_p_cfg->local_name, adv[2].len - 1);

        bleprofile_GenerateADVData(adv, 3);
    }
    blecm_setTxPowerInADV(0);
```

OTA初始化

```
    ws_upgrade_ota_init();

 bleprofile_Discoverable(HIGH_UNDIRECTED_DISCOVERABLE, hello_sensor_remote_addr);

 ble_trace1("E: Free bytes = 0x%08X", cfa_mm_MemFreeBytes());
}



/** @file  hello_sensor.c
*
* BLE Vendor Specific Device with Over the Air Upgrade
```

*
* During initialization the app registers with LE stack to receive various
* notifications including bonding complete, connection status change and
* peer write. When device is successfully bonded, application saves
* peer's Bluetooth Device address to the NVRAM. Bonded device can also
* write in to client configuration descriptor of the notification
* characteristic. That is also save in the NVRAM. When user pushes the
* button notification is sent to the bonded and registered host.
*
* Application also exposes a Vendor Specific Wiced Smart Upgrade service.
* The service exposes Control Point characteristic which application can
* use to send commands and receive notifications, and a Data characteristic
* which application uses to send chunks of data to the device.
*
* Features demonstrated
* - GATT database and Device configuration initialization
* - Registration with LE stack for various events
* - NVRAM read/write operation
* - Sending data to the client
* - Processing write requests from the client
* - Use of LED and Buzzer
* - Firmware over the air upgrade
*
* To demonstrate the app, work through the following steps.
* 1. Plug the WICED eval board into your computer
* 2. Build and download the application (to the WICED board)
* 3. Pair with a client
* 4. On the client side register for notifications
* 5. Push a button on the tag to send notifications to the client
* 6. Use WsOtaUpgrade application to try over the air upgrade
*
*/