USR-G402tf Android 驱动安装说明

文件版本：V1.0.2

目录

1. Android 下的驱动安装

1.1. 添加驱动

设备实际会枚举出 3 个串口，其中为

a) AT Port (/dev/ttyUSB2)
b) Modem Port (/dev/ttyUSB1)
c) Daig Port (/dev/ttyUSB0)

注意：以上对应关系并不一定固定，不同型号的数据卡会有区别，需根据实际情况判断。

1、确保内核编译选项 CONFIG_USB_SERIAL 打开
2、在某个串口驱动中加入对应的 PID 和 VID。如在/kernel/drivers/usb/serial/option.c 的设备列表中加入 {USB_DEVICE(0x19d2, 0x0536)}

1.2. 修改 rild

获取 root 权限
rild 运行过程中，需要对设备枚举出的串口进行操作，因而需要较高的权限。而 ril 在初始话完成后会丢弃掉 root 权限，故将此处代码去掉即可。

文件路径：/hardware/ril/rild/rild.c

```
void switchUser() {
prctl(PR_SET_KEEPCAPS, 1, 0, 0, 0);
    //setuid(AID_RADIO); //  注释掉此行，保持 root 权限

    struct __user_cap_header_struct header;
struct __user_cap_data_struct cap;
… …
}
```

1.3. 修改 reference-ril 库

reference-ril 库的代码主要在/hardware/ril/reference-ril/reference-ril.c 中，以下所作修改均针对此文件。

1.4. 修改设备 modem 口路径

修改宏“PPP_TTY_PATH”为对应的路径，如“/dev/ttyUSB1”

```
//#define PPP_TTY_PATH "/dev/omap_csmi_tty1"
#define PPP_TTY_PATH "/dev/ttyUSB1"
```

1.5. 修正主动上报状态的问题

Android 获取网络状态主要通过上层的“主动查询”和底层的“主动上报”两种方法，具体做法可为：
在响应上层的一些查询命令的时候，顺便告知上层底层某些状态有变化，这样来触发上层进行进一步的查询，以此获取一些不能主动上报的状态变更。
修改信号强度处理函数 requestSignalStrength()，添加如下代码：

```
RIL_onRequestComplete(t, RIL_E_SUCCESS, response, sizeof(response));
RIL_onUnsolicitedResponse (
        RIL_UNSOL_RESPONSE_NETWORK_STATE_CHANGED,
        NULL, 0);  // Add
at_response_free(p_response);
return;
```

1.6. 修正网络类型

由于 AT 命令“AT+CGREG？”的返回结果不标准，上层无法区分出正确的网络类型（UMTS、HSDPA、HSUPA、GPRS、EDGE 等）。因此需要做相应修改，具体为：
修改函数 requestRegistrationState()，添加如下代码：

```
…
asprintf(&responseStr[0], "%d", response[0]);
asprintf(&responseStr[1], "%x", response[1]);
asprintf(&responseStr[2], "%x", response[2]);
if (count > 3)
    asprintf(&responseStr[3], "%d", response[3]);

// 添加如下判断，将 response[3]替换为正确的值。
if ((request == RIL_REQUEST_GPRS_REGISTRATION_STATE) && (1 == commas))
{
        int i32err = 0;
        i32err = get_network_type(&response[3]);
        if((0 == i32err) && (response[3] > 0))
        {
            asprintf(&responseStr[3], "%d", response[3]);
```

```
                count = 4;
            }
        }
        RIL_onRequestComplete(t, RIL_E_SUCCESS, responseStr,
count*sizeof(char*));
        at_response_free(p_response);
        …
```

