添加自定义服务教程

Beken Corporation
博通集成电路(上海)有限公司
中国上海张江高科技园区
张东路 1387 科技领袖之都 41 栋
电话： (86)21 5108 6811
传真： (86)21 6087 1277

更改记录

版本号	日期	作者	注释
1.0	2018-01-24	许 海	文档建立

一、 实现自定义服务

添加一个自定义服务需要实现 6 个文件，分别为：

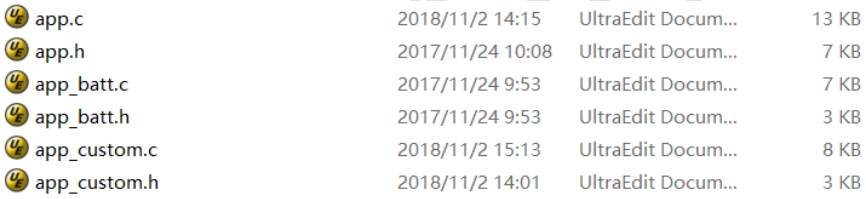
custom.c
custom.h
custom_task.c
custom_task.h
app_custom.c
app_custom.h

在 sdk\ble_stack\com\profiles\目录下新建 custom 文件夹，将 custom.h、custom_task.h 文件放在 api 目录下，custom.c、custom_task.c 两个文件放在 src 目录下。将 app_custom.c 与 app_custom.h 存放在 projects\ble_app_gatt\app\目录下。

本文档以 custom 服务为例介绍了如何添加一个服务到 RW 的详细步骤。

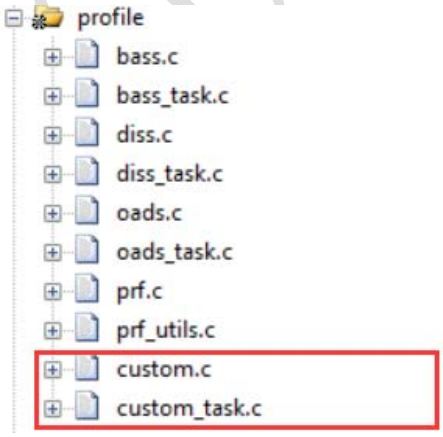
1. sdk\ble_stack\com\profiles\custom 目录如下：

2. projects\ble_app_gatt\app 目录如下：

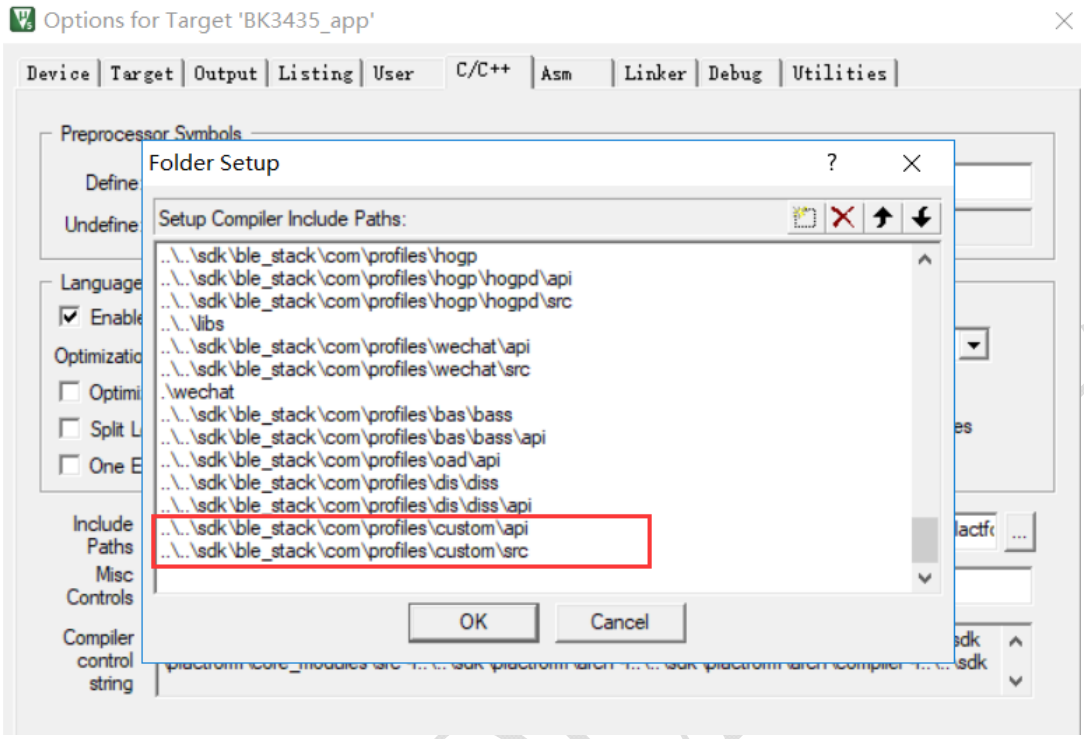


二、添加 service 到工程中

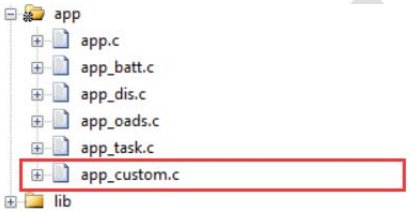
1. 使用 keil 5.12 打开对应工程，在工程目录 profile 目录下添加指定 profile 相关源文件，本例中是 custom.c 和 custom_task.c,如下图：



