什么是TuyaOS Bluetooth LE Peripheral

为了更快导入TuyaOS蓝牙服务并使用涂鸦提供的蓝牙服务，我们规范了一套跨多端的蓝牙接口。通过实现该接口，我们可以将指定涂鸦服务注册，并进行数据通道安全加密交互。通过实现TKL Ble Peripheral接口我们将所需服务进行注册并正确广播，通过实现读写接口以及数据回调实现加密数据交互等。

如何实现TuyaOS TKL ble接口

实现Ble Peripheral需要哪些接口

```
// 【必要】注册GAP回调，使用Peripheral时需要将TKL_BLE_EVT_STACK_INIT、TKL_BLE_GAP_EVT_CONNECT、TKL_BLE_GAP_EVT_DISCONNECT、TKL_BLE_GAP_EVT_CONN_PARAM_UPDATE等事件上报，不同的事件上报描述可参考tkl_bluetooth_def.h中TKL_BLE_GAP_EVT_TYPE_E
OPERATE_RET tkl_ble_gap_callback_register(CONST TKL_BLE_GAP_EVT_FUNC_CB gap_evt);

// 【必要】注册GATT回调，使用Peripheral时需要将TKL_BLE_GATT_EVT_MTU_RSP、TKL_BLE_GATT_EVT_WRITE_REQ等事件上报，不同的事件上报描述可参考tkl_bluetooth_def.h中TKL_BLE_GATT_EVT_TYPE_E
OPERATE_RET tkl_ble_gatt_callback_register(CONST TKL_BLE_GATT_EVT_FUNC_CB gatt_evt);

// 【必要】初始化蓝牙协议栈，可按需进行不同平台实现
OPERATE_RET tkl_ble_stack_init(UCHAR_T role);

// 【必要】这里的服务规则列表可参考开放文件tal_bluetooth.c中tal_ble_bt_init接口，涉及涂鸦使用的多类服务，按照服务->特征值->描述符规则实现即可。
OPERATE_RET tkl_ble_gatts_service_add(TKL_BLE_GATTS_PARAMS_T *p_service);

// 【必要】广播接口，使用Peripheral时需将如下广播接口进行实现，广播接口主要应用于我们将子设备的信息进行广播便于终端设备可发现且可连接上
OPERATE_RET tkl_ble_gap_adv_stop(VOID);
OPERATE_RET tkl_ble_gap_adv_start(TKL_BLE_GAP_ADV_PARAMS_T CONST *p_adv_params);
OPERATE_RET tkl_ble_gap_adv_rsp_data_set(TKL_BLE_DATA_T CONST *p_adv, TKL_BLE_DATA_T CONST *p_scan_rsp);
// 【注意】这里需说明该update接口，update是否开启广播由上一次开启状态决定，若实现平台有该接口，可直接对接。若无，我们可记录开启广播状态。
// 除特殊类产品外，我们在连接后update将不开启广播，仅作数据更新。
OPERATE_RET tkl_ble_gap_adv_rsp_data_update(TKL_BLE_DATA_T CONST *p_adv, TKL_BLE_DATA_T CONST *p_scan_rsp);

// 【必要】最大传输单元协商接口，使用Peripheral时会进行最大传输单元的协商行为，此时通过如下接口进行请求MTU，且最终协商后的MTU将通过TAL_BLE_EVT_MTU_RSP进行上报至业务端
OPERATE_RET tkl_ble_gattc_exchange_mtu_request(USHORT_T conn_handle, USHORT_T client_rx_mtu);

// 【必要】数据交互接口，使用Peripheral时会进行子设备数据上报，即通过notification方式进行数据上报至终端，此时将采用如下接口进行发送。
OPERATE_RET tkl_ble_gatts_value_notify(USHORT_T conn_handle, USHORT_T char_handle, UCHAR_T *p_data, USHORT_T length);
```

```
26
27  // 【按需】数据交互接口，使用Peripheral时会进行Read属性特征值进行数据更新，此时将采用
    value set方式进行设定。
28  // 如仅仅是配网需求，可按需实现该接口
29  OPERATE_RET tkl_ble_gatts_value_set(USHORT_T conn_handle, USHORT_T
    char_handle, UCHAR_T *p_data, USHORT_T length);
30
31  // 【按需】连接参数更新接口，使用Peripheral时我们将按需进行连接参数更新以达到功耗要求。针
    对OTA等大数据量交互时我们将缩短连接间隔以达到更大的吞吐效果。
32  // 如仅仅是配网需求，可按需实现该接口。
33  OPERATE_RET tkl_ble_gap_conn_param_update(USHORT_T conn_handle,
    TKL_BLE_GAP_CONN_PARAMS_T CONST *p_conn_params);
34
35  // 【按需】这里需要提及到扫描接口。如采用涂鸦产测或Ble Beacon等方案，需要将扫描接口进行实
    现，对应上报事件为TKL_BLE_GAP_EVT_ADV_REPORT。
36  // 如仅仅是配网需求，可按需实现该接口。
37  OPERATE_RET tkl_ble_gap_scan_start(TKL_BLE_GAP_SCAN_PARAMS_T CONST
    *p_scan_params);
38  OPERATE_RET tkl_ble_gap_scan_stop(VOID);
```