其中 get_network_type()函数的实现为：

```
        typedef struct
        {
            int     nType;
            char    *strDesc;
        } TPsratTableItem;

        /*          对          应          的          表          在
/frameworks/base/telephony/java/android/telephony/ServiceState.java 中
        const TPsratTableItem aPsratTable[] =
        {
            {3,     "UMTS"},
            {9,     "HSDPA"},
            {10,    "HSUPA"},
            {1,     "GPRS"},
            {2,     "EDGE"},
            {0,     "NONE"},
        };

        static int get_network_type(int *pi32type)
        {
            int i32err = 0;
            ATResponse *p_response = NULL;
            int i32redo_num = 0;
            char *line = NULL;
            char *presult = NULL;
            size_t i =0;
            int i32ret = -1;

            if(NULL == pi32type)
            {
                LOGD("LD %s %d.", __FUNCTION__, __LINE__);
                return -1;
            }

            *pi32type = 0;
```

```
    redo:
        if(i32redo_num > 20)
        {
            goto error;
        }

        i32err    =    at_send_command_singleline("AT+PSRAT",    "+PSRAT",
&p_response);
        if (i32err != 0) goto error;

        if(p_response->success != 1)
        {
            LOGD("LD %s %d.", __FUNCTION__, __LINE__);
            sleep(1);
            ++i32redo_num;
            goto redo;
        }
        if(NULL == p_response->p_intermediates)
        {
            goto error;
        }

        line = p_response->p_intermediates->line;

        i32err = at_tok_start (&line);

        if (i32err < 0) {
            LOGD("LD %s %d.", __FUNCTION__, __LINE__);
            sleep(1);
            ++i32redo_num;
            goto redo;
        }

        i32err = at_tok_nextstr(&line, &presult);
        if (i32err < 0) {
            LOGD("LD %s %d.", __FUNCTION__, __LINE__);
            sleep(1);
            ++i32redo_num;
            goto redo;
        }
        LOGD("LD %s %d.%s.", __FUNCTION__, __LINE__, presult);
        for(i = 0; i < sizeof (aPsratTable) / sizeof(TPsratTableItem); i++)
        {
            if (strncmp (presult, aPsratTable[i].strDesc,
```

```
                strlen(aPsratTable[i].strDesc)) == 0)
            {
                *pi32type = aPsratTable[i].nType;
                i32ret = 0;
                break;
            }
        }
        LOGD("LD %s %d.%d.", __FUNCTION__, __LINE__, *pi32type);
        at_response_free(p_response);
        return i32ret;
        error:
            LOGD("LD %s %d.", __FUNCTION__, __LINE__);
            *pi32type = -1;
            at_response_free(p_response);
            return -1;

    }
```

1.7. 建立数据链路（拨号）

拨号的建立需要另外两个工具：pppd，chat，其中 chat 用于 AT 命令的发送。这两个程序都可从网上获取到。

拨号过程由函数

```
static void requestSetupDataCall(void *data, size_t datalen, RIL_Token t)
```

处理，因此需要修改此函数，具体为：

```
if (fd >= 0) { /* the device doesn't exist on the emulator */
        …
} else {
        asprintf(&cmd, "AT+CGDCONT=1,\"IP\",\"%s\",,0,0", apn);
    //FIXME check for error here
    err = at_send_command(cmd, NULL);
    free(cmd);

    // Set required QoS params to default
    //err = at_send_command("AT+CGQREQ=1", NULL);

    // Set minimum QoS params to default
    //err = at_send_command("AT+CGQMIN=1", NULL);

    // packet-domain event reporting
    //err = at_send_command("AT+CGEREP=1,0", NULL);
```

```
        // Hangup anything that's happening there now
        //err = at_send_command("AT+CGACT=1,0", NULL);

        // Start data on PDP context 1
        //err = at_send_command("ATD*99***1#", &p_response);


        /*
           err = at_send_command("ATDT*99#", &p_response);
        if (err < 0 || p_response->success == 0) {
              goto error;
        }
           */

           err = at_modem_setup_data_call();
           if(err != 0)
           {
                 goto error;
           }
    }
```