2. 在 keil C/C++选项卡下面添加对应 profile 的路径，如下：



3. 在工程目录 app 目录下添加指定自定义服务在应用层文件：



4. 在 app_task.c 文件中找到 appm_msg_handler 函数，在其中自定义服务获取 handler 的函数，如下：

```
case (TASK_ID_CUSTOM):
{
    // Call the custom  Module
    msg_pol = appm_get_handler(&app_custom_table_handler, msgid, param, src_id);
} break;
```

5. 在 app.c 文件中的服务列表中添加自定义服务，如下：

```
/// List of service to add in the database
enum appm_svc_list
{
    APPM_SVC_CUSTOM,
    APPM_SVC_DIS,
    APPM_SVC_BATT,
    APPM_SVC_OADS,
    APPM_SVC_LIST_STOP ,
};
```

6. 在 app.c 文件中的函数列表中增加往 db 添加自定义服务的函数，如下：

```
/// List of functions used to create the database
static const appm_add_svc_func_t appm_add_svc_func_list[APPM_SVC_LIST_STOP] =
{
    (appm_add_svc_func_t)app_custom_add_custom,
    (appm_add_svc_func_t)app_dis_add_dis,
    (appm_add_svc_func_t)app_batt_add_bas,
    (appm_add_svc_func_t)app_oad_add_oads,
};
```

7. 在 app.c 文件中找到 appm_init 函数，在其中添加 app_custom_init()函数，如下：

```
// Device Information Module
app_dis_init();
//Battery Module
app_batt_init();
//oad Module init
app_oads_init();
//custom Module init
app_custom_init();
```

8. 在 rwip_task.h 文件中增加自定义服务的 task_id,如下：

```
/// Tasks types definition, this value shall be in [0-254] range
enum TASK_API_ID
{
    // Link Layer Tasks
    TASK_ID_LLM          = 0,
    TASK_ID_LLC          = 1,
    TASK_ID_LLD          = 2,
    TASK_ID_DBG          = 3,
    ...
    ...
    ...
    TASK_ID_FCC0S        = 74,   //FFC0 PROFILE SERVICE TASK
    TASK_ID_FEE0S        = 75,   //FEE0 PROFILE SERVICE TASK
    TASK_ID_CUSTOM       = 76,   //CUSTOM PROFILE SERVICE TASK

    /* 240 -> 241 reserved for Audio Mode 0 */
    TASK_ID_AM0          = 240,  // BLE Audio Mode 0 Task
    TASK_ID_AM0_HAS      = 241,  // BLE Audio Mode 0 Hearing Aid Service Task
}
```

9. 在 prf.c 文件中添加 custom_prf_itf_get()的调用：

```
/**
 ****************************************************************************************
 * @brief Retrieve profile interface
 ****************************************************************************************
 */
static const struct prf_task_cbs * prf_itf_get(uint16_t task_id)
{
    const struct prf_task_cbs* prf_cbs = NULL;

    switch(KE_TYPE_GET(task_id))
    {
        #if (BLE_CUSTOM_SERVER)
        case TASK_ID_CUSTOM:
            prf_cbs = custom_prf_itf_get();
            break;
        #endif // (BLE_CUSTOM_SERVER)

        #if (BLE_OADS_SERVER)
        case TASK_ID_OADS:
            prf_cbs = oads_prf_itf_get();
            break;
        #endif // (BLE_OADS_SERVER)
```

10. 在 prf.c 文件中添加 custom_prf_itf_get()的声明：

```
#if (BLE_FFF0_SERVER)
extern const struct prf_task_cbs* fff0s_prf_itf_get(void);
#endif // (BLE_FFF0_SERVER)

#if (BLE_CUSTOM_SERVER)
extern const struct prf_task_cbs* custom_prf_itf_get(void);
#endif // (BLE_CUSTOM_SERVER)
```

