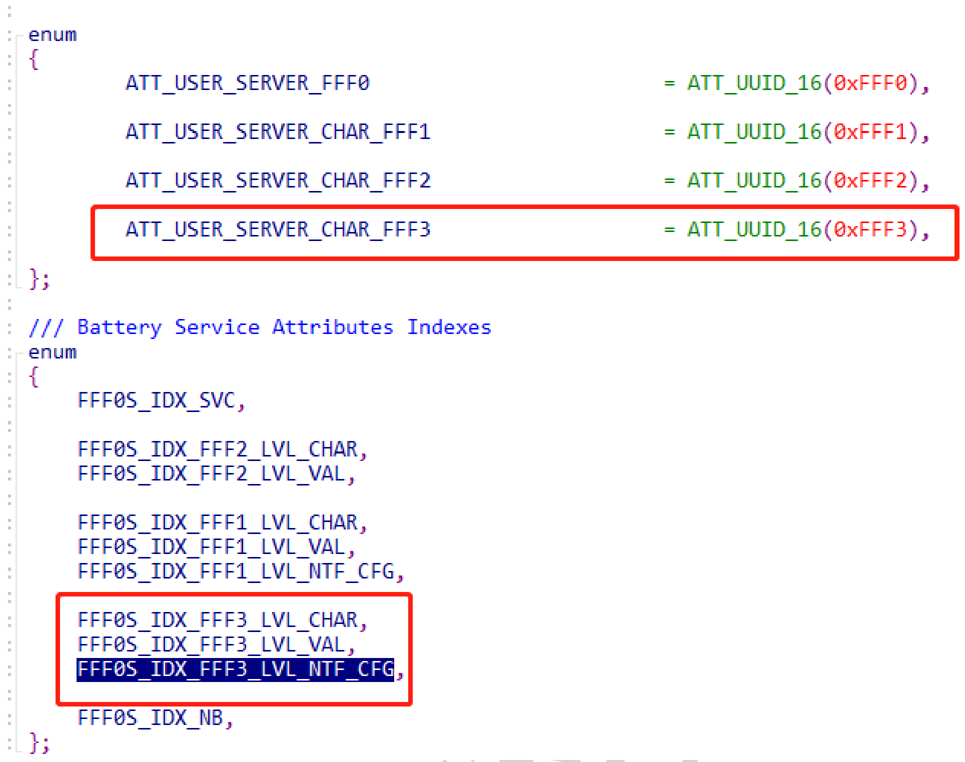
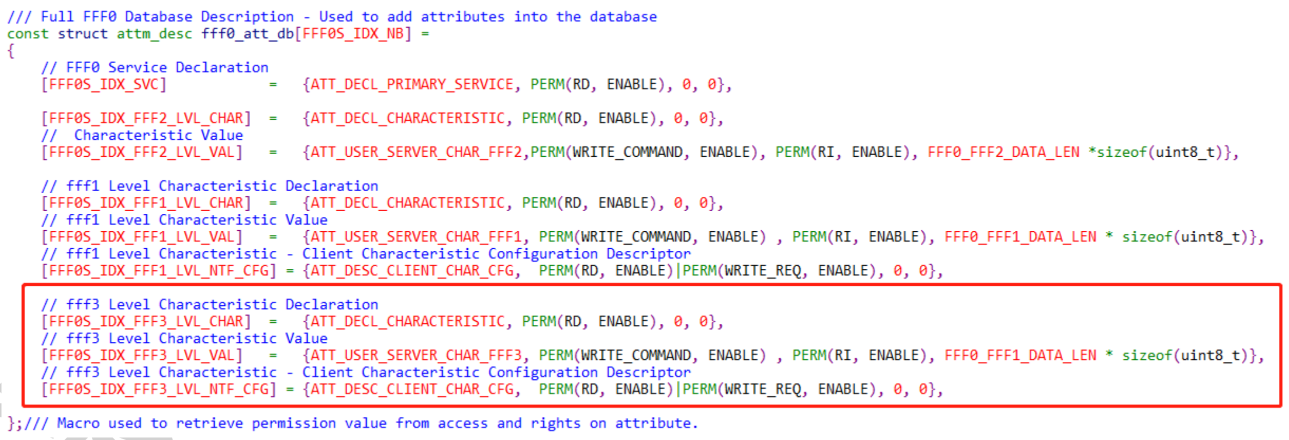
说明：该文档以FFF0服务下在增加一个与FFF1一样功能的通道FFF3为例子进行说明，客户可根据需要进行其他修改。

1. 在fff0s.h文件下添加枚举成员，如下图



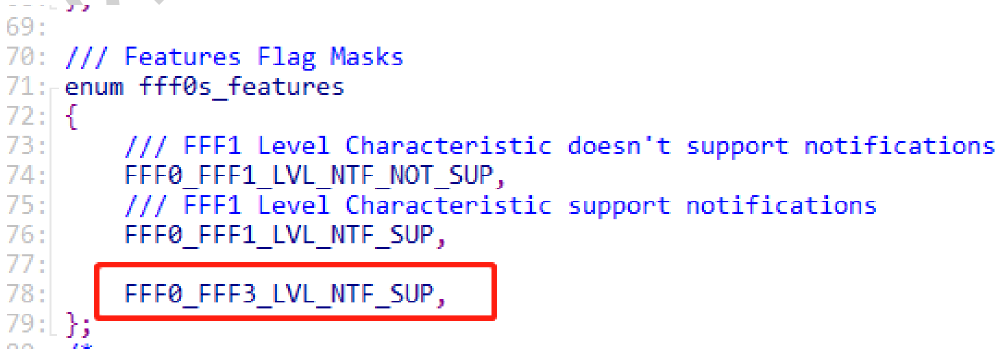
```
enum
{
        ATT_USER_SERVER_FFF0                        = ATT_UUID_16(0xFFF0),

        ATT_USER_SERVER_CHAR_FFF1                   = ATT_UUID_16(0xFFF1),

        ATT_USER_SERVER_CHAR_FFF2                   = ATT_UUID_16(0xFFF2),

        ATT_USER_SERVER_CHAR_FFF3                   = ATT_UUID_16(0xFFF3),
};

/// Battery Service Attributes Indexes
enum
{
    FFF0S_IDX_SVC,

    FFF0S_IDX_FFF2_LVL_CHAR,
    FFF0S_IDX_FFF2_LVL_VAL,

    FFF0S_IDX_FFF1_LVL_CHAR,
    FFF0S_IDX_FFF1_LVL_VAL,
    FFF0S_IDX_FFF1_LVL_NTF_CFG,

    FFF0S_IDX_FFF3_LVL_CHAR,
    FFF0S_IDX_FFF3_LVL_VAL,
    FFF0S_IDX_FFF3_LVL_NTF_CFG,

    FFF0S_IDX_NB,
};
```