其中 int at_modem_setup_data_call()函数定义为：

```
    static int at_modem_setup_data_call()
    {
        kill_pppd();
        sleep(1);
        system("echo \"TIMEOUT 5 " AT OK AT OK ATE0V1 OK AT OK ATS0=0 OK AT OK ATE0V1 OK AT OK ATDT*99# CONNECT "\" > /chatscript;");
        system("pppd /dev/ttyUSB1 persist defaultroute usepeerdns connect \"chat -v -f /chatscript\"");
        return 0;
    }

    static int kill_pppd()
    {
        int i32ret = 0;
        int i32ppp_pid_num = -1;
        pid_t *pi32pid = NULL;
        int i = 0;

        i32ppp_pid_num = _pidof("pppd", &pi32pid);
        LOGI("%s,%d.pid num is %d.",__FUNCTION__, __LINE__, i32ppp_pid_num);
```

```
            while(i32ppp_pid_num > 0)
            {
                for(i = 0; i < i32ppp_pid_num; ++i)
                {
                    if (kill(pi32pid[i], SIGTERM) != 0)
                    {
                       LOGI("Could not kill pid '%d'", pi32pid[i]);
                   }
                }
                free(pi32pid);
                i32ppp_pid_num = 0;
                i32ppp_pid_num = _pidof("PPPD", &pi32pid);
                LOGI("%s,%d.pid  num  is  %d.",__FUNCTION__,  __LINE__,
i32ppp_pid_num);
                if(i32ppp_pid_num > 0)
                {
                    sleep(1);
                }
            }
            return 0;
        }

        static int f_read(const char *path, void *buffer, int max)
        {
           int f;
           int n;

           if ((f = open(path, O_RDONLY)) < 0) return -1;
           n = read(f, buffer, max);
           close(f);
           return n;
        }

        static int f_read_string(const char *path, char *buffer, int max)
        {
           if (max <= 0) return -1;
           int n = f_read(path, buffer, max - 1);
           buffer[(n > 0) ? n : 0] = 0;
           return n;
        }

        static char *psname(int pid, char *buffer, int maxlen)
        {
           char buf[512];
```

```
    char path[64];
    char *p;

    if (maxlen <= 0) return NULL;
    *buffer = 0;
    sprintf(path, "/proc/%d/stat", pid);
    if ((f_read_string(path, buf, sizeof(buf)) > 4) && ((p = strrchr(buf, ')'))
!= NULL))
    {
        *p = 0;
        if (((p = strchr(buf, '(')) != NULL) && (atoi(buf) == pid))
        {
            strncpy(buffer, p + 1, maxlen);
        }
    }
    return buffer;
}

static int _pidof(const char *name, pid_t** pids)
{
    const char *p;
    char *e;
    DIR *dir;
    struct dirent *de;
    pid_t i;
    int count;
    char buf[256];

    count = 0;
    *pids = NULL;

    if ((p = strchr(name, '/')) != NULL) name = p + 1;

    if ((dir = opendir("/proc")) != NULL)
    {
        while ((de = readdir(dir)) != NULL)
        {
            i = strtol(de->d_name, &e, 10);
            if (*e != 0) continue;
            if (strcmp(name, psname(i, buf, sizeof(buf))) == 0)
            {
                if ((*pids = realloc(*pids, sizeof(pid_t) * (count + 1))) ==
NULL)
                {
```

```
                        return -1;
                    }
                    (*pids)[count++] = i;
                }
            }
        }
        closedir(dir);
        return count;
    }
```

1.8. 断开数据连接

reference-ril 库可能没有添加对命令“RIL_REQUEST_DEACTIVATE_DATA_CALL”的处理，此处需要手动添加。

在 onRequest()函数中，添加如下的“case”判断：

```
    switch (request) {
        ....
    case RIL_REQUEST_DEACTIVATE_DATA_CALL:
            requestDeactiveDataCall(data, datalen, t);
            break;
        ....
    }
```

其中 requestDeactiveDataCall()的实现为：

```
    static void requestDeactiveDataCall(void *data, size_t datalen, RIL_Token t)
    {
        kill_pppd();
        RIL_onRequestComplete(t, RIL_E_SUCCESS, NULL, 0);
    }
```

1.9. Pin Code 处理

由于设备不能主动上报 SIM Card 的详细状态，因此进行 PIN 码相关操作后，需要主动查询 SIM 卡状态，并主动上报状态的变更，以触发上层的主动查询。

故添加如下代码：

```
    static void    requestEnterSimPin(void*    data, size_t    datalen, RIL_Token
t)
    {
        ...
        err = at_send_command_singleline(cmd, "+CPIN:", &p_response);
```

```
        free(cmd);
        i32SIMPinStatus = getSIMStatus();
        if(SIM_READY != i32SIMPinStatus){
    error:
            RIL_onRequestComplete(t,     RIL_E_PASSWORD_INCORRECT,
NULL, 0);
            RIL_onUnsolicitedResponse (
                RIL_UNSOL_RESPONSE_SIM_STATUS_CHANGED,
                NULL, 0);
        } else {
            RIL_onRequestComplete(t, RIL_E_SUCCESS, NULL, 0);
            RIL_onUnsolicitedResponse (
                RIL_UNSOL_RESPONSE_NETWORK_STATE_CHANGED,
                NULL, 0);
            setRadioState(RADIO_STATE_SIM_READY);
        }
        at_response_free(p_response);
    }
```

