Custom_Msg 通信协议应用描述

Custom_Msg 通信协议是一种用于 Linux-Host 与 AIC-MCU 之间数据传输、交互控制的通信协议。

```
        Linux-host1
          ctl-app          Custom_Msg
             |    <-------------------------->
      ioctl  |    ^            ^             ^
      (sock) |    |            ^             |
             v    v         USB/SDIO         v         COM
          Linux-host1 <-------------------> AIC-MCU <---------> Linux-host2
             driver                                              debug-mcu
```

基于 SDIO/USB 底层数据连接方式，封装数据消息包， MCU 与 Host 双方通过解析消息id，调用相应的指令函数，完成相应的通信功能

消息定义

1. 定义消息 id

1）MCU端在 wifi/hostif/fhostif_cmd.h 中定义

```
enum CUSTOM_MSG_TAG
{
    /// Memory read request
    CUSTOM_MSG_CONNECT_REQ = LMAC_FIRST_MSG(TASK_CUSTOM),
    CUSTOM_MSG_CONNECT_CFM,
    CUSTOM_MSG_CONNECT_IND,
    CUSTOM_MSG_DISCONNECT_REQ,
    CUSTOM_MSG_DISCONNECT_CFM,
    CUSTOM_MSG_DISCONNECT_IND,
    CUSTOM_MSG_ENTER_SLEEP_REQ,
    CUSTOM_MSG_ENTER_SLEEP_CFM,
    CUSTOM_MSG_EXIT_SLEEP_REQ,
    CUSTOM_MSG_EXIT_SLEEP_CFM,
    CUSTOM_MSG_SET_MAC_ADDR_REQ,
    CUSTOM_MSG_GET_MAC_ADDR_REQ,
    CUSTOM_MSG_GET_MAC_ADDR_CFM,
    CUSTOM_MSG_GET_WLAN_STATUS_REQ,
    CUSTOM_MSG_GET_WLAN_STATUS_CFM,
    CUSTOM_MSG_START_AP_REQ,
    CUSTOM_MSG_START_AP_CFM,
    CUSTOM_MSG_STOP_AP_REQ,
    CUSTOM_MSG_STOP_AP_CFM,
    CUSTOM_MSG_SCAN_WIFI_REQ,
    CUSTOM_MSG_SCAN_WIFI_CFM,
    CUSTOM_MSG_HOST_OTA_REQ,
    CUSTOM_MSG_HOST_OTA_CFM,

    CUSTOM_MSG_MAX = LMAC_FIRST_MSG(TASK_CUSTOM) + 50,
};
```

2）Host端在 wifi/LinuxDriver/aic8800_netdrv/rwnx_main.h 中定义

```c
enum CUSTOM_CMD_TAG
{
    CUST_CMD_CONNECT_REQ = TASK_CUSTOM,
    CUST_CMD_CONNECT_CFM,
    CUST_CMD_CONNECT_IND,
    CUST_CMD_DISCONNECT_REQ,
    CUST_CMD_DISCONNECT_CFM,
    CUST_CMD_DISCONNECT_IND,
    CUST_CMD_ENTER_SLEEP_REQ,
    CUST_CMD_ENTER_SLEEP_CFM,
    CUST_CMD_EXIT_SLEEP_REQ,
    CUST_CMD_EXIT_SLEEP_CFM,
    CUST_CMD_SET_MAC_ADDR_REQ,
    CUST_CMD_GET_MAC_ADDR_REQ,
    CUST_CMD_GET_MAC_ADDR_CFM,
    CUST_CMD_GET_WLAN_STATUS_REQ,
    CUST_CMD_GET_WLAN_STATUS_CFM,
    CUST_CMD_START_AP_REQ,
    CUST_CMD_START_AP_CFM,
    CUST_CMD_STOP_AP_REQ,
    CUST_CMD_STOP_AP_CFM,
    CUST_CMD_SCAN_WIFI_REQ,
    CUST_CMD_SCAN_WIFI_CFM,
    CUST_CMD_HOST_OTA_REQ,
    CUST_CMD_HOST_OTA_CFM,

    CUST_CMD_MAX
};
```

注意

a. CUSTOM_MSG_TAG 与 **CUSTOM_CMD_TAG** 中的枚举成员必须一一对应

b. 增加或删除指令时，必须在 **CUSTOM_MSG_TAG**、**CUSTOM_CMD_TAG** 中同时修改

c. 消息id后缀说明

```
REQ: request  -->  表示主控向MCU发起指令请求
CFM: confirm  -->  表示MCU收到指令，向主控确认开始执行指令
IND: indicate -->  表示MCU执行完指令，向主控反馈指令执行结果
```