2. 在fff0s.c文件下的fff0_att_db数组添加FFF3通道属性成员。



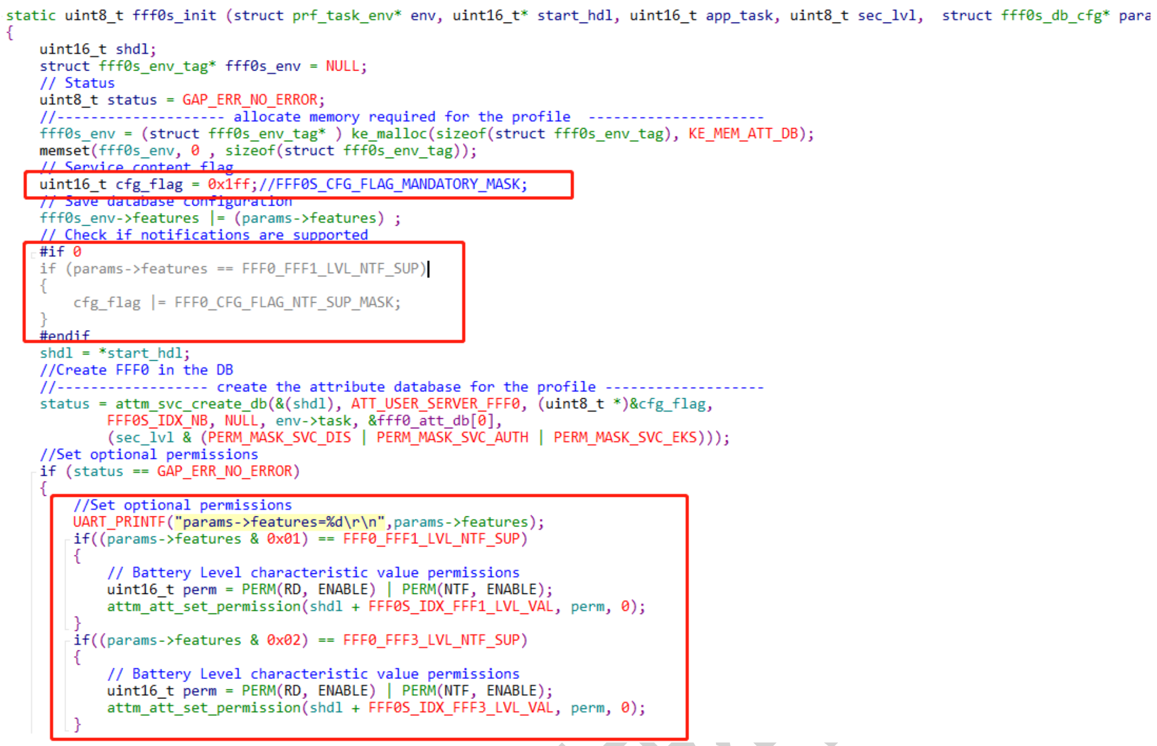
```
/// Full FFF0 Database Description - Used to add attributes into the database
const struct attm_desc fff0_att_db[FFF0S_IDX_NB] =
{
    // FFF0 Service Declaration
    [FFF0S_IDX_SVC]             =   {ATT_DECL_PRIMARY_SERVICE, PERM(RD, ENABLE), 0, 0},

    [FFF0S_IDX_FFF2_LVL_CHAR]  =   {ATT_DECL_CHARACTERISTIC, PERM(RD, ENABLE), 0, 0},
    //  Characteristic Value
    [FFF0S_IDX_FFF2_LVL_VAL]   =   {ATT_USER_SERVER_CHAR_FFF2,PERM(WRITE_COMMAND, ENABLE), PERM(RI, ENABLE), FFF0_FFF2_DATA_LEN *sizeof(uint8_t)},

    // fff1 Level Characteristic Declaration
    [FFF0S_IDX_FFF1_LVL_CHAR]  =   {ATT_DECL_CHARACTERISTIC, PERM(RD, ENABLE), 0, 0},
    // fff1 Level Characteristic Value
    [FFF0S_IDX_FFF1_LVL_VAL]   =   {ATT_USER_SERVER_CHAR_FFF1, PERM(WRITE_COMMAND, ENABLE) , PERM(RI, ENABLE), FFF0_FFF1_DATA_LEN * sizeof(uint8_t)},
    // fff1 Level Characteristic - Client Characteristic Configuration Descriptor
    [FFF0S_IDX_FFF1_LVL_NTF_CFG] = {ATT_DESC_CLIENT_CHAR_CFG,  PERM(RD, ENABLE)|PERM(WRITE_REQ, ENABLE), 0, 0},

    // fff3 Level Characteristic Declaration
    [FFF0S_IDX_FFF3_LVL_CHAR]  =   {ATT_DECL_CHARACTERISTIC, PERM(RD, ENABLE), 0, 0},
    // fff3 Level Characteristic Value
    [FFF0S_IDX_FFF3_LVL_VAL]   =   {ATT_USER_SERVER_CHAR_FFF3, PERM(WRITE_COMMAND, ENABLE) , PERM(RI, ENABLE), FFF0_FFF1_DATA_LEN * sizeof(uint8_t)},
    // fff3 Level Characteristic - Client Characteristic Configuration Descriptor
    [FFF0S_IDX_FFF3_LVL_NTF_CFG] = {ATT_DESC_CLIENT_CHAR_CFG,  PERM(RD, ENABLE)|PERM(WRITE_REQ, ENABLE), 0, 0},

};/// Macro used to retrieve permission value from access and rights on attribute.
```

3. 在fff0s_task.h文件下的fff0s_features添加枚举成员FFF0_FFF3_LVL_NTF_SUP



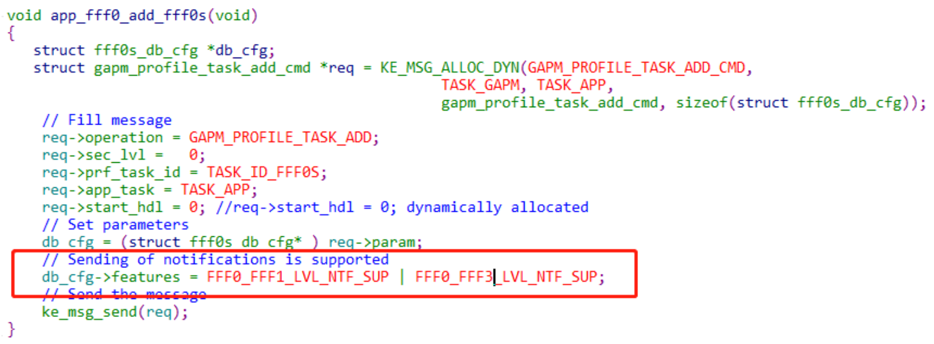
```

/// Features Flag Masks
enum fff0s_features
{
    /// FFF1 Level Characteristic doesn't support notifications
    FFF0_FFF1_LVL_NTF_NOT_SUP,
    /// FFF1 Level Characteristic support notifications
    FFF0_FFF1_LVL_NTF_SUP,

    FFF0_FFF3_LVL_NTF_SUP,
};
```

4. 在fff0s.h文件下的fff0s_init函数做如下图修改。



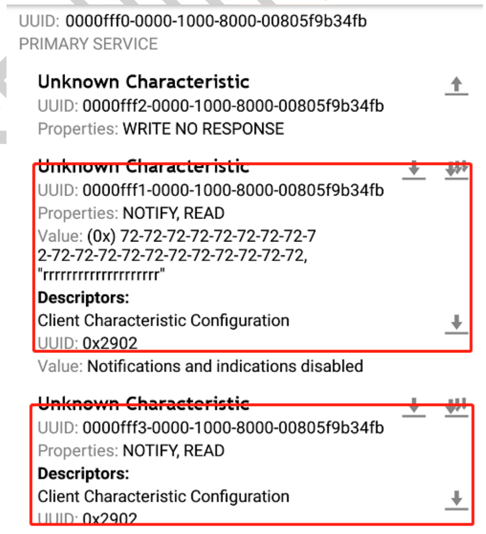
```
static uint8_t fff0s_init (struct prf_task_env* env, uint16_t* start_hdl, uint16_t app_task, uint8_t sec_lvl,  struct fff0s_db_cfg* para
{
    uint16_t shdl;
    struct fff0s_env_tag* fff0s_env = NULL;
    // Status
    uint8_t status = GAP_ERR_NO_ERROR;
    //-------------------- allocate memory 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) ;
    // Check if notifications are supported
    #if 0
    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
        UART_PRINTF("params->features=%d\r\n",params->features);
        if((params->features & 0x01) == FFF0_FFF1_LVL_NTF_SUP)
        {
            // Battery Level characteristic value permissions
            uint16_t perm = PERM(RD, ENABLE) | PERM(NTF, ENABLE);
            attm_att_set_permission(shdl + FFF0S_IDX_FFF1_LVL_VAL, perm, 0);
        }
        if((params->features & 0x02) == FFF0_FFF3_LVL_NTF_SUP)
        {
            // Battery Level characteristic value permissions
            uint16_t perm = PERM(RD, ENABLE) | PERM(NTF, ENABLE);
            attm_att_set_permission(shdl + FFF0S_IDX_FFF3_LVL_VAL, perm, 0);
        }
```

5. 在app_fff0.c文件下函数app_fff0_add_fff0s做如下图修改。



```
void app_fff0_add_fff0s(void)
{
   struct fff0s_db_cfg *db_cfg;
   struct gapm_profile_task_add_cmd *req = KE_MSG_ALLOC_DYN(GAPM_PROFILE_TASK_ADD_CMD,
                                                  TASK_GAPM, TASK_APP,
                                                  gapm_profile_task_add_cmd, sizeof(struct fff0s_db_cfg));

    // Fill message
    req->operation = GAPM_PROFILE_TASK_ADD;
    req->sec_lvl =   0;
    req->prf_task_id = TASK_ID_FFF0S;
    req->app_task = TASK_APP;
    req->start_hdl = 0; //req->start_hdl = 0; dynamically allocated
    // Set parameters
    db_cfg = (struct fff0s_db_cfg* ) req->param;
    // Sending of notifications is supported
    db_cfg->features = FFF0_FFF1_LVL_NTF_SUP | FFF0_FFF3_LVL_NTF_SUP;
    // Send the message
    ke_msg_send(req);
}
```