其中 getSIMStatus()需要做如下修改：

```
    static SIM_Status
    getSIMStatus()
    {
    …
    redo:
        if (err != 0) {
            ret = SIM_NOT_READY;
            goto done;
        }
    switch (at_get_cme_error(p_response)) {
            case CME_SUCCESS:
                break;

            case CME_SIM_NOT_INSERTED:
                ret = SIM_ABSENT;
                goto done;
            case CME_SIM_BUSY:
                sleep(1);
                goto redo;
            default:
                ret = SIM_NOT_READY;
                goto done;
    }
    …
```

```
    }

    其中“CME_SIM_BUSY”定义在/hardware/ril/reference-ril/atchannel.h 中：
    typedef enum {
        CME_ERROR_NON_CME = -1,
        CME_SUCCESS = 0,
        CME_SIM_NOT_INSERTED = 10,
        CME_SIM_BUSY = 14,
    } AT_CME_Error;
```

1.10. 在请求处理中添加一些保护性判断

修改函数 onRequest()，如下：

```
        if (sState == RADIO_STATE_OFF
            && !(request == RIL_REQUEST_RADIO_POWER
                || request == RIL_REQUEST_GET_SIM_STATUS)
        ) {
            RIL_onRequestComplete(t,      RIL_E_RADIO_NOT_AVAILABLE,
NULL, 0);
            return;
    }

        if(request == RIL_REQUEST_GET_CURRENT_CALLS)
        {
            RIL_onRequestComplete(t,  RIL_E_GENERIC_FAILURE,  NULL,
0);
            return ;
    }

    if ((sState != RADIO_STATE_SIM_READY)
            && (request == RIL_REQUEST_GET_CURRENT_CALLS
                || request == RIL_REQUEST_REGISTRATION_STATE
                ||                  request                  ==
RIL_REQUEST_GPRS_REGISTRATION_STATE
                ||                  request                  ==
RIL_REQUEST_QUERY_NETWORK_SELECTION_MODE
                || request == RIL_REQUEST_OPERATOR)
            )
        {
            RIL_onRequestComplete(t,  RIL_E_GENERIC_FAILURE,  NULL,
0);
            return;
```

```
}

switch (request) {
…
```

1.11. 修改初始化流程

由于部分 AT 命令不支持或不兼容，需要去掉初始化过程中的一些 AT 命令。
修改函数 initializeCallback()，注释如下 AT 命令：

```
//at_send_command("ATS0=0", NULL);

/*  Call Waiting notifications */
    //at_send_command("AT+CCWA=1", NULL);

    /*  Alternating voice/data off */
    //at_send_command("AT+CMOD=0", NULL);

    /*  Not muted */
    //at_send_command("AT+CMUT=0", NULL);

    /*  +CSSU unsolicited supp service notifications */
    //at_send_command("AT+CSSN=0,1", NULL);

    /*  no connected line identification */
//at_send_command("AT+COLP=0", NULL);

/*  USSD unsolicited */
//at_send_command("AT+CUSD=1", NULL);
```

1.12. 修改设备拔出后的回调函数

修改函数 onATReaderClosed()，进行数据卡拔出后的后续处理，添加如下操作：

```
static void onATReaderClosed()
{
    kill_pppd();
at_close();

…
}
```

1.13. 自动优先注册到 3G 网络

在初始化回调函数 initializeCallback()中做如下更改：

```
static void initializeCallback(void *param)
{
    …
at_send_command("AT+CGEREP=1,0", NULL);

    /*  SMS PDU mode */
    at_send_command("AT+CMGF=0", NULL);
setsearchmode(2);
…
}
```

