Contents

SCOPE

This document is a brief description on:

1. How to port the RIL code to one’s Android system.
2. How to build the RIL code on one’s Android system.
3. How to use the RIL library on one’s Android system.
4. How to debug the RIL library on one’s Android system.
5. How to add a new request to RIL on one’s Android system.
6. Support Android ril 2.3,4.0，4.2，4.4.

1 Porting the RIL code

If the source code has been received from SIMCOM one can start to port the code to his own Android system.

Since the source code from SIMCOM is based on Android 2.3, so please make more patiently and carefully while porting the code to the system other than 2.3.

All the modification to the source code by SIMCOM is covered under SIMCOM_RADIO feature, so one only needs to port these codes.

Following files need to be ported:

```
include/telephony/ril.h
rild/radiooptions.c
libril/Android.mk
libril/ril.cpp
libril/Ril_commands.h
reference-ril/reference-ril.c
reference-ril/Android.mk
Reference-ril-simcom.c        //add by simcom
Reference-ril-simcom.h        //add by simcom
reference-ril/reference-simcom-cdma-sms.c       //add by simcom
reference-ril/reference-simcom-cdma-sms.h      //add by simcom
```

As the reference code may be different from SIMCOM, one may need to modify some other files

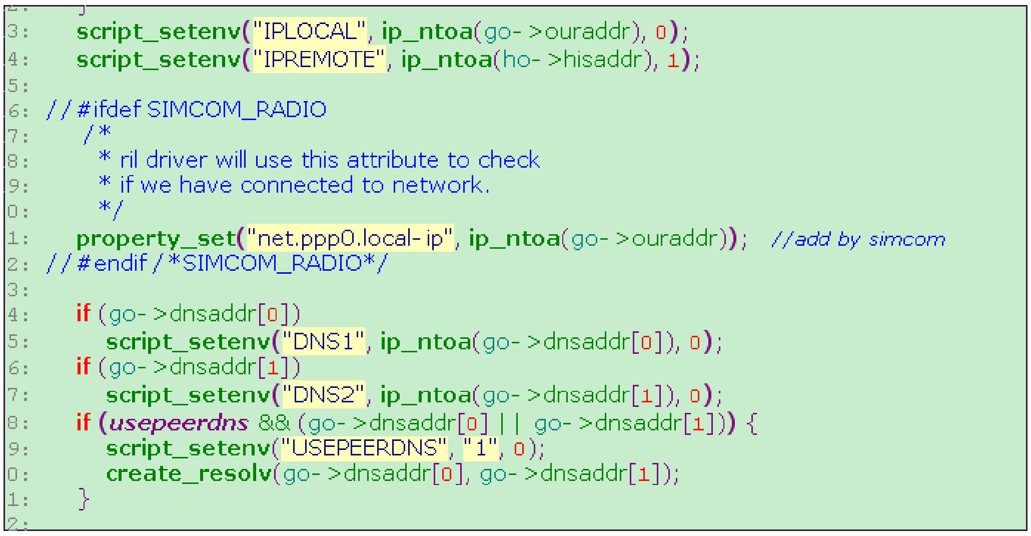
1. system/core/init/property_service.c:

```
/*
 * White list of UID that are allowed to start/stop services.
 * Currently there are no user apps that require.
 */
struct {
    const char *service;
    unsigned int uid;
    unsigned int gid;
} control_perms[] = {
    { "dumpstate",AID_SHELL, AID_LOG },
//#ifdef SIMCOM_RADIO
    { "pppd_gprs",AID_RADIO, AID_RADIO },  //add by simcom
//#endif /*SIMCOM_RADIO*/
     {NULL, 0, 0 }
};
```

If this file has been modified, one needs to rebuild and replace the "init" process.

2. external/ppp/pppd/ipcp.c:

```
static void ipcp_up(f)
{
 ……
```



```
    script_setenv("IPLOCAL", ip_ntoa(go->ouraddr), 0);
    script_setenv("IPREMOTE", ip_ntoa(ho->hisaddr), 1);

//#ifdef SIMCOM_RADIO
    /*
     * ril driver will use this attribute to check
     * if we have connected to network.
     */
    property_set("net.ppp0.local-ip", ip_ntoa(go->ouraddr));  //add by simcom
//#endif /*SIMCOM_RADIO*/
//
    if (go->dnsaddr[0])
        script_setenv("DNS1", ip_ntoa(go->dnsaddr[0]), 0);
    if (go->dnsaddr[1])
        script_setenv("DNS2", ip_ntoa(go->dnsaddr[1]), 0);
    if (usepeerdns && (go->dnsaddr[0] || go->dnsaddr[1])) {
        script_setenv("USEPEERDNS", "1", 0);
        create_resolv(go->dnsaddr[0], go->dnsaddr[1]);
    }
```

```
……
}
```

If this file has been modified, one needs to rebuild and replace the tool “pppd”.