6. 修改完成后，下载程序。FFF3通道的初始化就已经完成了，可以看到和FFF1通道一样的图标，如下图。但是还不能notify监听接收数据。



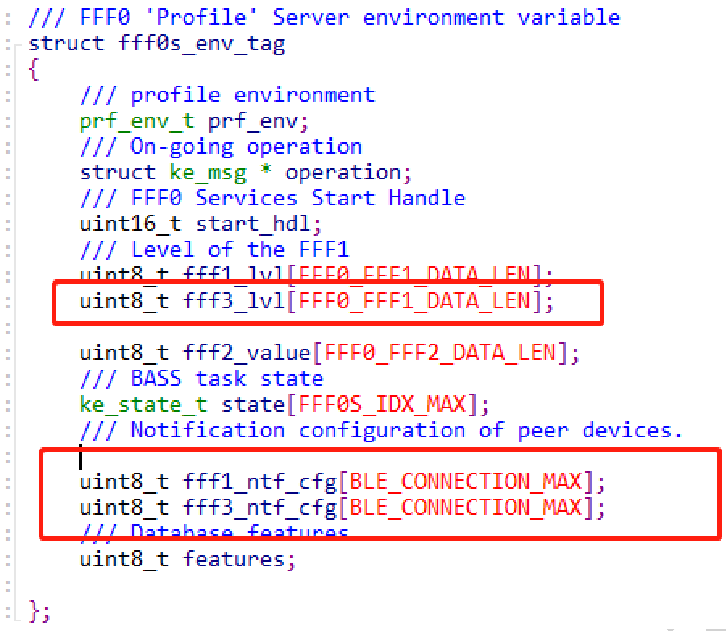
UUID: 0000fff0-0000-1000-8000-00805f9b34fb
PRIMARY SERVICE

Unknown Characteristic
UUID: 0000fff2-0000-1000-8000-00805f9b34fb
Properties: WRITE NO RESPONSE

Unknown Characteristic
UUID: 0000fff1-0000-1000-8000-00805f9b34fb
Properties: NOTIFY, READ
Value: (0x) 72-72-72-72-72-72-72-72-72-7
2-72-72-72-72-72-72-72-72-72-72-72,
"rrrrrrrrrrrrrrrrrrrr"
Descriptors:
Client Characteristic Configuration
UUID: 0x2902
Value: Notifications and indications disabled

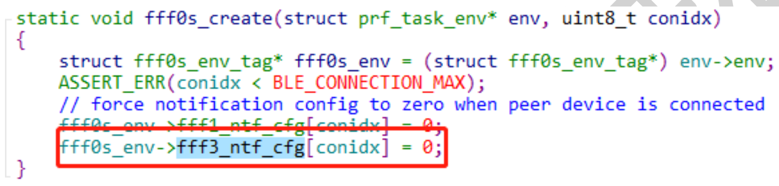
Unknown Characteristic
UUID: 0000fff3-0000-1000-8000-00805f9b34fb
Properties: NOTIFY, READ
Descriptors:
Client Characteristic Configuration
UUID: 0x2902

7. 对fff0s.h文件下的fff0s_env_tag结构体将ntf_cfg数组名修改成fff1_ntf_cfg，并且在程序里做对应修改；增加fff3_lvl、 fff3_ntf_cfg数组。

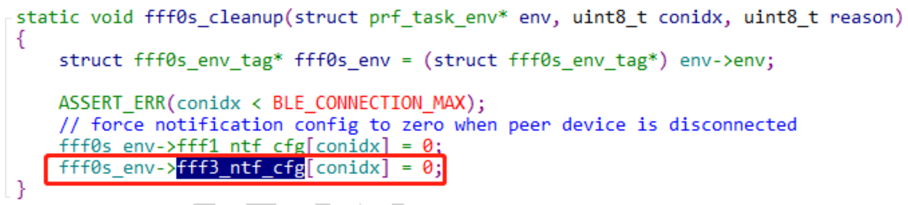


```
/// FFF0 'Profile' Server environment variable
struct fff0s_env_tag
{
    /// profile environment
    prf_env_t prf_env;
    /// On-going operation
    struct ke_msg * operation;
    /// FFF0 Services Start Handle
    uint16_t start_hdl;
    /// Level of the FFF1
    uint8_t fff1_lvl[FFF0_FFF1_DATA_LEN];
    uint8_t fff3_lvl[FFF0_FFF1_DATA_LEN];

    uint8_t fff2_value[FFF0_FFF2_DATA_LEN];
    /// BASS task state
    ke_state_t state[FFF0S_IDX_MAX];
    /// Notification configuration of peer devices.

    uint8_t fff1_ntf_cfg[BLE_CONNECTION_MAX];
    uint8_t fff3_ntf_cfg[BLE_CONNECTION_MAX];
    /// Database features
    uint8_t features;

};
```

8. 在fff0s.c文件下的fff0s_cleanup和fff0s_create函数对fff3_ntf_cfg进行清零

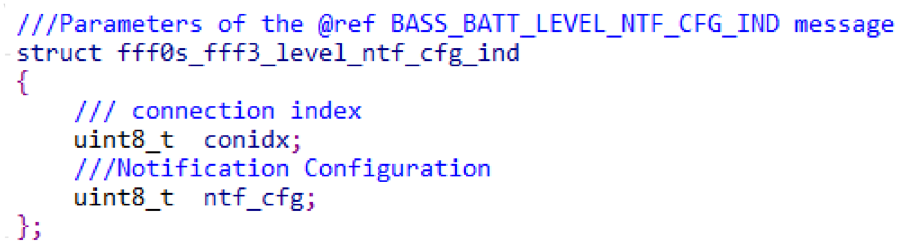


```
static void fff0s_create(struct prf_task_env* env, uint8_t conidx)
{
    struct fff0s_env_tag* fff0s_env = (struct fff0s_env_tag*) env->env;
    ASSERT_ERR(conidx < BLE_CONNECTION_MAX);
    // force notification config to zero when peer device is connected
    fff0s_env->fff1_ntf_cfg[conidx] = 0;
    fff0s_env->fff3_ntf_cfg[conidx] = 0;
}
```

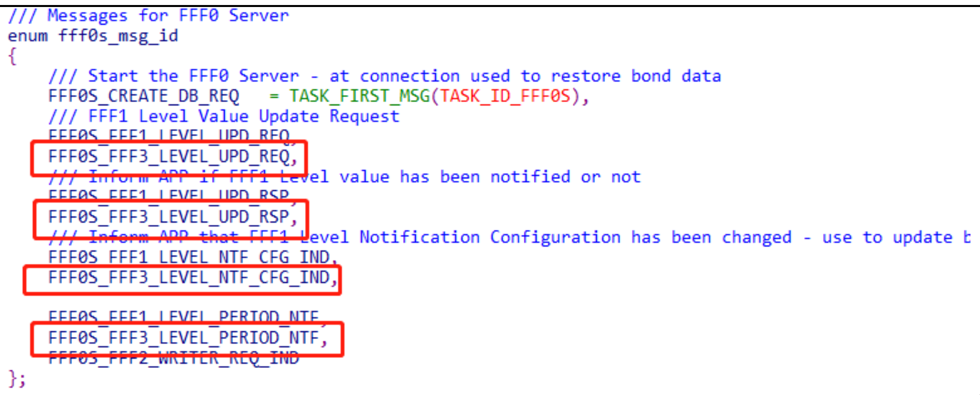


```
static void fff0s_cleanup(struct prf_task_env* env, uint8_t conidx, uint8_t reason)
{
    struct fff0s_env_tag* fff0s_env = (struct fff0s_env_tag*) env->env;

    ASSERT_ERR(conidx < BLE_CONNECTION_MAX);
    // force notification config to zero when peer device is disconnected
    fff0s_env->fff1_ntf_cfg[conidx] = 0;
    fff0s_env->fff3_ntf_cfg[conidx] = 0;
}
```

9. 在fff0s_task.h文件下增加结构体fff0s_fff3_level_ntf_cfg_ind和在枚举fff0s_msg_id增加枚举成员FFF0S_FFF3_LEVEL_NTF_CFG_IND、FFF0S_FFF3_LEVEL_PERIOD_NTF、FFF0S_FFF3_LEVEL_UPD_REQ和FFF0S_FFF3_LEVEL_UPD_RSP。