其中，setsearchmode()实现为：

```
/*
1, UMTS ONLY
2, UMTS PREFERRED
3, GSM ONLY
4, GSM PREFERRED
*/
static int setsearchmode(int i32mode)
{
    ATResponse *p_response = NULL;
    int err;
    int ret;
    int i32cme_err = 0;
    char *cpinLine;
    char *cpinResult;
    int i32retry = 0;
#if 0
    if (sState == RADIO_STATE_OFF || sState ==
RADIO_STATE_UNAVAILABLE) {
        ret = -1;
        goto done;
    }
#endif
redo:

    if(++i32retry > 10)
    {
```

```
                ret = -1;
                goto done;
            }
            if(p_response != NULL)
            {
                at_response_free(p_response);
                p_response = NULL;
            }
            err = at_send_command_singleline("AT+MODODR=1", "+MODODR:",
&p_response);
            LOGD("LD %s %d.%d.", __FUNCTION__, __LINE__, err);
            if (err != 0) {
                sleep(1);
                LOGD("LD %s %d.", __FUNCTION__, __LINE__);
                goto done;
            }
            i32cme_err = at_get_cme_error(p_response);
             LOGD("LD %s %d.%d.", __FUNCTION__, __LINE__,i32cme_err);
            switch (at_get_cme_error(p_response)) {
                case CME_SUCCESS:
                    break;

                case CME_SIM_NOT_INSERTED:
                    ret = SIM_ABSENT;
                    goto done;
                case CME_SIM_BUSY:
                    sleep(1);
                    goto redo;
                default:
                    ret = SIM_NOT_READY;
                    goto done;
            }

            /* CPIN? has succeeded, now look at the result */

            cpinLine = p_response->p_intermediates->line;
            err = at_tok_start (&cpinLine);

            if (err < 0) {
                LOGD("LD %s %d.", __FUNCTION__, __LINE__);
                sleep(1);
                goto redo;
            }
```

```
        err = at_tok_nextstr(&cpinLine, &cpinResult);

        if (err < 0) {
            LOGD("LD %s %d.", __FUNCTION__, __LINE__);
            sleep(1);
            goto redo;
        }

        if (0 == strcmp (cpinResult, "OK")) {
            ret = 1;
            goto done;
        } else {
            LOGD("LD %s %d.", __FUNCTION__, __LINE__);
            sleep(1);
            goto redo;
        }

        at_response_free(p_response);
        p_response = NULL;
        cpinResult = NULL;

        ret = 1;

    done:
        at_response_free(p_response);
        return ret;
    }
```

1.14. 新编译驱动、rild、libreference-ril.so

修改 Android 启动脚本 init.rc

加载驱动

如果驱动被编译成模块，则需要在启动脚本中动态加载。可在“on boot”标签前添加如下行：

```
    … …
    insmod /driver-path/xxx.ko    //添加此行

    on boot: //在此行之前
```

1.15. 自动启动 rild 服务

把 rild 服务添加到 init.rc 中

```
service ril-daemon /system/bin/rild –l /system/lib/libreference-ril.so -- -d
/dev/ttyUSB2
    socket rild stream 660 root radio
    socket rild-debug stream 660 radio system
    user root
group radio cache inet misc audio sdcard_rw
```

其中：

d) “ril-deamon”为该服务的名称，固定不变
e) “/system/bin/rild”为 rild 的存放路径
f) “-l xxx.so”指示 reference-ril 动态库的路径
g) “--”表示其后的参数为动态库的参数
h) “-d /dev/ttyUSB2”表示 AT 口的设备路径

2. 联系方式

公 司：济南有人物联网技术有限公司

地 址：山东省济南市高新区新泺大街 1166 号奥盛大厦 1 号楼 11 层

网 址：http://www.usr.cn

客户支持中心：http://h.usr.cn

邮 箱：sales@usr.cn

企 业 QQ：8000 25565

电 话：4000-255-652 或者 0531-88826739

有人愿景：国内联网通讯第一品牌

公司文化：有人在认真做事!

产品理念：简单 可靠 价格合理

有人信条：天道酬勤 厚德载物 共同成长

3. 免责声明

本文档未授予任何知识产权的许可，并未以明示或暗示，或以禁止发言或其它方式授予任何知识产权许可。除在其产品的销售条款和条件声明的责任之外，我公司概不承担任何其它责任。并且，我公司对本产品的销售和/或使用不作任何明示或暗示的担保，包括对产品的特定用途适用性，适销性或对任何专利权，版权或其它知识产权的侵权责任等均不作担保。本公司可能随时对产品规格及产品描述做出修改，恕不另行通知。

4. 更新历史

2016-01-13 V1.0 初始版本建立