注：上述中必要接口将作为TuyaOS Ble Peripheral必须实现的接口逻辑；

如何实现接口

实现TuyaOS TKL上述接口不限于Linux/FreeRTOS/None-OS等等，对于不同平台的实现可按照具体规则进行实现；

单片机端/FreeRTOS端可参考开放SDK中Nordic Ble进行参考，对于Linux端可参考BlueZ中btgatt-server实现方式或采用dbus-bluetoothd方式进行Peripheral实现。

涉及平台类导入问题可咨询原厂或TuyaOS论坛提出相关逻辑性问题；

如何基于TuyaOS TKL ble接口实现一个简单的Peripheral

```
1   #define TUYA_BLE_MAX_MTU_SIZE       (247)
2
3   STATIC UCHAR_T adv_data_const[31] =
4   {
5       0x02,
6       0x01,
7       0x06,
8       0x03,
9       0x02,
10      0x50, 0xFD,
11      0x17,
12      0x16,
13      0x50, 0xFD,
14      0x41, 0x00,       //Frame Control
15      0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00
16  };
17
18  STATIC UCHAR_T scan_rsp_data_const[31] =
19  {
```

```
    0x17,             // length
    0xFF,
    0xD0,
    0x07,
    0x00, //Encry Mode(8)
    0x00,0x00, //communication way bit0-mesh bit1-wifi bit2-zigbee bit3-NB
    0x00, //FLAG

 0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00
,0x00,
    0x03, //24
    0x09,
    0x54, 0x59,
};

STATIC TAL_BLE_PEER_INFO_T ble_peripheral_info;

STATIC UCHAR_T ble_peripheral_connected = 0;

STATIC VOID tuyaos_bluetooth_event_callback(TAL_BLE_EVT_PARAMS_T *p_event)
{
    switch(p_event->type) {
        // Service Register and Stack init Event
        case TAL_BLE_STACK_INIT:
            if(p_event->ble_event.init == 0) {
                TAL_BLE_DATA_T adv_data;
                TAL_BLE_DATA_T rsp_data;

                adv_data.p_data = adv_data_const;
                adv_data.len = sizeof(adv_data_const);

                rsp_data.p_data = scan_rsp_data_const;
                rsp_data.len = sizeof(scan_rsp_data_const);

                tal_ble_advertising_data_set(&adv_data, &rsp_data);
                // After Init Successfully, we will do adv
                tal_ble_advertising_start(TUYAOS_BLE_DEFAULT_ADV_PARAM);
            }
            break;

        // Peripheral Connect Event
        case TAL_BLE_EVT_PERIPHERAL_CONNECT:
            if(p_event->ble_event.connect.result == 0) {
                TAL_BLE_DATA_T read_data;
                UCHAR_T read_buffer[512];

                memcpy(&ble_peripheral_info, &p_event-
>ble_event.connect.peer, sizeof(TAL_BLE_PEER_INFO_T));
                memcpy(read_buffer, adv_data_const,
sizeof(adv_data_const));
                memcpy(&read_buffer[sizeof(adv_data_const)],
scan_rsp_data_const, sizeof(scan_rsp_data_const));

                read_data.p_data = read_buffer;
                read_data.len = sizeof(adv_data_const) +
sizeof(scan_rsp_data_const);
```

```
70
71                  // Verify Read Char
72                  tal_ble_server_common_read_update(&read_data);
73
74                  // Try to request the mtu exchange
75                  tal_ble_client_exchange_mtu_request(ble_peripheral_info,