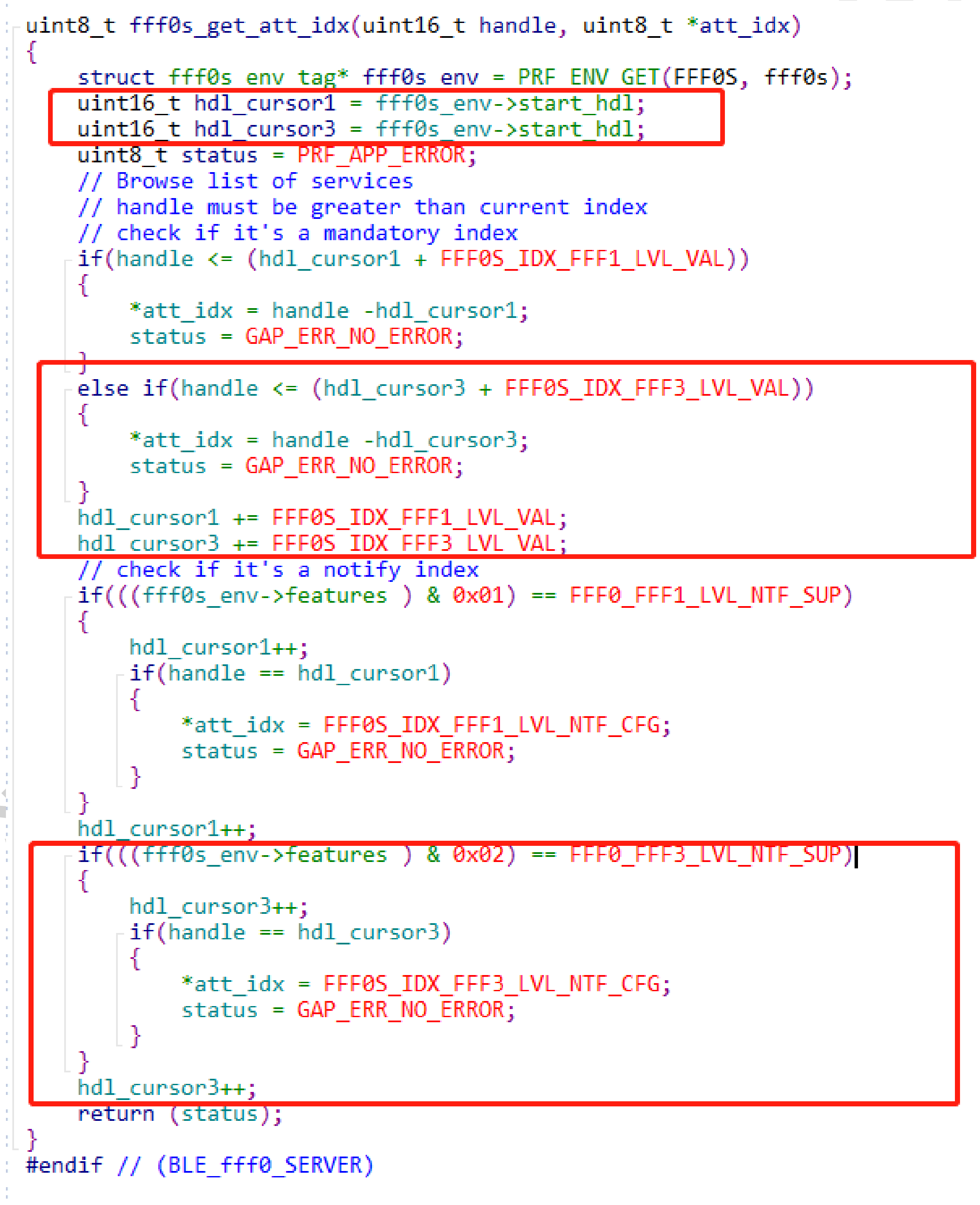


```
///Parameters of the @ref BASS_BATT_LEVEL_NTF_CFG_IND message
struct fff0s_fff3_level_ntf_cfg_ind
{
    /// connection index
    uint8_t  conidx;
    ///Notification Configuration
    uint8_t  ntf_cfg;
};
```



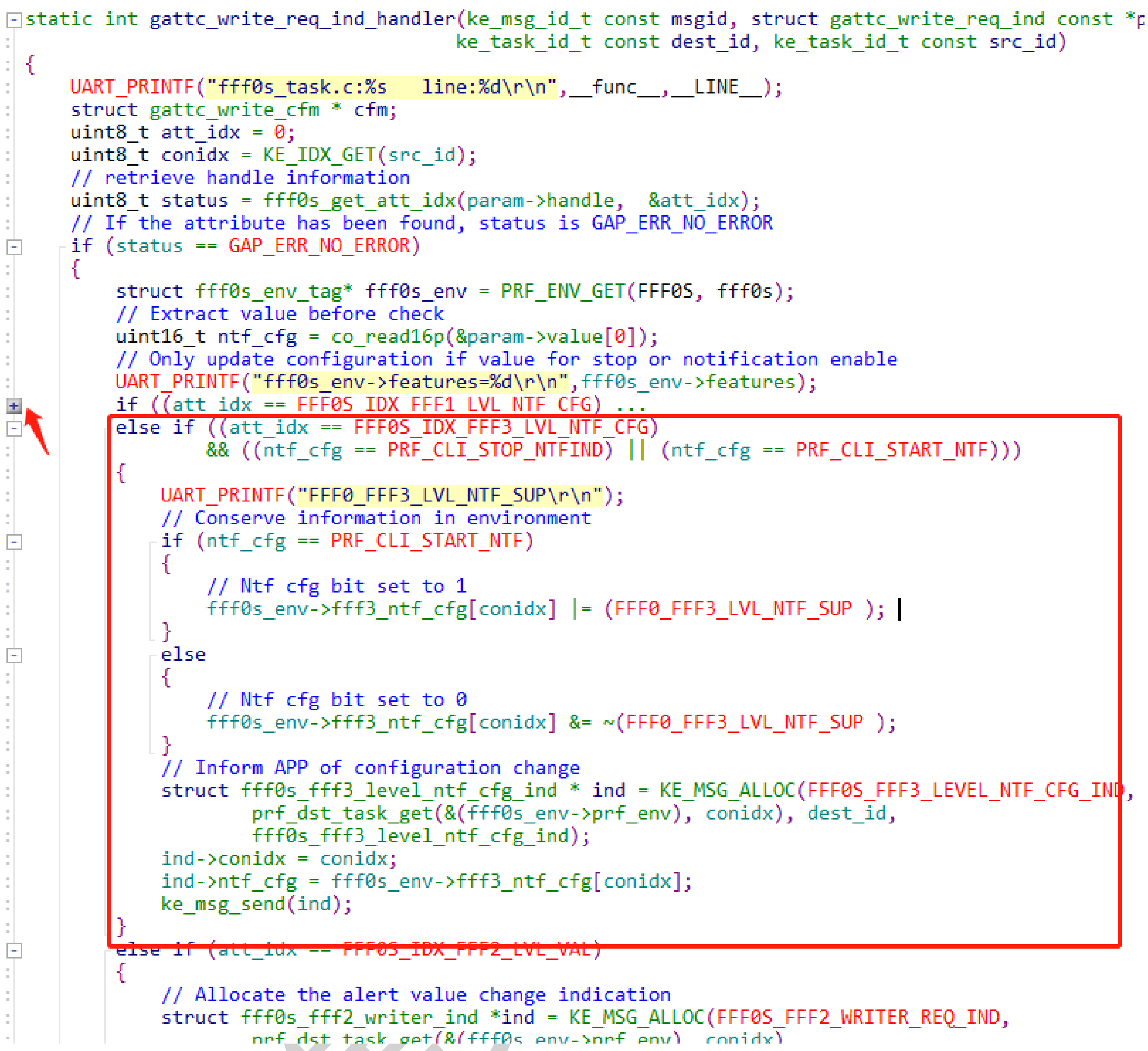
```
/// Messages for FFF0 Server
enum fff0s_msg_id
{
    /// Start the FFF0 Server - at connection used to restore bond data
    FFF0S_CREATE_DB_REQ    = TASK_FIRST_MSG(TASK_ID_FFF0S),
    /// FFF1 Level Value Update Request
    FFF0S_FFF1_LEVEL_UPD_REQ,
    FFF0S_FFF3_LEVEL_UPD_REQ,
    /// Inform APP if FFF1 Level value has been notified or not
    FFF0S_FFF1_LEVEL_UPD_RSP,
    FFF0S_FFF3_LEVEL_UPD_RSP,
    /// Inform APP that FFF1 Level Notification Configuration has been changed - use to update b
    FFF0S_FFF1_LEVEL_NTF_CFG_IND,
    FFF0S_FFF3_LEVEL_NTF_CFG_IND,

    FFF0S_FFF1_LEVEL_PERIOD_NTF,
    FFF0S_FFF3_LEVEL_PERIOD_NTF,
    FFF0S_FFF2_WRITER_REQ_IND
};
```