2 Building the RIL code

After porting the source code, one needs to re-compile the RIL code in order to generate the new RIL library. There is no need to re-compile the whole Android system, RIL Library is enough.

Following steps depict how to compile the RIL code based on my environment:

1. cd ~/work/froyo_r2
2. . build/envsetup.sh
3. choosecombo 1 1 imx51_bbg 3
4. mmm ~/work/froyo_r2/hardware/ril/reference-ril // libreference-ril.so generated
5. mmm ~/work/froyo_r2/hardware/ril/libril // libril.so generated

If all right the library: libreference-ril.so, libril.so will be created.

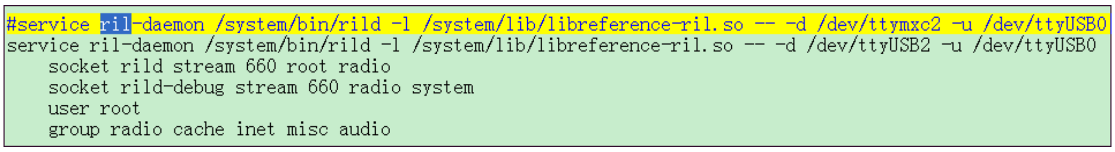
3 Using the RIL library

Now two Library files: libril.so and libreference-ril.so are existed, together with another three script files (init.rc, init.gprs-pppd, 3gdata_call.conf which can also be received from SIMCOM) one can use his device to dial, send/receive sms, browse internet and so on.

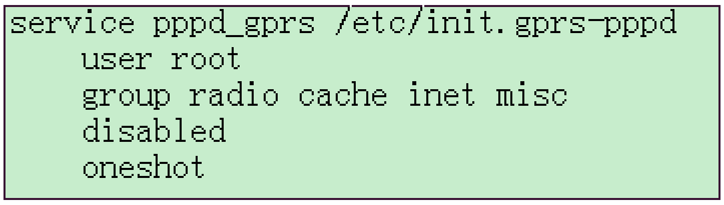
Following paragraph will use data call as an example on how to use RIL library:

3.1 Prepare

1. Refer to SIMCOM’s init.rc please modify the init.rc on one’s own device.



```
#service ril-daemon /system/bin/rild -l /system/lib/libreference-ril.so -- -d /dev/ttymxc2 -u /dev/ttyUSB0
service ril-daemon /system/bin/rild -l /system/lib/libreference-ril.so -- -d /dev/ttyUSB2 -u /dev/ttyUSB0
    socket rild stream 660 root radio
    socket rild-debug stream 660 radio system
    user root
    group radio cache inet misc audio
```



```
service pppd_gprs /etc/init.gprs-pppd
    user root
    group radio cache inet misc
    disabled
    oneshot
```

The parameter /dev/ttyUSB2 is not fixed, and it could be another value when you change to use a different device.

A similar parameter occurs in file “init.gprs-pppd”, it is the parameter of pppd. You should change it according to your device.

2. Put the file: init.gprs-pppd, 3gdata_call.conf to the directory: /etc/, don’t forget add execute permission to file init.gprs-pppd.
3. Make sure the tool “chat” is existed on the device since SIMCOM’s pppd will use this tool to dial-up.
4. Make other tools ready like pppd and dial tools; of course most of them are implemented by Android system.

