

说明：该文档以FFF0服务下的FFF1通道增加写属性为例子进行说明，客户可根据需要进行其他修改。

1. 在fff0s.c文件下做如下图修改。

```
static uint8_t fff0s_init (struct prf_task_env* env, uint16_t* start_hdl, uint16_t app_task, uint8_t
{
    uint16_t shdl;
    struct fff0s_env_tag* fff0s_env = NULL;
    // Status
    uint8_t status = GAP_ERR_NO_ERROR;
    //----- allocate memory required for the profile -----
    fff0s_env = (struct fff0s_env_tag*) ke_malloc(sizeof(struct fff0s_env_tag), KE_MEM_ATT_DB);
    memset(fff0s_env, 0, sizeof(struct fff0s_env_tag));
    // Service content flag
    uint16_t cfg_flag = 0x1ff; //FFF0S_CFG_FLAG_MANDATORY_MASK;
    // Save database configuration
    fff0s_env->features |= (params->features);
    #if 0
    // Check if notifications are supported
    if (params->features == FFF0_FFF1_LVL_NTF_SUP)
    {
        cfg_flag |= FFF0_CFG_FLAG_NTF_SUP_MASK;
    }
    #endif
    shdl = *start_hdl;
    //Create FFF0 in the DB
    //----- create the attribute database for the profile -----
    status = attm_svc_create_db(&(shdl), ATT_USER_SERVER_FFF0, (uint8_t *)&cfg_flag,
        FFF0S_IDX_NB, NULL, env->task, &fff0_att_db[0],
        (sec_lvl & (PERM_MASK_SVC_DIS | PERM_MASK_SVC_AUTH | PERM_MASK_SVC_EKS)));
    //Set optional permissions
    if (status == GAP_ERR_NO_ERROR)
    {
        //Set optional permissions
        if((params->features & 0x01) == FFF0_FFF1_LVL_NTF_SUP)
        {
            // Battery Level characteristic value permissions
            uint16_t perm = fff0_att_db[FFF0S_IDX_FFF1_LVL_VAL].perm;
            perm |= PERM(RD, ENABLE) | PERM(NTF, ENABLE);
            attm_att_set_permission(shdl + FFF0S_IDX_FFF1_LVL_VAL, perm, 0);
        }
    }
    //----- Update profile task information -----
    if (status == ATT_ERR_NO_ERROR)
    {

```

2. 在fff0s_task.h文件下的枚举类型fff0s_msg_id添加枚举成员

FFF0S_FFF1_WRITER_REQ_IND和结构体fff0s_fff1_writer_ind。

```
/// Messages for FFF0 Server
enum fff0s_msg_id
{
    /// Start the FFF0 Server - at connection used to restore bond data
    FFF0S_CREATE_DB_REQ = TASK_FIRST_MSG(TASK_ID_FFF0S),
    /// FFF1 Level Value Update Request
    FFF0S_FFF1_LEVEL_UPD_REQ,
    /// Inform APP if FFF1 Level value has been notified or not
    FFF0S_FFF1_LEVEL_UPD_RSP,
    /// Inform APP that FFF1 Level Notification Configuration has been changed - use to update
    FFF0S_FFF1_LEVEL_NTF_CFG_IND,
    FFF0S_FFF1_WRITER_REQ_IND,
    FFF0S_FFF2_WRITER_REQ_IND,
    FFF0S_FFF1_LEVEL_PERIOD_NTF
};
/// Parameters of the @ref FFF0S_FFF1_WRITER_REQ_IND message
struct fff0s_fff1_writer_ind
{
    /// Alert level
    uint8_t fff1_value[FFF0_FFF1_DATA_LEN];
    uint8_t length;
    /// Connection index
    uint8_t conidx;
};
```

3. 在ff0s_task.c文件下的gattc_write_req_ind_handler函数下增加代码。

```

8 static int gattc_write_req_ind_handler(ke_msg_id_t const msgid, struct gattc_write_req_ind const
9                                     ke_task_id_t const dest_id, ke_task_id_t const src_id)
10 {
11     UART_PRINTF("ff0s_task.c:%s line:%d\r\n", __func__, __LINE__);
12     struct gattc_write_cfm * cfm;
13     uint8_t att_idx = 0;
14     uint8_t conidx = KE_IDX_GET(src_id);
15     // retrieve handle information
16     uint8_t status = fff0s_get_att_idx(param->handle, &att_idx);
17     // If the attribute has been found, status is GAP_ERR_NO_ERROR
18     if (status == GAP_ERR_NO_ERROR)
19     {
20         struct fff0s_env_tag* fff0s_env = PRF_ENV_GET(FFF0S, fff0s);
21         // Extract value before check
22         uint16_t ntf_cfg = co_read16p(&param->value[0]);
23         // Only update configuration if value for stop or notification enable
24         if ((att_idx == FFF0S_IDX_FFF1_LVL_NTF_CFG) ...)
25         {
26             else if (param->handle == fff0s_env->start_hdl + FFF0S_IDX_FFF1_LVL_VAL)
27             {
28                 // Allocate the alert value change indication
29                 struct fff0s_fff1_writer_ind *ind = KE_MSG_ALLOC(FFF0S_FFF1_WRITER_REQ_IND,
30                     prf_dst_task_get(&(fff0s_env->prf_env), conidx),
31                     dest_id, fff0s_fff1_writer_ind);
32                 // Fill in the parameter structure
33                 memcpy(ind->fff1_value, &param->value[0], param->length);
34                 ind->conidx = conidx;
35                 ind->length = param->length;
36                 // Send the message
37                 ke_msg_send(ind);
38             }
39             else if (att_idx == FFF0S_IDX_FFF2_LVL_VAL)
40             {
41                 // Allocate the alert value change indication

```

4. 在app_ff0.c文件下增加函数fff1_writer_req_handler并且添加到数组app_ff0_msg_handler_list[]中。

```

static int fff1_writer_req_handler(ke_msg_id_t const msgid,
                                   struct fff0s_fff1_writer_ind *param,
                                   ke_task_id_t const dest_id,
                                   ke_task_id_t const src_id)
{
    UART_PRINTF("param->length=%d\r\n", param->length);
    for(uint8_t i = 0; i < param->length; i++)
    {
        UART_PRINTF("fff1_value=%02x\r\n", param->fff1_value[i]); //将app透传发送的数据打印出来
    }
    return (KE_MSG_CONSUMED);
}

/// Default State handlers definition
const struct ke_msg_handler app_fff0_msg_handler_list[] =
{
    // Note: first message is latest message checked by kernel so default is put on top.
    {KE_MSG_DEFAULT_HANDLER, (ke_msg_func_t)app_fff0_msg_dflt_handler},
    {FFF0S_FFF1_LEVEL_NTF_CFG_IND, (ke_msg_func_t)fff0s_fff1_level_ntf_cfg_ind_handler},
    {FFF0S_FFF1_LEVEL_UPD_RSP, (ke_msg_func_t)fff1_level_upd_handler},
    {FFF0S_FFF1_WRITER_REQ_IND, (ke_msg_func_t)fff1_writer_req_handler},
    {FFF0S_FFF2_WRITER_REQ_IND, (ke_msg_func_t)fff2_writer_req_handler},
    {FFF0S_FFF1_LEVEL_PERIOD_NTF, (ke_msg_func_t)fff1_period_ntf_handler},
};

```

5. 修改完成后，ff0服务下的fff1通道不仅有原来read、notify的功能，也具有了fff2的write no response功能。