10. 对在fff0s.c文件下的fff0s_get_att_idx函数做如下图修改。



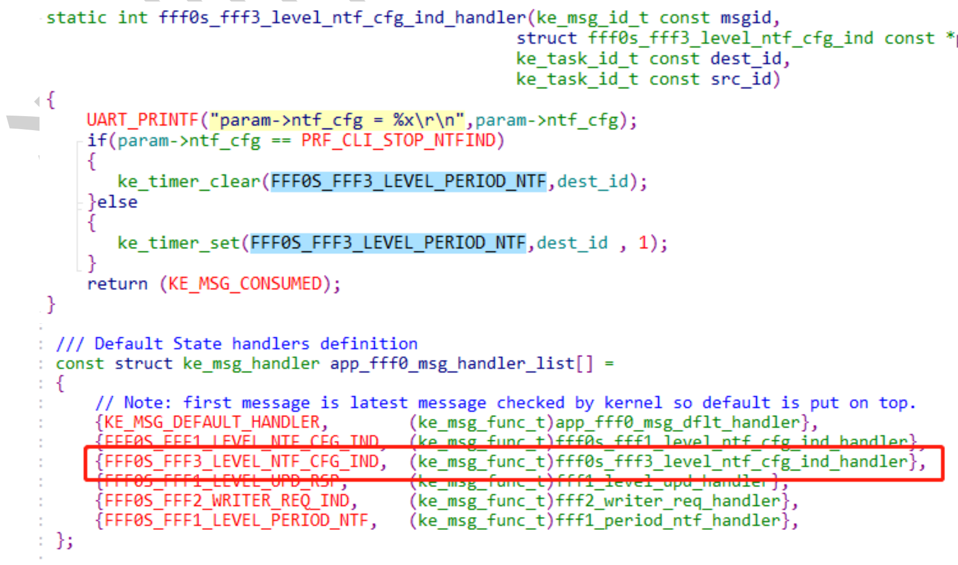
```
uint8_t fff0s_get_att_idx(uint16_t handle, uint8_t *att_idx)
{
    struct fff0s_env_tag* fff0s_env = PRF_ENV_GET(FFF0S, fff0s);
    uint16_t hdl_cursor1 = fff0s_env->start_hdl;
    uint16_t hdl_cursor3 = fff0s_env->start_hdl;
    uint8_t status = PRF_APP_ERROR;
    // Browse list of services
    // handle must be greater than current index
    // check if it's a mandatory index
    if(handle <= (hdl_cursor1 + FFF0S_IDX_FFF1_LVL_VAL))
    {
        *att_idx = handle -hdl_cursor1;
        status = GAP_ERR_NO_ERROR;
    }
    else if(handle <= (hdl_cursor3 + FFF0S_IDX_FFF3_LVL_VAL))
    {
        *att_idx = handle -hdl_cursor3;
        status = GAP_ERR_NO_ERROR;
    }
    hdl_cursor1 += FFF0S_IDX_FFF1_LVL_VAL;
    hdl_cursor3 += FFF0S_IDX_FFF3_LVL_VAL;
    // check if it's a notify index
    if(((fff0s_env->features ) & 0x01) == FFF0_FFF1_LVL_NTF_SUP)
    {
        hdl_cursor1++;
        if(handle == hdl_cursor1)
        {
            *att_idx = FFF0S_IDX_FFF1_LVL_NTF_CFG;
            status = GAP_ERR_NO_ERROR;
        }
    }
    hdl_cursor1++;
    if(((fff0s_env->features ) & 0x02) == FFF0_FFF3_LVL_NTF_SUP)
    {
        hdl_cursor3++;
        if(handle == hdl_cursor3)
        {
            *att_idx = FFF0S_IDX_FFF3_LVL_NTF_CFG;
            status = GAP_ERR_NO_ERROR;
        }
    }
    hdl_cursor3++;
    return (status);
}
#endif // (BLE_fff0_SERVER)
```

11. 在fff0s_task.c文件下的gattc_write_req_ind_handler函数添加程序，参照下图。



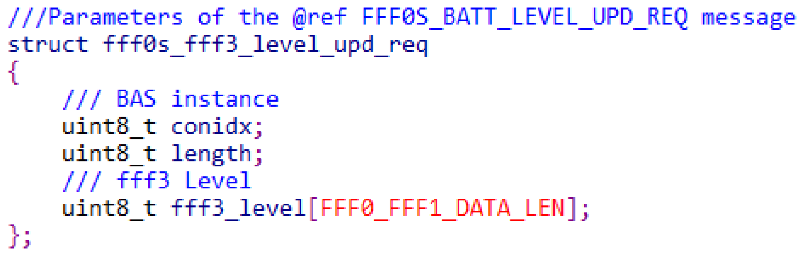
```
static int gattc_write_req_ind_handler(ke_msg_id_t const msgid, struct gattc_write_req_ind const *p
                                       ke_task_id_t const dest_id, ke_task_id_t const src_id)
{
    UART_PRINTF("fff0s_task.c:%s   line:%d\r\n",__func__,__LINE__);
    struct gattc_write_cfm * cfm;
    uint8_t att_idx = 0;
    uint8_t conidx = KE_IDX_GET(src_id);
    // retrieve handle information
    uint8_t status = fff0s_get_att_idx(param->handle,  &att_idx);
    // If the attribute has been found, status is GAP_ERR_NO_ERROR
    if (status == GAP_ERR_NO_ERROR)
    {
        struct fff0s_env_tag* fff0s_env = PRF_ENV_GET(FFF0S, fff0s);
        // Extract value before check
        uint16_t ntf_cfg = co_read16p(&param->value[0]);
        // Only update configuration if value for stop or notification enable
        UART_PRINTF("fff0s_env->features=%d\r\n",fff0s_env->features);
        if ((att_idx == FFF0S_IDX_FFF1_LVL_NTF_CFG) ...
        else if ((att_idx == FFF0S_IDX_FFF3_LVL_NTF_CFG)
                && ((ntf_cfg == PRF_CLI_STOP_NTFIND) || (ntf_cfg == PRF_CLI_START_NTF)))
        {
            UART_PRINTF("FFF0_FFF3_LVL_NTF_SUP\r\n");
            // Conserve information in environment
            if (ntf_cfg == PRF_CLI_START_NTF)
            {
                // Ntf cfg bit set to 1
                fff0s_env->fff3_ntf_cfg[conidx] |= (FFF0_FFF3_LVL_NTF_SUP );
            }
            else
            {
                // Ntf cfg bit set to 0
                fff0s_env->fff3_ntf_cfg[conidx] &= ~(FFF0_FFF3_LVL_NTF_SUP );
            }
            // Inform APP of configuration change
            struct fff0s_fff3_level_ntf_cfg_ind * ind = KE_MSG_ALLOC(FFF0S_FFF3_LEVEL_NTF_CFG_IND,
                    prf_dst_task_get(&(fff0s_env->prf_env), conidx), dest_id,
                    fff0s_fff3_level_ntf_cfg_ind);
            ind->conidx = conidx;
            ind->ntf_cfg = fff0s_env->fff3_ntf_cfg[conidx];
            ke_msg_send(ind);
        }
        else if (att_idx == FFF0S_IDX_FFF2_LVL_VAL)
        {
            // Allocate the alert value change indication
            struct fff0s_fff2_writer_ind *ind = KE_MSG_ALLOC(FFF0S_FFF2_WRITER_REQ_IND,
                    prf_dst_task_get(&(fff0s_env->prf_env), conidx),
```

12. 在app_fff0.c添加函数fff0s_fff3_level_ntf_cfg_ind_handler和在app_fff0_msg_handler_list数组添加fff0s_fff3_level_ntf_cfg_ind_handler函数的回调。