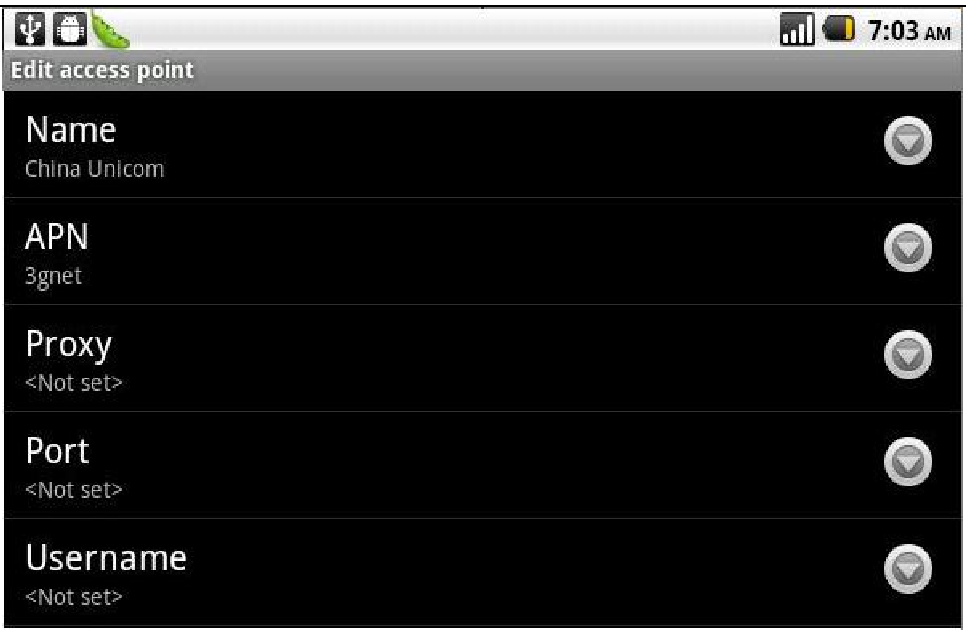
3.2 Use

Now one can setup a data call:

1. Create an APN

   Before dial-up one must create an APN first.

   Android Menu “Settings -> Wireless & networks -> Mobile networks -> Access Point Names -> New APN”


Edit access point

Name
China Unicom

APN
3gnet

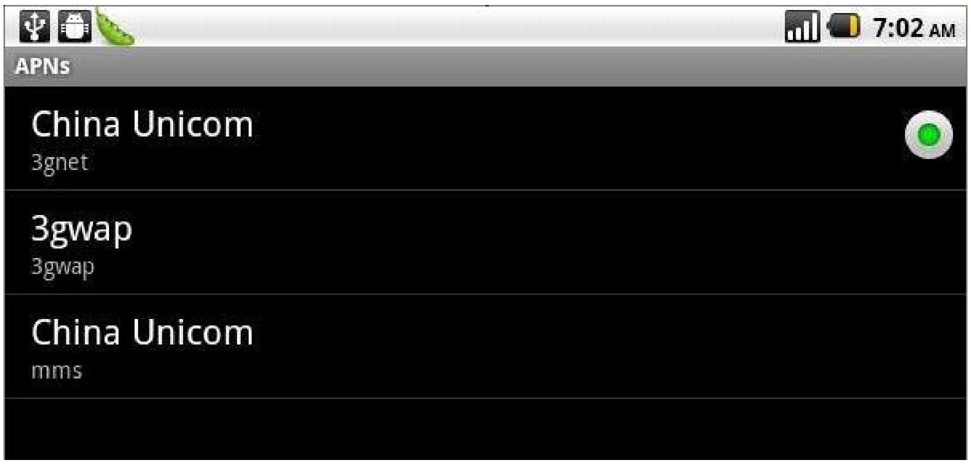
Proxy
<Not set>

Port
<Not set>

Username
<Not set>

The APN must be set rightly, or the device can’t connect to network successfully.

2. Select the right APN
After APN has been created, one needs to let such APN become effective:
Android Menu “Settings -> Wireless & networks -> Mobile networks -> Access Point Names”


