How to use Linux driver
General Notes
SIMCom offers this information as a service to its customers, to support application and engineering efforts that use the products designed by SIMCom. The information provided is based upon requirements specifically provided to SIMCom by the customers. SIMCom has not undertaken any independent search for additional relevant information, including any information that may be in the customer’s possession. Furthermore, system validation of this product designed by SIMCom within a larger electronic system remains the responsibility of the customer or the customer’s system integrator. All specifications supplied herein are subject to change.

Copyright
This document contains proprietary technical information which is the property of SIMCom Limited., copying of this document and giving it to others and the using or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights reserved in the event of grant of a patent or the registration of a utility model or design. All specification supplied herein are subject to change without notice at any time.

Copyright © Shanghai SIMCom Wireless Solutions Ltd. 2012
# Version History

<table>
<thead>
<tr>
<th>Version</th>
<th>Chapter</th>
<th>Author</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>V1.0</td>
<td>New Version</td>
<td>aaron</td>
<td>New version</td>
</tr>
<tr>
<td>V1.1</td>
<td>Chapter 2</td>
<td>aaron</td>
<td>We recommend to use the driver option instead of driver generic</td>
</tr>
<tr>
<td>V1.2</td>
<td>Chapter 1.1.3</td>
<td>aaron</td>
<td>Add flag for short packet transmission in some case</td>
</tr>
</tbody>
</table>
Contents

General Notes .................................................................................................................................................. - 2 -
Copyright ..................................................................................................................................................... - 2 -
Version History .............................................................................................................................................. - 3 -
  1  Driver issued by Linux kernel .................................................................................................................. - 5 -
  1.1  Modify the driver ..................................................................................................................................... - 5 -
  1.1.1  Support system suspend/resume ....................................................................................................... - 5 -
  1.1.2  Support low power mode .................................................................................................................. - 6 -
  1.1.3  Add short packet flag ......................................................................................................................... - 6 -
  1.2  Build the driver ....................................................................................................................................... - 7 -
  1.3  Use the driver ......................................................................................................................................... - 11 -
  1.3.1  Install the driver(driver as module only) ......................................................................................... - 11 -
  1.3.2  Use the driver ..................................................................................................................................... - 12 -
  1.3.3  Remove the driver .............................................................................................................................. - 14 -
SCAPE

This document is a brief description on:
1. How to build and use the driver on Linux issued by SIMCom in order to use SIMCom devices.
2. How to modify, build and use the driver on Linux issued by Linux kernel in order to use SIMCom devices.

1 Driver issued by Linux kernel

In fact the kernel with version of 2.6.20 and later has a common driver named usbserial which can also be used by SIMCom device.

Succeeding sections will use the kernel code of 2.6.35 as an example to depict how to modify, build and use kernel driver for SIMCom device in fail detail.

1.1 Modify the driver

One needs to add the vendor ID and product ID of SIMCom to kernel driver in order to support SIMCom device.

```
/* some devices interfaces need special handling due to a number of reasons */
enum option blacklist_reason {
    OPTION_BLACKLIST_NONE = 0,
    OPTION_BLACKLIST_SENDSETUP = 1,
    OPTION_BLACKLIST_RESERVED = 2
};
```

```
/*add by simcom*/
#define OLIVETT_VENDOR_ID 0x003C
#define OLIVETT_PRODUCT_ID 0x0000
/*end by simcom*/
```

```
static int option_blacklist_info[] = {
    .u8 = 1
};
```

```
static const u8 four_g_w14_no_sendsetup[] = {
    .info len = ARRAY_SIZE(four_g_w14_no_sendsetup),
    .ifaceinfo = four_g_w14_no_sendsetup,
    .reason = OPTION_BLACKLIST_SENDSETUP
};
```

```
static const struct usb device id option ids[] = {
    {USB DEVICE|SIMCOM WCMA_VENDOR_ID, SIMCOM WCMA PRODUCT ID }, /*add by simcom*/
    {USB DEVICE|SIMCOM_VENDOR_ID, SIMCOM_PRODUCT_ID },
    {USB DEVICE|SIMCOM_VENDOR_ID, SIMCOM_PRODUCT_RICOLA },
    {USB DEVICE|SIMCOM_VENDOR_ID, SIMCOM_PRODUCT_RICOLA },
};
```

1.1.1 Support system suspend/resume

Add .reset_resume call-back function if kernel support, for some USB HOST controller issue a bus reset to USB devices when system resume, USB port will be unloaded, and loaded later, the reset_resume call-back function will avoid the port unloading when system resume, for more detail please refer to kernel USB driver.
documents.