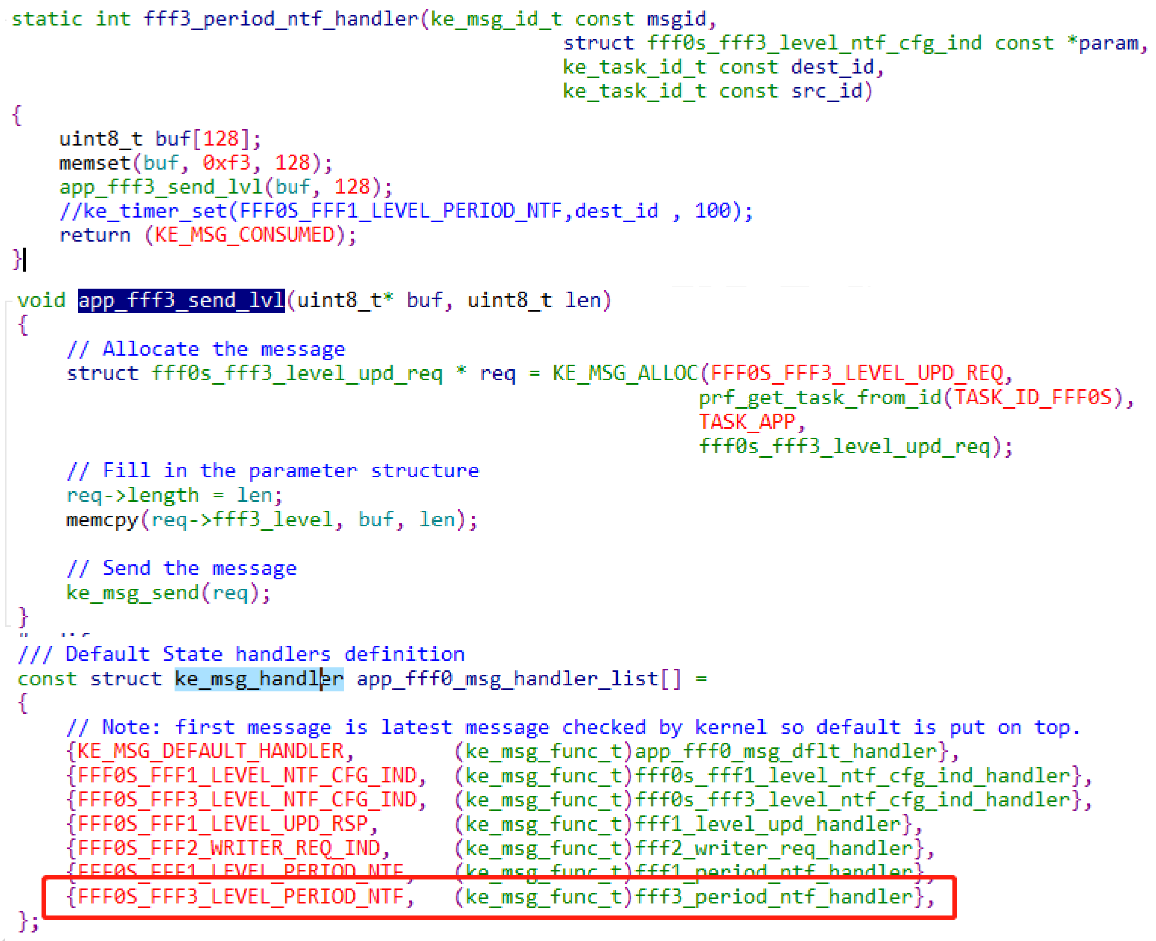
```
static int fff0s_fff3_level_ntf_cfg_ind_handler(ke_msg_id_t const msgid,
                                                struct fff0s_fff3_level_ntf_cfg_ind const *param,
                                                ke_task_id_t const dest_id,
                                                ke_task_id_t const src_id)

{
    UART_PRINTF("param->ntf_cfg = %x\r\n",param->ntf_cfg);
    if(param->ntf_cfg == PRF_CLI_STOP_NTFIND)
    {
        ke_timer_clear(FFF0S_FFF3_LEVEL_PERIOD_NTF,dest_id);
    }else
    {
        ke_timer_set(FFF0S_FFF3_LEVEL_PERIOD_NTF,dest_id , 1);
    }
    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_FFF3_LEVEL_NTF_CFG_IND,   (ke_msg_func_t)fff0s_fff3_level_ntf_cfg_ind_handler},
    {FFF0S_FFF1_LEVEL_UPD_RSP,       (ke_msg_func_t)fff1_level_upd_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},
};
```

13. 在fff0s._task.h文件下添加结构体fff0s_fff3_level_upd_req



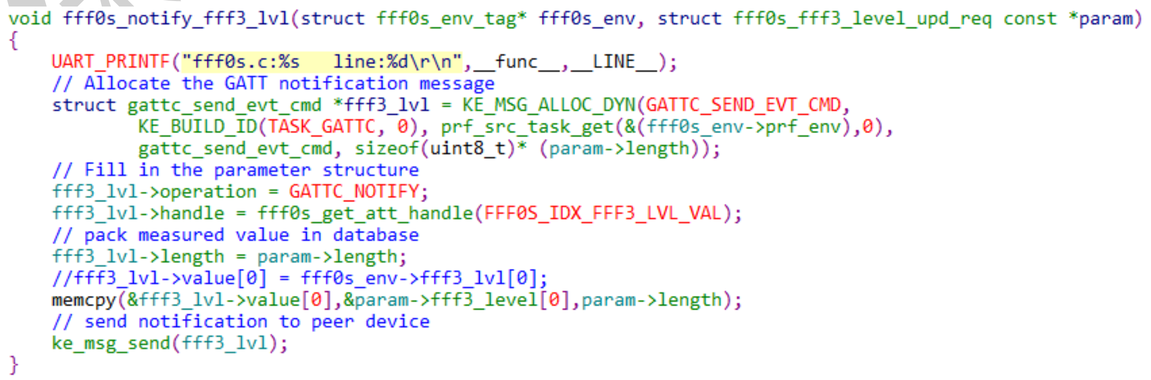
```
///Parameters of the @ref FFF0S_BATT_LEVEL_UPD_REQ message
struct fff0s_fff3_level_upd_req
{
    /// BAS instance
    uint8_t conidx;
    uint8_t length;
    /// fff3 Level
    uint8_t fff3_level[FFF0_FFF1_DATA_LEN];
};
```

14. 在app_fff0.c添加函数fff3_period_ntf_handler、app_fff3_send_lvl和在app_fff0_msg_handler_list数组添加fff3_period_ntf_handler函数的回调。



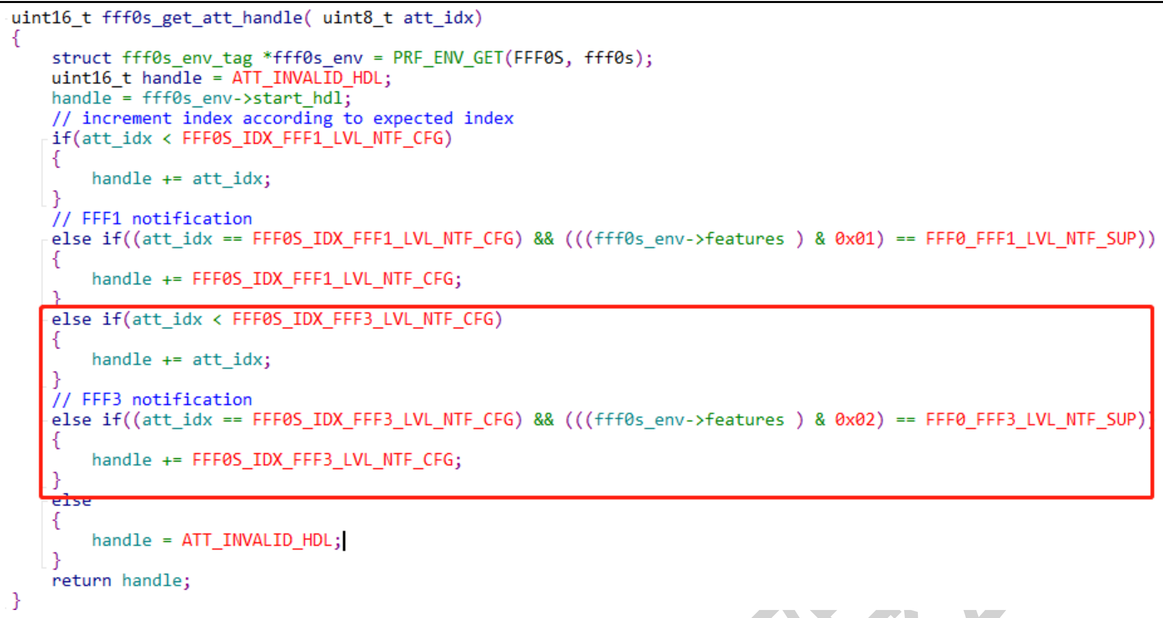
```
static int fff3_period_ntf_handler(ke_msg_id_t const msgid,
                                           struct fff0s_fff3_level_ntf_cfg_ind const *param,
                                           ke_task_id_t const dest_id,
                                           ke_task_id_t const src_id)
{
    uint8_t buf[128];
    memset(buf, 0xf3, 128);
    app_fff3_send_lvl(buf, 128);
    //ke_timer_set(FFF0S_FFF1_LEVEL_PERIOD_NTF,dest_id , 100);
    return (KE_MSG_CONSUMED);
}

void app_fff3_send_lvl(uint8_t* buf, uint8_t len)
{
    // Allocate the message
    struct fff0s_fff3_level_upd_req * req = KE_MSG_ALLOC(FFF0S_FFF3_LEVEL_UPD_REQ,
                                                        prf_get_task_from_id(TASK_ID_FFF0S),
                                                        TASK_APP,
                                                        fff0s_fff3_level_upd_req);

    // Fill in the parameter structure
    req->length = len;
    memcpy(req->fff3_level, buf, len);

    // Send the message
    ke_msg_send(req);
}

/// 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_FFF3_LEVEL_NTF_CFG_IND,  (ke_msg_func_t)fff0s_fff3_level_ntf_cfg_ind_handler},
    {FFF0S_FFF1_LEVEL_UPD_RSP,      (ke_msg_func_t)fff1_level_upd_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},
    {FFF0S_FFF3_LEVEL_PERIOD_NTF,   (ke_msg_func_t)fff3_period_ntf_handler},
};
```