APNs

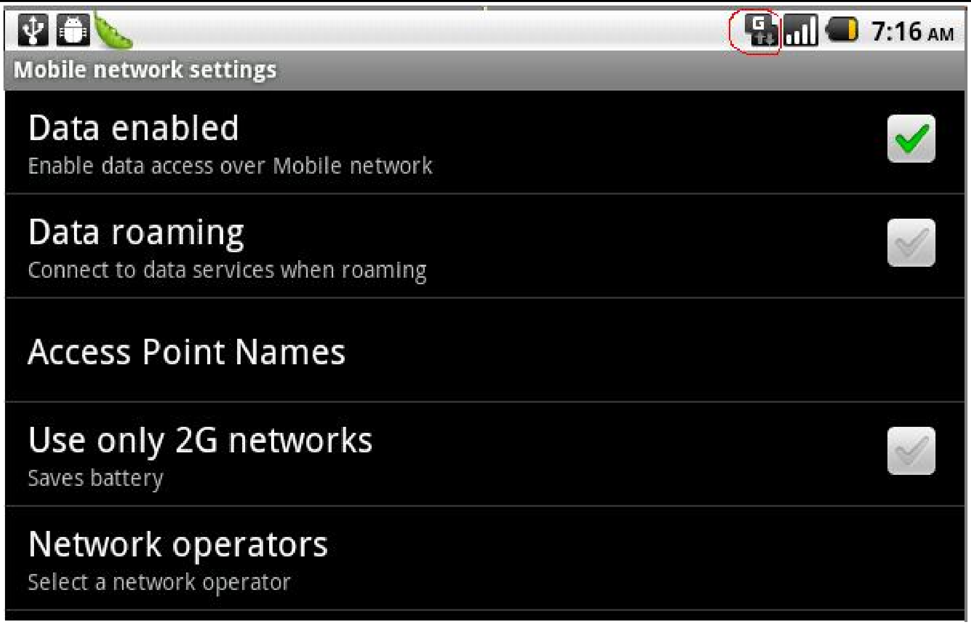
China Unicom
3gnet

3gwap
3gwap

China Unicom
mms

3. Connect to the network.
Now APN is OK, the device is ready to dial-up.

Android Menu “Settings -> Wireless & networks -> Mobile networks -> Data enabled”



7:16 AM

Mobile network settings

Data enabled
Enable data access over Mobile network ✓

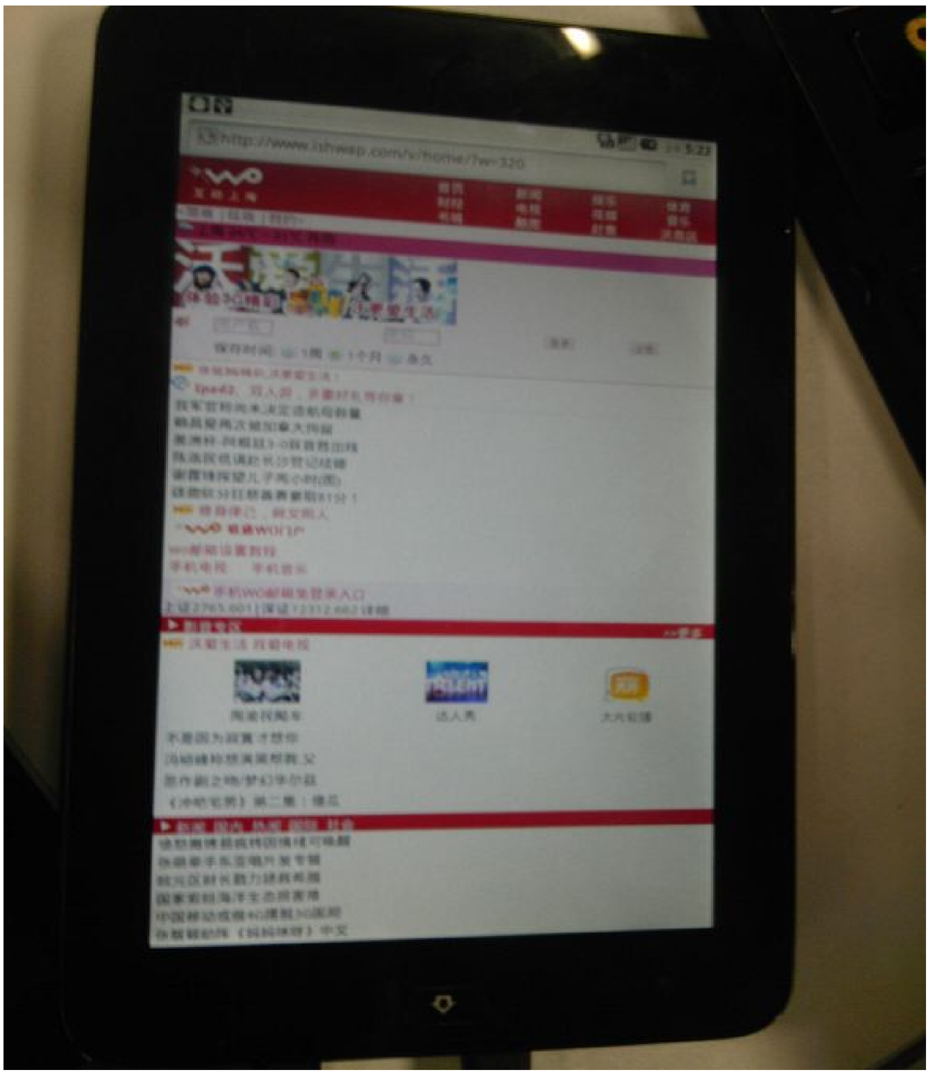
Data roaming
Connect to data services when roaming