2. 定义消息体

消息体用于 **MCU** 与 **Host** 之间传输通信数据

1）MCU端在 wifi/hostif/fhostif_cmd.h 中定义

```c
struct custom_msg_connect_req
{
    uint8_t ssid[32];
    uint8_t pw[64];
};

struct custom_msg_connect_ind
{
    uint8_t  ussid[32];
    uint32_t ip;
```

```
    uint32_t gw;
};

struct custom_msg_mac_addr_cfm
{
    uint8_t chip_mac_addr[6];
};
```

_req: Host 发送到 MCU 的消息体

_ind: MCU 发送到 Host 的消息体

_cfm: MCU 发送到 Host 的消息体

2）Host端在 wifi/LinuxDriver/aic8800_netdrv/rwnx_main.h 中定义

```
// cmd and data from netdrv to mcu
struct custom_msg_hdr
{
    u32_l rsv0;
    u16_l cmd_id;
    u16_l rsv[2];
    u16_l len;
};

struct custom_msg_connect_req
{
    struct custom_msg_hdr hdr;
    u8_l ssid[32];
    u8_l pw[64];
};

struct custom_msg_connect_ind
{
    uint8_t ussid[32];
    u32  ip;
    u32  gw;
};

struct custom_msg_mac_addr_cfm
{
    uint8_t mac_addr[6];
};
```

_req: Host 发送到 MCU 的消息体

_ind: MCU 发送到 Host 的消息体

_cfm: MCU 发送到 Host 的消息体

注意

a. 在 Host 中定义 **_req** 消息体必须先包含 **custom_msg_hdr** 结构体

b. 在 **Host** 与 **MCU** 中定义的结构体成员必须一一对应

消息绑定

1. Host 获取应用指令并发送消息

如 rwnx_main.c 中 handle_custom_msg 函数：

```
int handle_custom_msg(struct net_device *net, char *command, u32 cmd_len)
{
    /*
     * 省略：解析主控应用层指令
     */
    // 1.定义指令请求结构体
    aicwf_custom_msg_app_cmd cust_app_cmd;
    switch(ret) {
        case APP_CMD_CONNECT:
            // 2.设置消息id
            cust_app_cmd.connect_req.hdr.cmd_id = CUST_CMD_CONNECT_REQ;
            // 3.设置消息体参数
            // TODO: set req
            // 4.调用rwnx_tx_msg发送消息
            rwnx_tx_msg((u8 *)&cust_app_cmd.connect_req,
sizeof(cust_app_cmd.connect_req));
            break;
        case APP_CMD_DISCONNECT:
            ...
            break;
        case APP_CMD_ENTER_SLEEP:
            ...
            break;
        case APP_CMD_EXIT_SLEEP:
            ...
            break;
        case APP_CMD_GET_MAC_ADDR:
            ...
            break;
        case APP_CMD_GET_WLAN_STATUS:
            ...
            break;
        case APP_CMD_START_AP:
            ...
            break;
        case APP_CMD_STOP_AP:
            ...
            break;
        case APP_CMD_SCAN_WIFI:
            ...
            break;
        case APP_CMD_HOST_OTA:
            ...
            break;
    }
}
```

2. MCU 绑定消息 handler

如 fhostif_cmd.c 中，在 **const struct ke_msg_handler custom_msg_state[]** 中绑定消息id **CUSTOM_MSG_CONNECT_REQ** 对应的指令函数 **custom_msg_connect_handler**，

```
///  custom msg handler
const struct ke_msg_handler custom_msg_state[] = {
    {CUSTOM_MSG_CONNECT_REQ,
                (ke_msg_func_t)custom_msg_connect_handler},
    {CUSTOM_MSG_DISCONNECT_REQ,
                (ke_msg_func_t)custom_msg_disconnect_handler},
    {CUSTOM_MSG_ENTER_SLEEP_REQ,
                (ke_msg_func_t)custom_msg_enter_sleep_handler},
    {CUSTOM_MSG_EXIT_SLEEP_REQ,
                (ke_msg_func_t)custom_msg_exit_sleep_handler},
    {CUSTOM_MSG_SET_MAC_ADDR_REQ,
                (ke_msg_func_t)custom_msg_set_mac_addr_handler},
    {CUSTOM_MSG_GET_MAC_ADDR_REQ,
                (ke_msg_func_t)custom_msg_get_mac_addr_handler},
    {CUSTOM_MSG_GET_WLAN_STATUS_REQ,
                (ke_msg_func_t)custom_msg_get_wlan_status_handler},
    {CUSTOM_MSG_START_AP_REQ,
                (ke_msg_func_t)custom_msg_start_ap_handler},
    {CUSTOM_MSG_STOP_AP_REQ,
                (ke_msg_func_t)custom_msg_stop_ap_handler},
    {CUSTOM_MSG_SCAN_WIFI_REQ,
                (ke_msg_func_t)custom_msg_scan_wifi_handler},
    {CUSTOM_MSG_HOST_OTA_REQ,
                (ke_msg_func_t)custom_msg_host_ota_handler},
};
```