    247);
76
77                  // Try to request connection parameter update
78                  tal_ble_conn_param_update(ble_peripheral_info,
    TUYAOS_BLE_DEFAULT_CONN_PARAM);
79
80                  ble_peripheral_connected  = 1;
81              }else {
82                  memset(&ble_peripheral_info, 0,
    sizeof(TAL_BLE_PEER_INFO_T));
83              }
84              break;
85
86          // Disconnect Event
87          case TAL_BLE_EVT_DISCONNECT: {
88                  ble_peripheral_connected = 0;
89                  tal_ble_advertising_start(TUYAOS_BLE_DEFAULT_ADV_PARAM);
90              } break;
91
92          // Data From the Client
93          case TAL_BLE_EVT_WRITE_REQ: {
94                  TAL_BLE_DATA_T send_data;
95                  UCHAR_T send_buffer[247];
96
97                  if(p_event->ble_event.write_report.peer.char_handle[0] ==
    ble_peripheral_info.char_handle[TAL_COMMON_WRITE_CHAR_INDEX]) {
98                      send_data.p_data = send_buffer;
99                      memcpy(send_data.p_data, p_event-
    >ble_event.write_report.report.p_data, p_event-
    >ble_event.write_report.report.len);
100                     send_data.len = p_event-
    >ble_event.write_report.report.len;
101
102                     printf("Connect Handle(0x%02x), Char
    Handle(0x%02x)\r\n", p_event->ble_event.write_report.peer.conn_handle,
103                         p_event-
    >ble_event.write_report.peer.char_handle[0]);
104
105                     // Send the same data into the client
106                     tal_ble_server_common_send(&send_data);
107                 }
108             } break;
109
110         // Connection Parameters Update Report
111         case TAL_BLE_EVT_CONN_PARAM_UPDATE: {
112                 // Show Conn Parameters Info
113
114                printf("Conn Param Update: Min = %f ms, Max = %f ms, Latency
    = %d, Sup = %d ms\n",
```

```
                    p_event->ble_event.conn_param.conn.min_conn_interval * 1.25,
                    p_event->ble_event.conn_param.conn.max_conn_interval * 1.25,
                    p_event->ble_event.conn_param.conn.latency,
                    p_event->ble_event.conn_param.conn.conn_sup_timeout * 10);
            } break;

        // MTU Exchange Response Event
        case TAL_BLE_EVT_MTU_RSP:{
                // Show MTU Rsponse Data
               printf("Get Response MTU Size = %d", p_event->ble_event.exchange_mtu.mtu);
            } break;
    }
}

int main(void)
{
    tal_ble_bt_init(TAL_BLE_ROLE_PERIPERAL, tuyaos_bluetooth_event_callback);

    while (true) {
        sleep(1);
    }
}
```

通过实现TKL Bluetooth接口，我们对该类接口进行简要的逻辑整理并输出了统一的业务端接口。通过上述Demo以及tal bluetooth的实现，我们将完成一个简要的Demo作为测试；