Access Point Names

Use only 2G networks
Saves battery

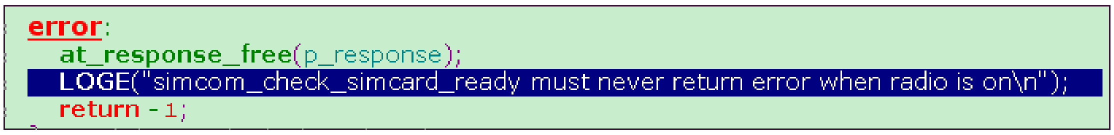
Network operators
Select a network operator

If all right now one can browse the website.

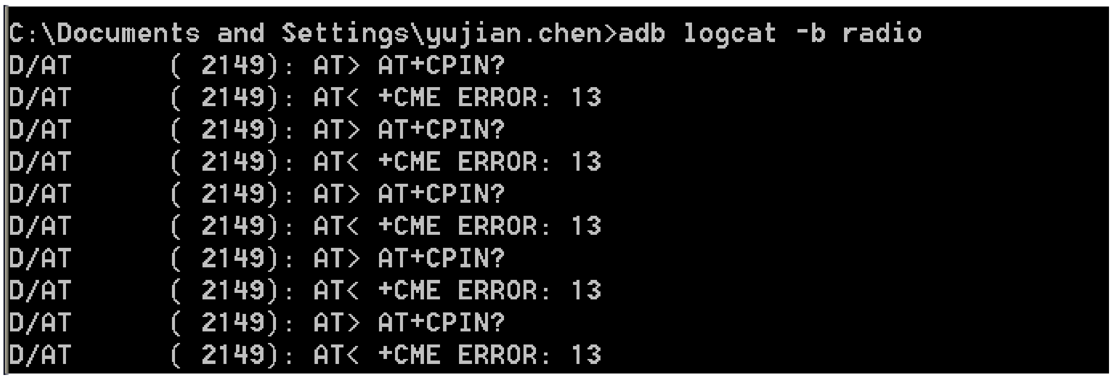


4 Debugging the RIL library

During the R&D process any problems may be encountered, one can use the macro "LOGD/LOGE/LOGW" in code to print log information, and use adb on the PC side to show these logs, also one can save these logs and send to SIMCOM, SIMCOM will help to analyze the bugs.



```
error:
   at_response_free(p_response);
   LOGE("simcom_check_simcard_ready must never return error when radio is on\n");
   return -1;
```



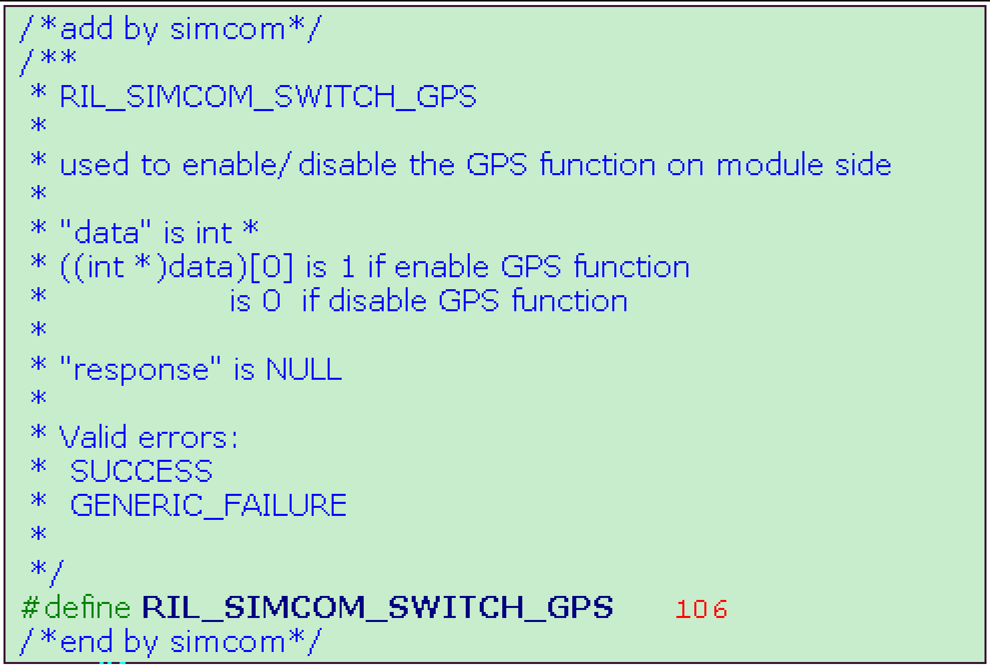
```
C:\Documents and Settings\yujian.chen>adb logcat -b radio
D/AT      ( 2149): AT> AT+CPIN?
D/AT      ( 2149): AT< +CME ERROR: 13
D/AT      ( 2149): AT> AT+CPIN?
D/AT      ( 2149): AT< +CME ERROR: 13
D/AT      ( 2149): AT> AT+CPIN?
D/AT      ( 2149): AT< +CME ERROR: 13
D/AT      ( 2149): AT> AT+CPIN?
D/AT      ( 2149): AT< +CME ERROR: 13
D/AT      ( 2149): AT> AT+CPIN?
D/AT      ( 2149): AT< +CME ERROR: 13
```