15. 在fff0s.c文件下添加函数fff0s_notify_fff3_lvl，并在fff0s.h做外部声明。



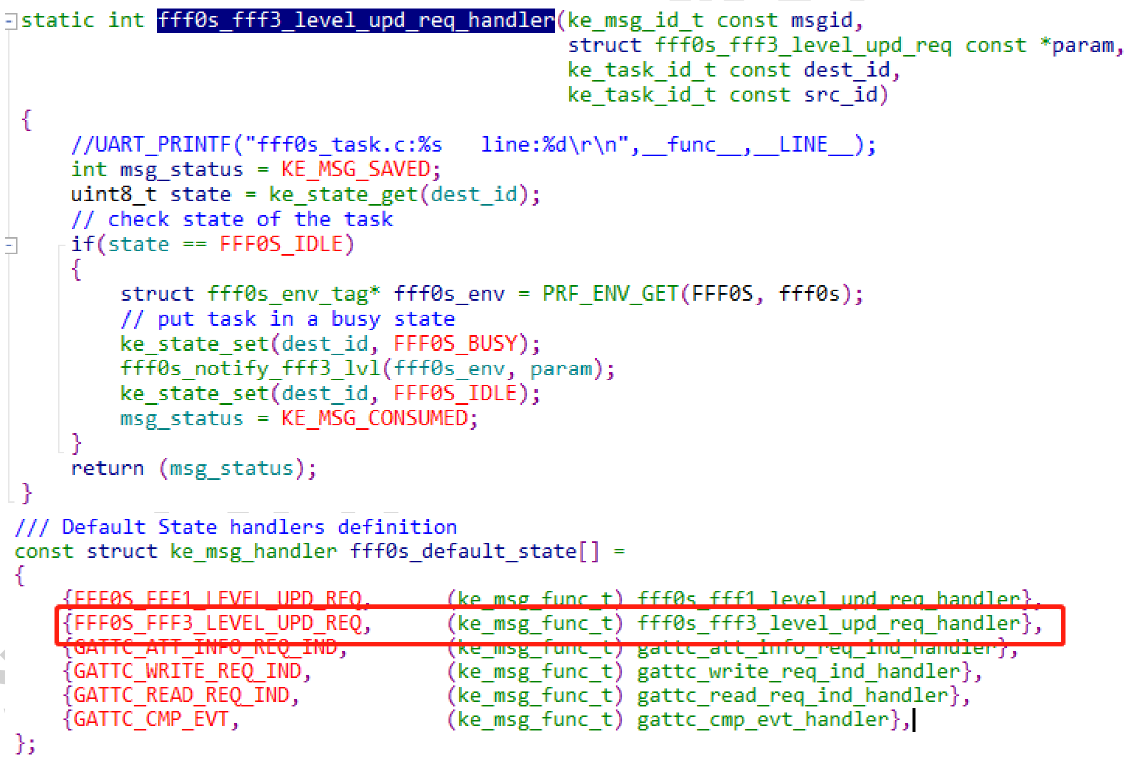
```
void fff0s_notify_fff3_lvl(struct fff0s_env_tag* fff0s_env, struct fff0s_fff3_level_upd_req const *param)
{
    UART_PRINTF("fff0s.c:%s   line:%d\r\n",__func__,__LINE__);
    // Allocate the GATT notification message
    struct gattc_send_evt_cmd *fff3_lvl = KE_MSG_ALLOC_DYN(GATTC_SEND_EVT_CMD,
            KE_BUILD_ID(TASK_GATTC, 0), prf_src_task_get(&(fff0s_env->prf_env),0),
            gattc_send_evt_cmd, sizeof(uint8_t)* (param->length));
    // Fill in the parameter structure
    fff3_lvl->operation = GATTC_NOTIFY;
    fff3_lvl->handle = fff0s_get_att_handle(FFF0S_IDX_FFF3_LVL_VAL);
    // pack measured value in database
    fff3_lvl->length = param->length;
    //fff3_lvl->value[0] = fff0s_env->fff3_lvl[0];
    memcpy(&fff3_lvl->value[0],&param->fff3_level[0],param->length);
    // send notification to peer device
    ke_msg_send(fff3_lvl);
}
```

16. 在fff0s.c文件下添加函数fff0s_get_att_handle做如下图修改。



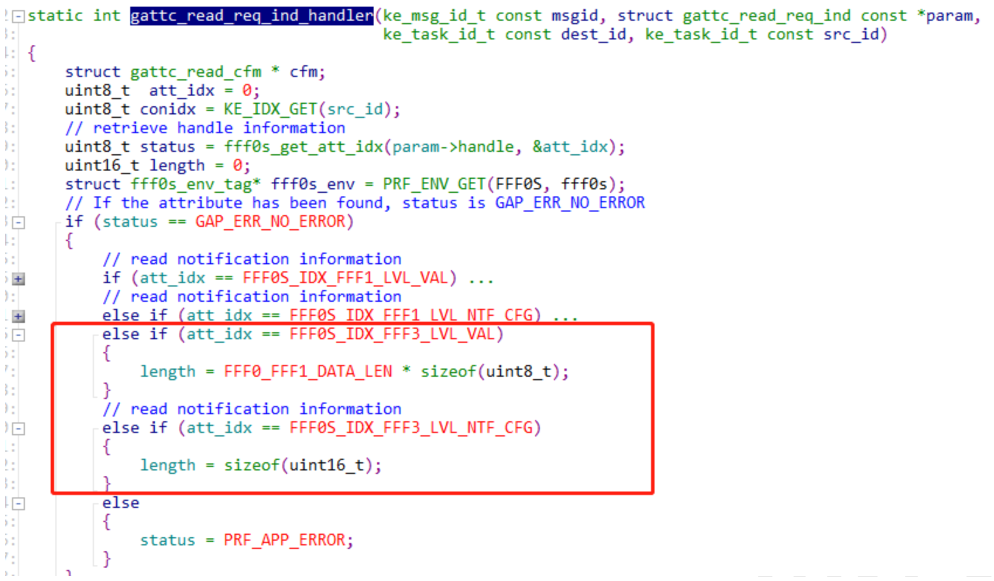
```
uint16_t fff0s_get_att_handle( uint8_t att_idx)
{
    struct fff0s_env_tag *fff0s_env = PRF_ENV_GET(FFF0S, fff0s);
    uint16_t handle = ATT_INVALID_HDL;
    handle = fff0s_env->start_hdl;
    // increment index according to expected index
    if(att_idx < FFF0S_IDX_FFF1_LVL_NTF_CFG)
    {
        handle += att_idx;
    }
    // FFF1 notification
    else if((att_idx == FFF0S_IDX_FFF1_LVL_NTF_CFG) && (((fff0s_env->features ) & 0x01) == FFF0_FFF1_LVL_NTF_SUP))
    {
        handle += FFF0S_IDX_FFF1_LVL_NTF_CFG;
    }
    else if(att_idx < FFF0S_IDX_FFF3_LVL_NTF_CFG)
    {
        handle += att_idx;
    }
    // FFF3 notification
    else if((att_idx == FFF0S_IDX_FFF3_LVL_NTF_CFG) && (((fff0s_env->features ) & 0x02) == FFF0_FFF3_LVL_NTF_SUP))
    {
        handle += FFF0S_IDX_FFF3_LVL_NTF_CFG;
    }
    else
    {
        handle = ATT_INVALID_HDL;
    }
    return handle;
}
```

17. 在fff0s_task.c文件下添加函数fff0s_fff3_level_upd_req_handler并且在fff0s_default_state数组添加回调。