11. 在 rwprf_config.h 文件中增加如下定义：

```
//  <e> CFG_PRF_CUSTOM
//  <i> CUSTOM profile Role
//  </e>
#if ( 1 )
#define CFG_PRF_CUSTOM
#endif

///CUSTOM Profile role
#if defined(CFG_PRF_CUSTOM)
#define BLE_CUSTOM_SERVER        1
#else
#define BLE_CUSTOM_SERVER        0
#endif // defined(CFG_PRF_CUSTOM)

#if (BLE_PROX_REPORTER || BLE_FINDME_TARGET || BLE_HT_THERMOM || BLE_BP_SENSOR \
        || BLE_TIP_SERVER || BLE_HR_SENSOR || BLE_DIS_SERVER || BLE_SP_SERVER \
        || BLE_BATT_SERVER || BLE_HID_DEVICE || BLE_GL_SENSOR || BLE_RSC_SENSOR  \
        || BLE_CSC_SENSOR || BLE_CP_SENSOR || BLE_LN_SENSOR || BLE_AN_SERVER \
        || BLE_PAS_SERVER || BLE_IPS_SERVER || BLE_ENV_SERVER || BLE_WSC_SERVER \
        || BLE_UDS_SERVER || BLE_BCS_SERVER || BLE_WPT_SERVER || BLE_PLX_SERVER \
        || BLE_FFF0_SERVER || BLE_FFE0_SERVER || BLE_CUSTOM_SERVER)
#define BLE_SERVER_PRF           1
#else
#define BLE_SERVER_PRF           0
#endif
```

BLE_CUSTOM_SERVER 这个宏定义在 custom.c、custom.h、custom_task.c、custom_task.h 中都有用到，只有当这个宏被打开后，编译器在编译的时候才能把自定义的服务编译进去。部分截图如下：

custom.c

```
12   */
13
14  #include "rwip_config.h"
15
16 #if (BLE_CUSTOM_SERVER)
17  #include "attm.h"
18  #include "custom.h"
19  #include "custom_task.h"
20  #include "prf_utils.h"
21  #include "prf.h"
22  #include "ke_mem.h"
23
24  #include "uart.h"
25
```

custom_task.c

```
19
20  #include "rwip_config.h"
21
22 #if (BLE_CUSTOM_SERVER)
23
24  #include "gap.h"
25  #include "gattc_task.h"
26  #include "attm.h"
27  #include "atts.h"
28  #include "co_utils.h"
29  #include "custom.h"
30  #include "custom_task.h"
31  #include "uart.h"
32  #include "prf_utils.h"
33
```

以上步骤全部完成之后，整体编译 project，使用 lightblue 连接进行验证是否添加成功。

三、app_custom.c 接口说明：

1. custom 服务实现了两个特征，其 UUID 及相关属性如下表所示：

UUID	特征权限	可发送/接收数据长度	备注
0xAAB0			Service UUID
0xAAB1	Norify	20bytes	Tx
0xAAB2	Write without response	20bytes	Rx

2. 数据发送 API

```
void app_char1_send_lvl(uint8_t* buf, uint8_t len)
{
    // Allocate the message
    struct custom_char1_level_upd_req * req = KE_MSG_ALLOC(CUSTOM_CHAR1_LEVEL_UPD_REQ,
                                                           prf_get_task_from_id(TASK_ID_CUSTOM),
                                                           TASK_APP,
                                                           custom_char1_level_upd_req);
    // Fill in the parameter structure
    req->length = len;
    memcpy(req->char1_level, buf, len);

    // Send the message
    ke_msg_send(req);
}
```

3. 数据接收回调

```
static int char2_writer_req_handler(ke_msg_id_t const msgid,
                                    struct custom_char2_writer_ind *param,
                                    ke_task_id_t const dest_id,
                                    ke_task_id_t const src_id)
{
    // Drop the message
    UART_PRINTF("char2 param->value = 0x ");

    for(uint8_t i = 0;i < param->length;i++)
    {
        UART_PRINTF("%02x ",param->char2_value[i]);
    }
    UART_PRINTF("\r\n");

    return (KE_MSG_CONSUMED);
}
```

4. 数据发送状态回复

```
static int char1_level_upd_handler(ke_msg_id_t const msgid,
                                   struct custom_char1_level_upd_rsp const *param,
                                   ke_task_id_t const dest_id,
                                   ke_task_id_t const src_id)
{
    if(param->status == GAP_ERR_NO_ERROR)
    {
        //test
        //uint8_t buf[20];
        //memset(buf, 0xcc, 20);
        //app_char1_send_lvl(buf, 20);
    }

    return (KE_MSG_CONSUMED);
}
```

5. Notify 成功使能回调

```
static int custom_char1_level_ntf_cfg_ind_handler(ke_msg_id_t const msgid,
                                    struct custom_char1_level_ntf_cfg_ind const *param,
                                    ke_task_id_t const dest_id,
                                    ke_task_id_t const src_id)
{
    if(param->ntf_cfg == PRF_CLI_STOP_NTFIND)
    {
        //notify disable
    }else
    {
        //notify enable
    }

    return (KE_MSG_CONSUMED);
}
```

为了系统的稳定，应用层发送 notify 到手机端时需要在发送完成一包数据之后再触发下一次的发送。