5 Adding a new request to RIL

Since SIMCOM has not implemented all of the functions in RIL, so one may need to add the new request to RIL by himself.

Following steps will use "Enable/disable GPS" as an example on how to add a new request to RIL code.

1. define the new request in ril.h
   hardware\ril\include\telephony\ril.h:

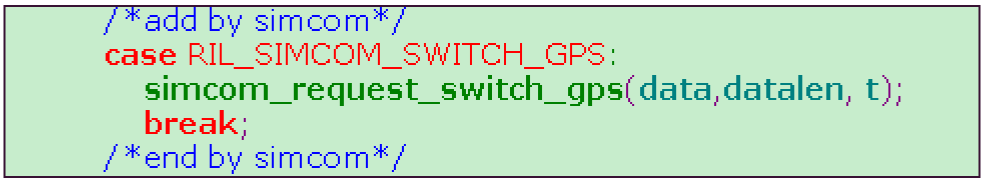


```
/*add by simcom*/
/**
 * RIL_SIMCOM_SWITCH_GPS
 *
 * used to enable/disable the GPS function on module side
 *
 * "data" is int *
 * ((int *)data)[0] is 1 if enable GPS function
 *                 is 0  if disable GPS function
 *
 * "response" is NULL
 *
 * Valid errors:
 *  SUCCESS
 *  GENERIC_FAILURE
 *
 */
#define RIL_SIMCOM_SWITCH_GPS     106
/*end by simcom*/
```

2. deal with the request

hardware\ril\reference-ril\reference-ril.c:

```
static void OnRequest(int request, void *data, size_t datalen, RIL_Token t)
{
    …
```