3. MCU 定义消息 handler

```
static int custom_msg_connect_handler(uint16_t const host_type,
                          void *param,
                          ke_task_id_t const dest_id,
                          ke_task_id_t const src_id)
{
    /*
     * 省略
     */

    dbg("custom_msg_connect_handler\r\n");
    struct custom_msg_common *cfm = (struct custom_msg_common *)e2a_msg_alloc(
        host_type, CUSTOM_MSG_CONNECT_CFM, 0, 0, sizeof(struct
custom_msg_common));
    if (cfm == NULL) {
        dbg(D_ERR "msg alloc err\r\n");
        return -1;
    }
    e2a_msg_send(host_type, cfm);

    if (WLAN_CONNECTED == wlan_get_connect_status()) {
        dbg("AP has connected, Disconnect first.\n");
        wlan_disconnect_sta(0);
    }
    struct custom_msg_connect_req *req = (struct custom_msg_connect_req *)param;
```

```
    if (wlan_start_sta(req->ssid, req->pw, 30000)) {
        dbg("connect failed\n");
    }

    /*
     * 省略
     */
    return KE_MSG_CONSUMED;
}
```

4. MCU 发送消息

```
static int custom_msg_get_mac_addr_handler(uint16_t const host_type,
                         void *param,
                         ke_task_id_t const dest_id,
                         ke_task_id_t const src_id)
{
    // 依据Host端的消息体结构custom_msg_mac_addr_cfm，构造消息体
    // 其中e2a_msg_alloc第二个参数必须设置为Host端接收的对应消息体id，如
CUSTOM_MSG_GET_MAC_ADDR_CFM
    struct custom_msg_mac_addr_cfm *mac_addr = (struct custom_msg_mac_addr_cfm
*)e2a_msg_alloc(
        host_type, CUSTOM_MSG_GET_MAC_ADDR_CFM, 0, 0, sizeof(struct
custom_msg_mac_addr_cfm));
    if (mac_addr == NULL) {
        dbg(D_ERR "msg alloc err\r\n");
        return -1;
    }

    /*
     * TODO: set mac_addr
     */

    // Post the confirm message to the host
    // 发送消息
    e2a_msg_send(host_type, mac_addr);

    return KE_MSG_CONSUMED;
}
```

5. Host 接收消息

Host端接收消息处理，在 **rwnx_rx.c** 的 **rwnx_rx_handle_msg** 中，

```
aicwf_custom_msg_vnet g_custom_msg_vnet;
void rwnx_rx_handle_msg( void *dev, struct ipc_e2a_msg *msg)
{
    //printk("rwnx_rx_handle_msg 0x%X \r\n", msg->id);
    switch(msg->id) {
        case CUST_CMD_CONNECT_CFM:
            ...
            break;
        case CUST_CMD_CONNECT_IND:
            ...
            // 依据指定id的消息结构体获取custom_msg消息
            memcpy(&g_custom_msg_vnet.connect_ind, msg->param, sizeof(struct
custom_msg_connect_ind));
```

```
            break;
        case CUST_CMD_DISCONNECT_CFM:
            ...
            break;
        case CUST_CMD_DISCONNECT_IND:
            ...
            break;
        case CUST_CMD_ENTER_SLEEP_CFM:
            ...
            break;
        case CUST_CMD_EXIT_SLEEP_CFM:
            ...
            break;
        case CUST_CMD_GET_MAC_ADDR_CFM:
            ...
            break;
        case CUST_CMD_GET_WLAN_STATUS_CFM:
            ...
            break;
        case CUST_CMD_START_AP_CFM:
            ...
            break;
        case CUST_CMD_STOP_AP_CFM:
            ...
            break;
        case CUST_CMD_SCAN_WIFI_CFM:
            ...
            break;
        case CUST_CMD_HOST_OTA_CFM:
            ...
            break;
    }
}
```