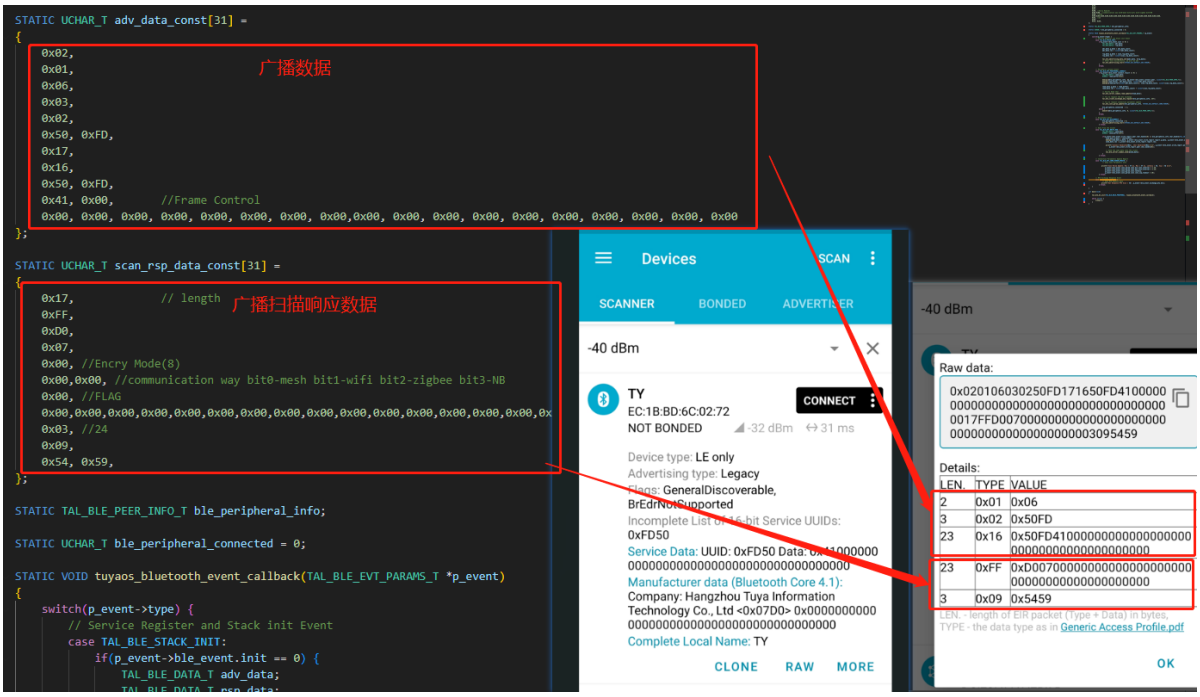
如何验证Peripheral通道可行性

验证方法

这里采用nordic推出的nrf-connect软件：[nRF Connect for Mobile - nordicsemi.com]

