USR-G402tf Android Drive User Guide

Version: V1.0.2

CONTENTS

1. Android Environment Drive Installation

1.1. Adding Drive

The device will actually enumerate 3 interfaces, including

► AT Port (/dev/ttyUSB2)
► Modem Port (/dev/ttyUSB1)
► Daig Port (/dev/ttyUSB0)

Note: The above corresponding relation is not fixed, different model data card has difference, should be judged according to actual situation.

1. Ensure core complying option CONFIG_USB_SERIAL be open
2. Add corresponding PID and VID in a certain interface. Such as adding {USB_DEVICE(0x19d2, 0x0536)} in device list of /kernel/drivers/usb/serial/option.c

1.2. Revise rild

* Obtain root authority
* In the running process of rild, interfaces enumerated by the device should be operated, so fairly high authority is needed. While ril will abandon root authority after completing initialization, so the code can be removed.
* File route: /hardware/ril/rild/rild.c

```
void switchUser() {
prctl(PR_SET_KEEPCAPS, 1, 0, 0, 0);
    //setuid(AID_RADIO); // annotate this line, keep root authority
    struct __user_cap_header_struct header;
struct __user_cap_data_struct cap;
… …
}
```

1.3. Revise reference-ril base

Reference-ril base code is mainly in /hardware/ril/reference-ril/reference-ril.c, hereinafter revision is for this file.

1.4. Revise the route of device modem interface

Revise macro “PPP_TTY_PATH” to the corresponding route，such as “/dev/ttyUSB1”

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

1.5. Amend active reporting status problem

Android obtains network status mainly by upper “active query” and lower “active report” methods, the detailed action is:

When responding some upper querying command, inform there are some changes for upper and lower status by the way, in order to strike the upper to conduct further query, to obtain some status change which can not report actively.

Revise signal intensity disposal function requestSignalStrength(), adding the below codes:

```
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. Amend network type

As AT command “AT+CGREG ? ” returning result is not standard, the upper can not distinguish the correct network types（UMTS, HSDPA, HSUPA, GPRS, EDGE, etc.). So, corresponding revision should be conducted, the details are as below:

Revise function requestRegistrationState(), adding the below codes:

```
…
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]);

// adding the below judge，replace response[3] to correct value。
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);
    …
```

In which get_network_type() function's realization is:

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

    /* corresponding tables in
/frameworks/base/telephony/java/android/telephony/ServiceState.java
    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. Setup data link (Dial)

Dial-up needs the other two tools: pppd & chat, while chat is used for AT command's sending. Both of the two procedures can be obtained in internet.

Dialing process is disposed by following function,

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

So this function needs to be revised, the details are as below:

```
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;
               }
      }
```

While int at_modem_setup_data_call() function is defined as：

```
      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. Disconnect from the data

Reference-ril base may not add disposal for the command " RIL_REQUEST_DEACTIVATE_DATA_CALL" It needs to be added manually.

In function of onRequest(), add the following "case" judge:

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

While requestDeactiveDataCall() can be realized by:

```
    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 Disposal

As the device can not actively report SIM Card's detailed status, so after relating operations for PIN code, SIM card status needs to be queried actively and actively report status change, to strike the upper active query.

So the below cods should be added:

```
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);
}
```

While getSIMStatus() needs to be conducted with the below changes:

```
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;
        }
        …
        }

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

1.10. Add some protective judgement in demand processing.

Amend the function of onRequest() as below:

```
            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. Revise initialization process

As some of AT commands are non-supporting or non-compatible, so some AT commands in initialization process should be removed.

Revise function initializeCallback(), annotate the following AT commands:

```
    //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. Revise callback function after drawing device

Revise function onATReaderClosed(), conduct subsequent disposal after drawing data card, adding the below operations:

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

…
}
```

1.13. Automatic preferentially register to 3G network

Conduct the below changes in initialization callback function initializeCallback():

```
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);
…
}
```

while，setsearchmode() can be realized as：

```
/*
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. Newly complied drive, rild, libreference-ril.so

* Revise Android starting script init.rc
* Loading drive
* If the drive is complied to module, dynamical loading is needed in starting script. Adding the below lines before the tag of “on boot”:

```
… …
insmod /driver-path/xxx.ko    //adding this line

on boot: // before this line
```

1.15. Automatic start rild service

Add rild service to init.rc

```
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
```

In which:

a) “ril-deamon” is name of this service and it is fixed without changing
b) “/system/bin/rild” is the storing route of rild
c) “-l xxx.so” indicates reference-ril dynamic base route
d) “--” means subsequent parameter is dynamic base’s parameter
e) “-d /dev/ttyUSB2” means AT interface’s device route

2. Contact

Company: Jinan USR IOT Technology Limited
Address: Floor 11, Building1, No.1166 Xinluo Street, Gaoxin Distric, Jinan, Shandong, China
Tel: 86-531-55507297, 86-531-88826739
Web: http://www.usriot.com
Support : http://h.usriot.com
Email: sales@usr.cn, tec@usr.cn

3. Declaimer

This file never granted any permission of intellectual property right, and never expressed or hinted, or banned to post or other modes to grant any intellectual property right permission. Our company will not bear any other responsibility beyond the obligation of our products' selling provisions and conditional declarations. Meanwhile, our company will not make any expressed or hinted guarantee for this product's selling and/or using, including for this product's specific utilization applicability, marketability or for any patent right, version right or other intellectual property right's tort liability. The company may make changes for product specification and product description at any time without prior notice.

4. Updating History

V1.0 initial version was set up on Jan. 13, 2016.