1）通过 **msg->id** 识别具体接收到的消息体的具体id，如 **CUST_CMD_CONNECT_IND**

2）通过消息体id对应的结构体类型，如 **custom_msg_connect_ind**，将 **msg->param** 数据复制到本地参数中

驱动端通信概要

custom_msg 实现的是 Linux-driver 与 AIC-MCU 之间的通信，而用户需通过 Linux-user 与 Linux-driver 交互，方可间接与 AIC-MCU 交互

a. 应用层下发指令

1. 在 wifi/LinuxDriver/app/custom_msg 目录下，custom_msg.c 通过 ioctl(sock) 实现了应用层与驱动层的交互
2. 编译执行 custom_msg.c

```
# 编译完成应用程序之后，输入./custom_msg，即可看到指令说明
"usage: custom_msg vnet0 [mode] <arg1> <arg2> <arg3>"
"---------------------"
">>>Interact with MCU:"
```

```
"custom_msg vnet0 1 ssid password          - connect ap"
"custom_msg vnet0 2                        - disconnect ap"
"custom_msg vnet0 3                        - enter sleep"
"custom_msg vnet0 4                        - exit sleep"
"custom_msg vnet0 5                        - get mcu mac"
"custom_msg vnet0 6                        - get wlan status"
"custom_msg vnet0 7 ssid password band     - start ap"
"                    -- band = <2.4G/5G>"
"custom_msg vnet0 8                        - stop ap"
"custom_msg vnet0 9                        - scan wifi"
"custom_msg vnet0 10 /your-path/update.bin - host ctrl OTA"
"-------------------------"
">>>Interact with kernel:"
"sudo custom_msg vnet0 ndev mac_address    - set vnet_dev mac"
```

b. 驱动传递指令

1. 驱动通过 ioctl 机制获取指令，并向 AIC-MCU 发起请求
2. Linux-driver 获取应用层指令流程如下

```
rwnx_do_ioctl --> mcu_cust_msg --> handle_custom_msg
```

3. 在 wifi/LinuxDriver/aic8800_netdrv 目录下，rwnx_main.h 定义了指令请求结构体，将传递请求过程中的消息体、变量都封装到 aicwf_custom_msg_app_cmd **cust_app_cmd** 变量中，设置完成相关消息体之后，通过 rwnx_tx_msg 调用 SDIO/USB 接口传输给 AIC-MCU

```
// custom_msg app cmd request information
typedef struct _aicwf_custom_msg_app_cmd {
    struct custom_msg_connect_req connect_req;
    struct custom_msg_hdr common_req;
    struct custom_msg_start_ap_req ap_req;
    char   file_path[APP_CMD_BUF_MAX];
} aicwf_custom_msg_app_cmd;
```

4. Linux-driver 获取应用层指令流程如下

```
rwnx_do_ioctl --> mcu_cust_msg --> handle_custom_msg
```

c. 驱动接收消息

1. Linux-driver 从 SDIO/USB 接口上，可以获取到来自 AIC-MCU 的 data据包，以及 msg数据包
2. msg数据包即是 AIC-MCU 反馈的消息，最终通过 rwnx_rx_handle_msg 接口进行解析
3. 在 wifi/LinuxDriver/aic8800_netdrv 目录下，rwnx_main.h 定义了接收消息结构体，将来自 AIC-MCU 消息都复制到 aicwf_custom_msg_vnet **g_custom_msg_vnet** 变量中

```
// custom_msg vnet status and information
typedef struct _aicwf_custom_msg_vnet {
    uint8_t wlan_status;
    int8_t ota_status;
    int8_t ap_status;
    bool comp_sign_get_mac_ready;
    bool comp_sign_get_wlan_ready;
    struct completion platform_get_mac_done;
    struct completion platform_get_wlan_done;
```

```c
    struct custom_msg_mac_addr_cfm macaddr_cfm;
    struct custom_msg_connect_ind connect_ind;
    struct custom_msg_wlan_status_cfm get_wlan_cfm;
    struct custom_msg_scan_wifi_result_cfm *scan_wifi_cfm_ptr;
    struct custom_msg_scan_wifi_result_cfm scan_wifi_cfm[32];
} aicwf_custom_msg_vnet;
```