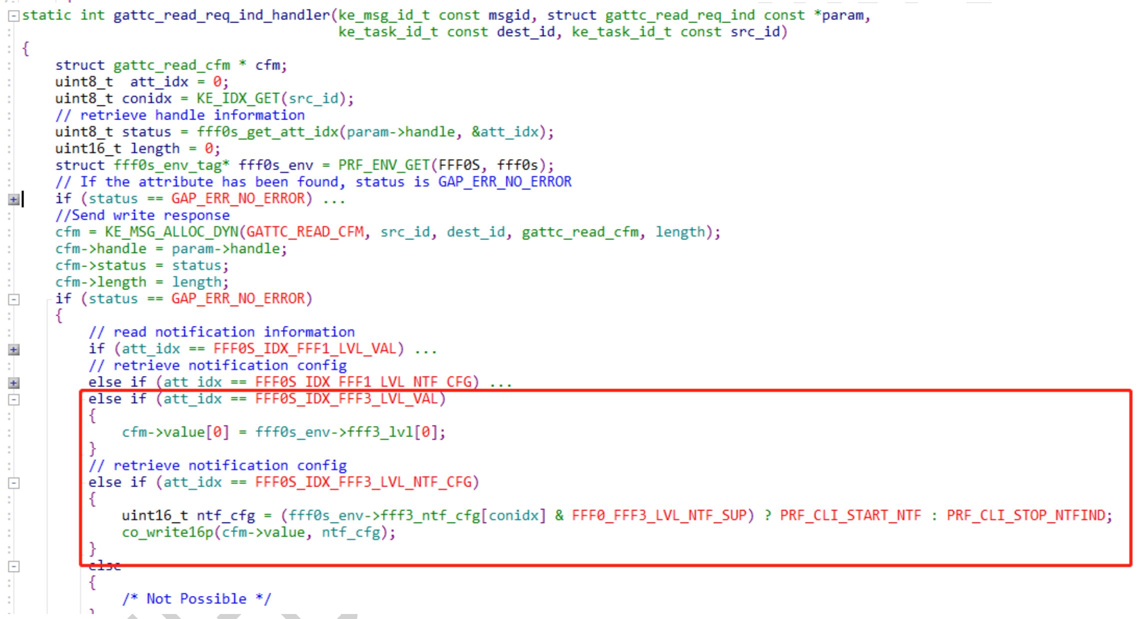


```
static int fff0s_fff3_level_upd_req_handler(ke_msg_id_t const msgid,
                                            struct fff0s_fff3_level_upd_req const *param,
                                            ke_task_id_t const dest_id,
                                            ke_task_id_t const src_id)
{
    //UART_PRINTF("fff0s_task.c:%s   line:%d\r\n",__func__,__LINE__);
    int msg_status = KE_MSG_SAVED;
    uint8_t state = ke_state_get(dest_id);
    // check state of the task
    if(state == FFF0S_IDLE)
    {
        struct fff0s_env_tag* fff0s_env = PRF_ENV_GET(FFF0S, fff0s);
        // put task in a busy state
        ke_state_set(dest_id, FFF0S_BUSY);
        fff0s_notify_fff3_lvl(fff0s_env, param);
        ke_state_set(dest_id, FFF0S_IDLE);
        msg_status = KE_MSG_CONSUMED;
    }
    return (msg_status);
}
/// Default State handlers definition
const struct ke_msg_handler fff0s_default_state[] =
{
    {FFF0S_FFF1_LEVEL_UPD_REQ,      (ke_msg_func_t) fff0s_fff1_level_upd_req_handler},
    {FFF0S_FFF3_LEVEL_UPD_REQ,      (ke_msg_func_t) fff0s_fff3_level_upd_req_handler},
    {GATTC_ATT_INFO_REQ_IND,        (ke_msg_func_t) gattc_att_info_req_ind_handler},
    {GATTC_WRITE_REQ_IND,           (ke_msg_func_t) gattc_write_req_ind_handler},
    {GATTC_READ_REQ_IND,            (ke_msg_func_t) gattc_read_req_ind_handler},
    {GATTC_CMP_EVT,                 (ke_msg_func_t) gattc_cmp_evt_handler},
};
```

18. 在fff0s_task.c文件下函数gattc_read_req_ind_handler添加程序

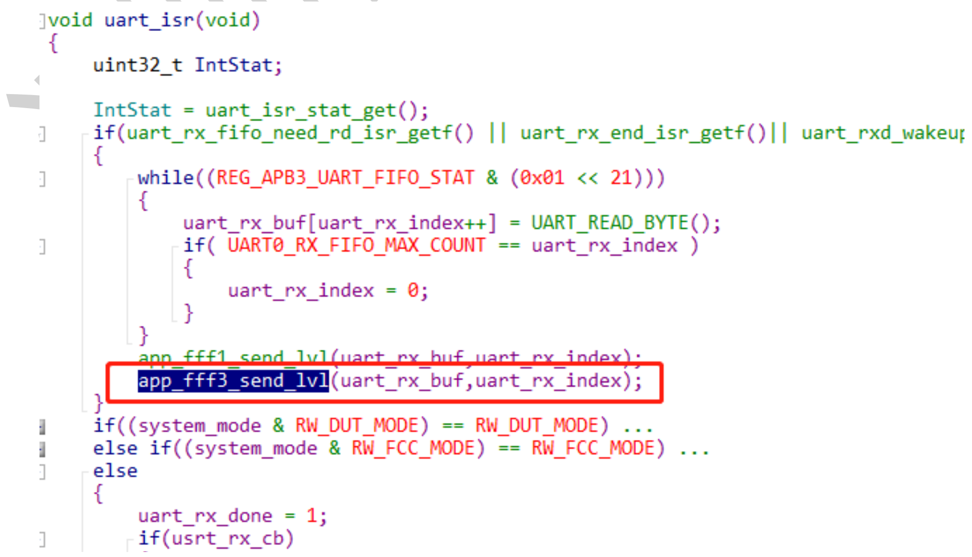


```
static int gattc_read_req_ind_handler(ke_msg_id_t const msgid, struct gattc_read_req_ind const *param,
                                      ke_task_id_t const dest_id, ke_task_id_t const src_id)
{
    struct gattc_read_cfm * cfm;
    uint8_t  att_idx = 0;
    uint8_t conidx = KE_IDX_GET(src_id);
    // retrieve handle information
    uint8_t status = fff0s_get_att_idx(param->handle, &att_idx);
    uint16_t length = 0;
    struct fff0s_env_tag* fff0s_env = PRF_ENV_GET(FFF0S, fff0s);
    // If the attribute has been found, status is GAP_ERR_NO_ERROR
    if (status == GAP_ERR_NO_ERROR)
    {
        // read notification information
        if (att_idx == FFF0S_IDX_FFF1_LVL_VAL) ...
        // read notification information
        else if (att_idx == FFF0S_IDX_FFF1_LVL_NTF_CFG) ...
        else if (att_idx == FFF0S_IDX_FFF3_LVL_VAL)
        {
            length = FFF0_FFF1_DATA_LEN * sizeof(uint8_t);
        }
        // read notification information
        else if (att_idx == FFF0S_IDX_FFF3_LVL_NTF_CFG)
        {
            length = sizeof(uint16_t);
        }
        else
        {
            status = PRF_APP_ERROR;
        }
```



```
static int gattc_read_req_ind_handler(ke_msg_id_t const msgid, struct gattc_read_req_ind const *param,
                                      ke_task_id_t const dest_id, ke_task_id_t const src_id)
{
    struct gattc_read_cfm * cfm;
    uint8_t  att_idx = 0;
    uint8_t conidx = KE_IDX_GET(src_id);
    // retrieve handle information
    uint8_t status = fff0s_get_att_idx(param->handle, &att_idx);
    uint16_t length = 0;
    struct fff0s_env_tag* fff0s_env = PRF_ENV_GET(FFF0S, fff0s);
    // If the attribute has been found, status is GAP_ERR_NO_ERROR
    if (status == GAP_ERR_NO_ERROR) ...
    //Send write response
    cfm = KE_MSG_ALLOC_DYN(GATTC_READ_CFM, src_id, dest_id, gattc_read_cfm, length);
    cfm->handle = param->handle;
    cfm->status = status;
    cfm->length = length;
    if (status == GAP_ERR_NO_ERROR)
    {
        // read notification information
        if (att_idx == FFF0S_IDX_FFF1_LVL_VAL) ...
        // retrieve notification config
        else if (att_idx == FFF0S_IDX_FFF1_LVL_NTF_CFG) ...
        else if (att_idx == FFF0S_IDX_FFF3_LVL_VAL)
        {
            cfm->value[0] = fff0s_env->fff3_lvl[0];
        }
        // retrieve notification config
        else if (att_idx == FFF0S_IDX_FFF3_LVL_NTF_CFG)
        {
            uint16_t ntf_cfg = (fff0s_env->fff3_ntf_cfg[conidx] & FFF0_FFF3_LVL_NTF_SUP) ? PRF_CLI_START_NTF : PRF_CLI_STOP_NTFIND;
            co_write16p(cfm->value, ntf_cfg);
        }
        else
        {
            /* Not Possible */
```

19. 在uart.c文件下的uart_isr中断函数调用app_fff3_send_lvl函数。



```
void uart_isr(void)
{
    uint32_t IntStat;

    IntStat = uart_isr_stat_get();
    if(uart_rx_fifo_need_rd_isr_getf() || uart_rx_end_isr_getf()|| uart_rxd_wakeup
    {
        while((REG_APB3_UART_FIFO_STAT & (0x01 << 21)))
        {
            uart_rx_buf[uart_rx_index++] = UART_READ_BYTE();
            if( UART0_RX_FIFO_MAX_COUNT == uart_rx_index )
            {
                uart_rx_index = 0;
            }
        }
        app_fff1_send_lvl(uart_rx_buf,uart_rx_index);
        app_fff3_send_lvl(uart_rx_buf,uart_rx_index);
    }
    if((system_mode & RW_DUT_MODE) == RW_DUT_MODE) ...
    else if((system_mode & RW_FCC_MODE) == RW_FCC_MODE) ...
    else
    {
        uart_rx_done = 1;
        if(usrt_rx_cb)
```

20. 编译下载程序，FFF3通道便具有和FFF1通道一样的功能。