### 1.1.2 Support low power mode

For kernel 2.6.36, add the following highlight code to end of `option_probe` function:

```c
if (serial->dev->descriptor.idProduct == SIMCOM_SIM5300_PRODUCT ||
    serial->dev->descriptor.idProduct == SIMCOM_SIM5310_PRODUCT)
{
    
    #ifdef CONFIG_PM
    serial->interface->needs_remote_wakeup = 1;  /* autosuspend (1s8 delay) */
    device_init_makeup(serial->interface->dev);
    serial->dev->autosuspend_delay = (15 / 1000);  /* for kernel 2.6.36 */
    
    #endif

    if (data)
    {
        return -EINVAL;
    }

    data->sendグループ = option_sendグループ;
    spin_lock_init(&data->mutex_lock);
    data->private = {void *id->driver_info;
    return 0;
}
```

For kernel 2.6.38, add the following highlight code to end of `option_probe` function:

```c
if (serial->dev->descriptor.idProduct == SIMCOM_SIM5300_PRODUCT ||
    serial->dev->descriptor.idProduct == SIMCOM_SIM5310_PRODUCT)
{
    
    #ifdef CONFIG_PM
    serial->interface->needs_remote_wakeup = 1;  /* autosuspend (1s8 delay) */
    device_init_makeup(serial->interface->dev);
    serial->dev->autosuspend_delay = (15 / 1000);  /* for kernel 2.6.38 and above */
    
    #endif

    ...
```

### 1.1.3 Add short packet flag

Since the maximum packet size of BULK endpoint on SIMCOM module in High USB speed is 512 bytes, in Full USB speed is 64 bytes, in addition the USB protocol says:

An endpoint must always transmit data payloads with a data field less than or equal to the endpoint’s reported `wMaxPacketSize` value. When a bulk IRP involves more data than can fit in one maximum-sized data payload, all data payloads are required to be maximum size except for the last data payload, which will contain the remaining data. A bulk transfer is complete when the endpoint does one of the following:

- Has transferred exactly the amount of data expected
- Transfers a packet with a payload size less than `wMaxPacketSize` or transfers a zero-length packet

When a bulk transfer is complete, the Host Controller retries the current IRP and advances to the next IRP. If a data payload is received that is larger than expected, all pending bulk IRPs for that endpoint will be aborted/retired.

So one needs to send an zero-length packet additional if one wants to transmit the data stream with length exactly multiple of `wMaxPacketSize`.  

- 6 - 26.08.2008
Fortunately one needs not to send zero packet manually, one only needs to modify a little driver code:

drivers/usbserial/usb_wwan.c:

```c
/* Setup urbs */
static void __usb_wwan_setup_urbs(struct usb_serial *serial)
{
    int i, j;
    struct usb_serial_port *port;
    struct usb_wwan_port_private *portdata;

    dbg("wwan", __func__);

    for (i = 0; i < serial->num_ports; i++) {
        port = serial->port[i];
        portdata = usb_get_serial_port_data(port);
        /* Do indat endpoints first */
        for (j = 0; j < N_IN_URB; ++j) {
            portdata->in_urb[j] = usb_wwan_setup_urb(serial, port->
                                bulk_in_endpointAddress, USB_DIR_IN, 
                                port, portdata->
                                in_buffer[j], IN_BUFLEN,
                                usb_wwan_indat_callback);
        }

        /* outdat endpoints */
        for (j = 0; j < N_OUT_URB; ++j) {
            portdata->out_urb[j] = usb_wwan_setup_urb(serial, port->
                                bulk_out_endpointAddress, USB_DIR_OUT,
                                port, portdata->
                                out_buffer[j],
                                OUT_BUFLEN,
                                usb_wwan_outdat_callback);

            portdata->out_urb[j]->transfer_flags |= URB ZERO PACKET; //add by simcon
        }
    }
}
```  

**NOTE:** This modification is only for the driver option.ko

1.2 Build the driver

One needs to setup the kernel development environment first which include kernel source code and cross compiler environment.