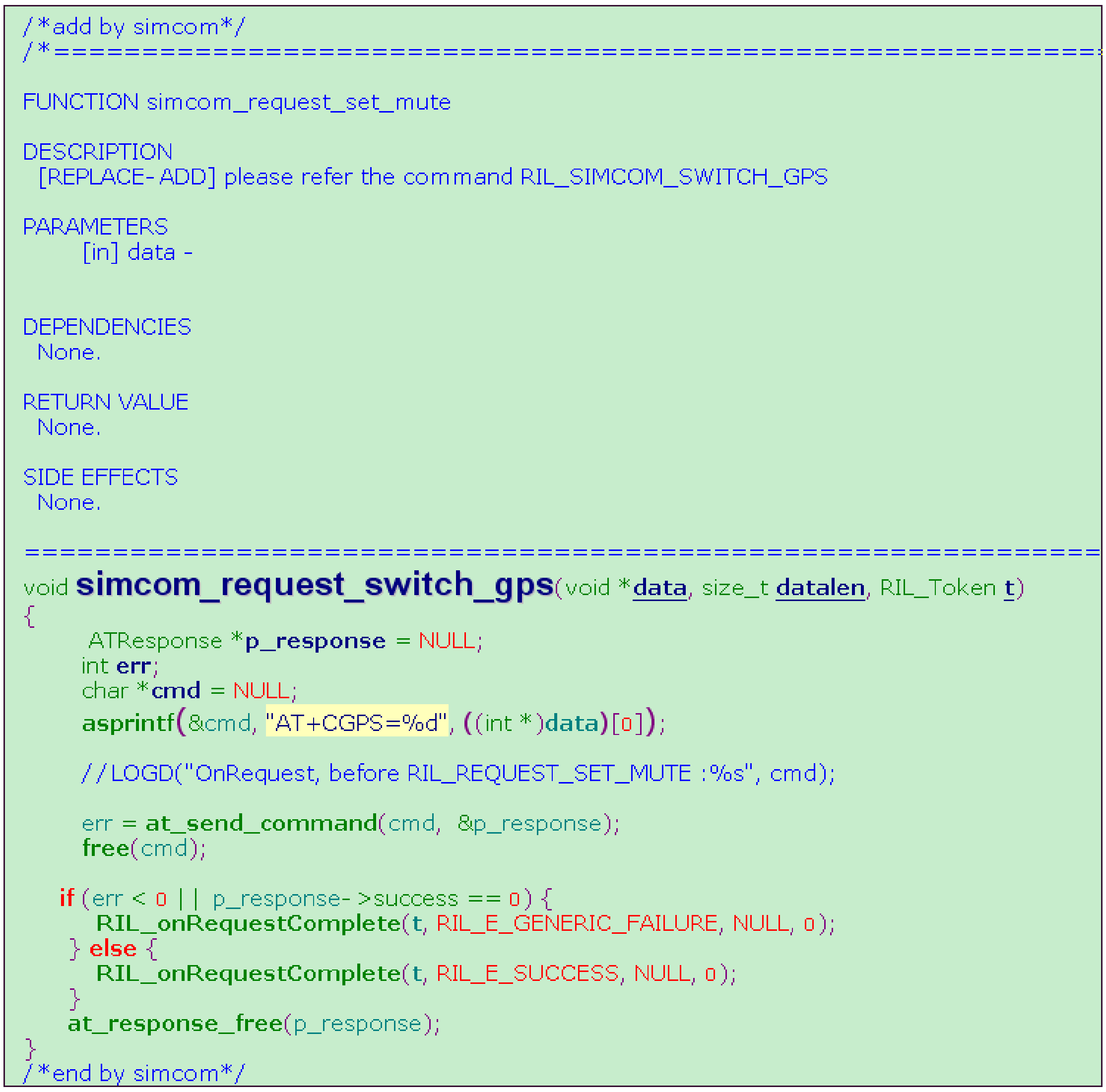
```
        /*add by simcom*/
        case RIL_SIMCOM_SWITCH_GPS:
            simcom_request_switch_gps(data,datalen, t);
            break;
        /*end by simcom*/
```

```
…
}
```

The GPS application will send a request to RIL in order to operate GPS; of course in this case it must send the request "RIL_SIMCOM_SWITCH_GPS" with the appropriate parameter to RIL. Or one can change this request to adapt the GPS application.

3. implement the request:

Hardware\ril\reference-ril\reference-ril-simcom.c:



```
/*add by simcom*/
/*=================================================================

FUNCTION simcom_request_set_mute

DESCRIPTION
  [REPLACE-ADD] please refer the command RIL_SIMCOM_SWITCH_GPS

PARAMETERS
        [in] data -


DEPENDENCIES
  None.

RETURN VALUE
  None.

SIDE EFFECTS
  None.

=================================================================
void simcom_request_switch_gps(void *data, size_t datalen, RIL_Token t)
{
        ATResponse *p_response = NULL;
        int err;
        char *cmd = NULL;
        asprintf(&cmd, "AT+CGPS=%d", ((int *)data)[0]);

        //LOGD("OnRequest, before RIL_REQUEST_SET_MUTE :%s", cmd);

        err = at_send_command(cmd,  &p_response);
        free(cmd);

     if (err < 0 || p_response->success == 0) {
         RIL_onRequestComplete(t, RIL_E_GENERIC_FAILURE, NULL, 0);
      } else {
         RIL_onRequestComplete(t, RIL_E_SUCCESS, NULL, 0);
      }
      at_response_free(p_response);
}
/*end by simcom*/
```

Now the GPS function on module side is enabled, and one can get GPS data (NMEA format) from ttyUSB1 (this is the NMEA port of SIMCOM module).

4. Get location information

Now the GPS data is outputted over the ttyUSB1 by SIMCOM module, and the GPS application needs to get the data and parse it in order to get the valid location information.