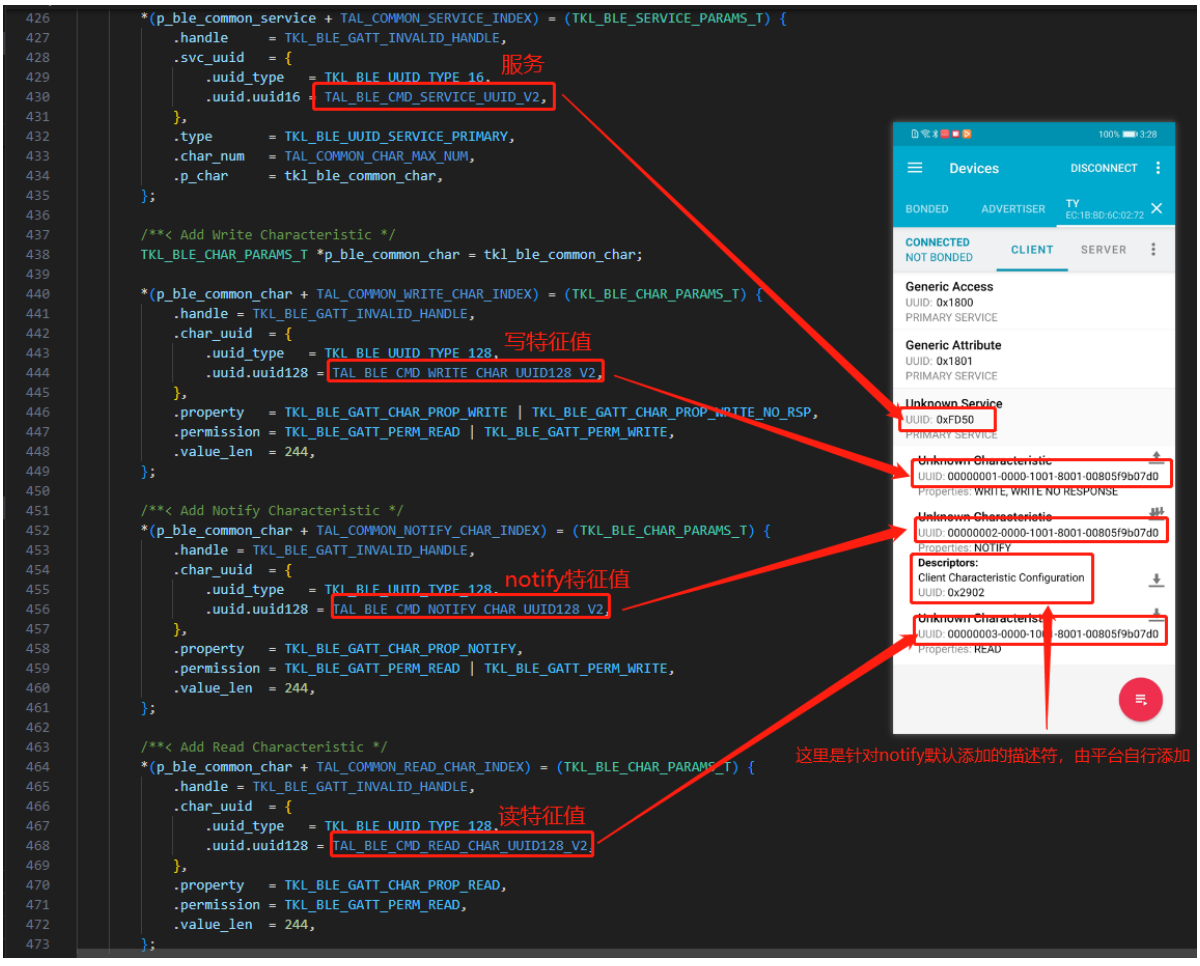
建议采用安卓手机下载并验证，IOS受限安全规则无法展示我们需要的所有信息。

【必要】验证广播与扫描响应包

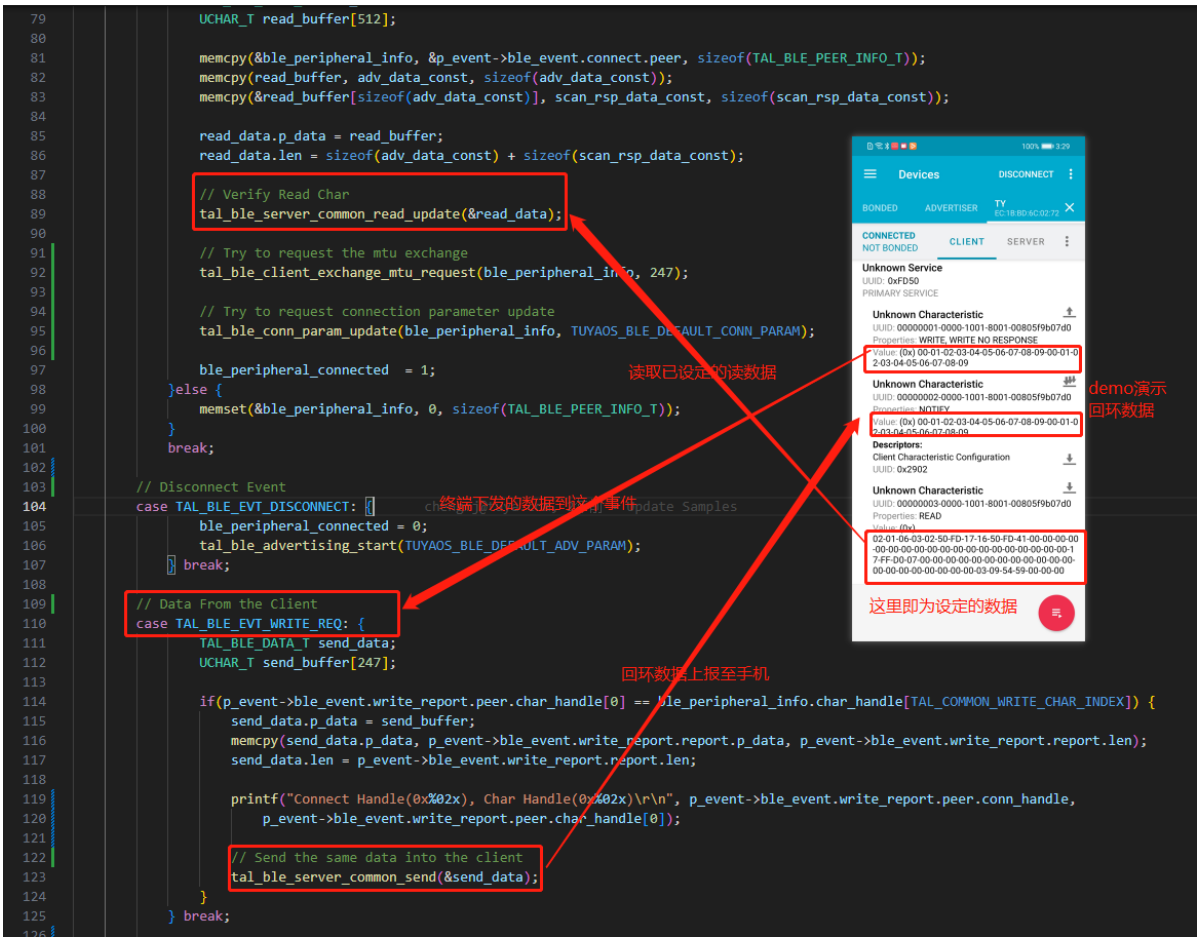


必须确认两笔数据和实际发送一致，特别是校验长度LEN。其次便是广播扫描响应数据是否有发出。这里的展示数据从RAW处点击进入。

【必要】验证服务注册



【必要】验证数据交互




```
79              UCHAR_T read_buffer[512];
80
81              memcpy(&ble_peripheral_info, &p_event->ble_event.connect.peer, sizeof(TAL_BLE_PEER_INFO_T));
82              memcpy(read_buffer, adv_data_const, sizeof(adv_data_const));
83              memcpy(&read_buffer[sizeof(adv_data_const)], scan_rsp_data_const, sizeof(scan_rsp_data_const));
84
85              read_data.p_data = read_buffer;
86              read_data.len = sizeof(adv_data_const) + sizeof(scan_rsp_data_const);
87
88              // Verify Read Char
89              tal_ble_server_common_read_update(&read_data);
90
91              // Try to request the mtu exchange
92              tal_ble_client_exchange_mtu_request(ble_peripheral_info, 247);
93
94              // Try to request connection parameter update
95              tal_ble_conn_param_update(ble_peripheral_info, TUYAOS_BLE_DEFAULT_CONN_PARAM);
96
97              ble_peripheral_connected  = 1;
98          }else {
99              memset(&ble_peripheral_info, 0, sizeof(TAL_BLE_PEER_INFO_T));
100         }
101         break;
102
103     // Disconnect Event
104     case TAL_BLE_EVT_DISCONNECT: {
105             ble_peripheral_connected = 0;
106             tal_ble_advertising_start(TUYAOS_BLE_DEFAULT_ADV_PARAM);
107         } break;
108
109     // Data From the Client
110     case TAL_BLE_EVT_WRITE_REQ: {
111             TAL_BLE_DATA_T send_data;
112             UCHAR_T send_buffer[247];
113
114             if(p_event->ble_event.write_report.peer.char_handle[0] == ble_peripheral_info.char_handle[TAL_COMMON_WRITE_CHAR_INDEX]) {
115                 send_data.p_data = send_buffer;
116                 memcpy(send_data.p_data, p_event->ble_event.write_report.report.p_data, p_event->ble_event.write_report.report.len);
117                 send_data.len = p_event->ble_event.write_report.report.len;
118
119                 printf("Connect Handle(0x%02x), Char Handle(0x%02x)\r\n", p_event->ble_event.write_report.peer.conn_handle,
120                     p_event->ble_event.write_report.peer.char_handle[0]);
121
122                 // Send the same data into the client
123                 tal_ble_server_common_send(&send_data);
124             }
125         } break;
```