Following is a step-by-step instruction on how to build the driver into kernel.

1) Use “sudo make menuconfig” to configure the kernel.

2) Enter into menu “Device Drivers”
3) Continue enter into menu “USB support”

4) Continue enter into menu “USB Serial Converter support”
5) Type “y” to select menu “USB driver for GSM and CDMA modems”, of course one can type “m” to compile the driver as a module.

6) Some other options need to be configured, so please enter into menu “Device Drivers -> Generic Driver Options”
7) Type “y” to select the following two options.

8) Exit and save the configuration.
After configuration, these items will be configured:

- CONFIG_USB = y
- CONFIG_USB_SERIAL=y
- CONFIG_USB_SERIAL_OPTION=y
- CONFIG_DEVTMPFS=y
- CONFIG_DEVTMPFS_MOUNT=y

2) Use “sudo make” to compile the kernel or use “sudo make modules” to compile the driver as a module.

1.3 Use the driver

As you move through this chapter new kernel firmware or new driver: option.ko(compiled as module) is ready.

1.3.1 Install the driver(driver as module only)

If one compiles the driver as a module one needs to install it first. one can use the following command to install the driver:

```
modprobe option.ko
```

This command will install all the needed drivers.
If all right the driver will be installed to the system, one can use the following command to query the result:

```
lsmod |grep option
```

```
root@freescale /lib/modules/2.6.35.3-571-gcca29a0/kernel/drivers/usb/serial$ lsmod |grep option
option     12548 0
usb_wwan    7381 1 option
usbserial   23430 2 option,usb_wwan
root@freescale /lib/modules/2.6.35.3-571-gcca29a0/kernel/drivers/usb/serial$
```

Note: this installation procedure is invalid when rebooting the system, so if one wants to install the driver automatically when starting the system, one should better put the installation instruction to the startup script.

### 1.3.2 Use the driver

After the driver installed one can use SIMCom device via the driver, now plug the SIMCom device to the host device via USB connector, and if the device is identified by the driver, 5 device files named ttyUSB0, ttyUSB1, ttyUSB2, ttyUSB3 and ttyUSB4 will be created in directory /dev.

The relationship between the device files and SIMCom composite device is like this:

<table>
<thead>
<tr>
<th>Device file</th>
<th>SIMCom composite device</th>
</tr>
</thead>
<tbody>
<tr>
<td>ttyUSB0</td>
<td>DIAG interface</td>
</tr>
<tr>
<td>ttyUSB1</td>
<td>NMEA interface</td>
</tr>
<tr>
<td>ttyUSB2</td>
<td>ATCOM interface</td>
</tr>
<tr>
<td>ttyUSB3</td>
<td>MODEM interface</td>
</tr>
<tr>
<td>ttyUSB4</td>
<td>Wireless Ethernet Adapter interface</td>
</tr>
</tbody>
</table>

SIMCom device is plugged in:

```
root@freescale /lib/modules/2.6.35.3-571-gcca29a0/kernel/drivers/usb/serial$ lsusb -v
USB Serial-COM

root@freescale /lib/modules/2.6.35.3-571-gcca29a0/kernel/drivers/usb/serial$ lsusb -v
USB Serial-COM
```

Device files are created:
NOTE:
1 In some composite devices of SIMCom not all of the interfaces are existed, so the relationship is dynamic.
2 Only the NMEA, ATCOM and MODEM interface can be worked correctly with this driver.

If one gets the device files ready one can use tools such as minicom, wvdial etc to use the device.
NMEA interface

1.3.3 Remove the driver

One can use the following command to uninstall the driver:

```
rmmod option
```

After removed one can use “lsmod |grep option” to check if the driver has been removed correctly.

*Note: when removing the driver one must disconnect the device and close all the tools using the device first.*