



SIM7500/SIM7600 _Linux_USB_User_Guide

LTE Module

Shanghai SIMCom Wireless Solutions Ltd.
Building A, SIM Technology Building, No.633, Jinzhong Road
Changning District 200335
Tel:86-21-31575100/31575200
support@simcom.com
www.simcom.com

Document Title:	SIM7500/SIM7600_Linux_USB_User_Guide
Version:	1.00.00
Date:	2020-05-11
Status:	Release

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. 2020

Version History

Version	Date	Chapter	What is new
V1.00.00	2020-01-02		New version

Contents

Version History	2
Contents	2
1 Introduction	3
1.1. Scope.....	3
1.2. Related Documents	3
2 Device Driver Configuration	3
2.1 USB Serial Driver	3
2.2. USB NDIS Driver	5
2.2.1 Add VID/PID for interface 5.	5
2.2.2 Add support for Raw IP mode.....	6
2.2.3 Kernel messages	10
2.3. USB RNDIS Driver.....	11
2.4. USB ECM Driver	11
3 Typical Usage	11
3.1. Test AT Commands.....	11
3.2. Use PPP Connection	12
3.2.1.How Does a PPP Dial-Up Connection Work?	12
3.3. Use ECM Connection	16
3.4. Use RNDIS Connection	16
3.5. Use NDIS Connection by AT\$QCRMCall.....	17
3.6. QMI	18
3.7. Concurrence of the internal TCPIP and external data call.....	19
4 Troubleshooting	19
5 Appendix and Abbreviations	19
Table 1: Terms and Abbreviations	19

1 Introduction

1.1. Scope

This user guide serves the following purpose:

- Short introductions how to customize the USB driver for SIM7500/SIM7600 module in Linux OS
- Describes how software developers can use Linux devices for typical use cases.

Typical use cases for SIM7500/SIM7600 in Linux using USB.

- PPP Dialup by USB serial (/dev/ttyUSB3 for example, VID 1E0E/PID 9001).
- NDIS Dialup by AT\$QCRMCALL(need configure the USB serial and USB wwan drivers on Linux, VID 1E0E/PID 9001)
- NDIS Dialup by QMI app (need configure the USB cdc-wdm and USB wwan drivers on Linux, VID 1E0E/PID 9001)
- RNDIS mode (The SIM7500/SIM7600 VID 1E0E/PID 9011)
- ECM mode(The SIM7500/SIM7600 VID 1E0E/PID 9018)

1.2. Related Documents

1: SIM7600 ATC.

2 Device Driver Configuration

In order to use the SIM7500/SIM7600 in Linux, the user will configure the drivers for SIM7500/SIM7600. For different use case, the user will configure different usb drivers. The usb drivers include: usb-serial, cdc-wdm , usb wwan, rndis_host and cdc-ether(ecm).

2.1 USB Serial Driver

The usb serial driver for SIM7500/SIM7600 is in drivers/usb/serial/option.c in Linux kernel source tree. So the user will modify this source file by adding vid and pid of the SIM7500/SIM7600 and do something else like skipping some interface by adding blacklist. For different use case, the user will add different VID and PID.

Add the definition of the VID/PID.

```
#define SIM7500_SIM7600_VID 0x1e0e
```

```
#define SIM7500_SIM7600_PID_9001      0x9001
#define SIM7500_SIM7600_PID_9011      0x9011
#define SIM7500_SIM7600_PID_9018      0x9018
```

Add the VID and PID in list including the blacklist.

```
static const struct option_blacklist_info sim7500_sim7600_9001_blacklist = {
    .reserved = BIT(5),
};

static const struct option_blacklist_info sim7500_sim7600_9011_blacklist = {
    .reserved = BIT(0)|BIT(1),
};

static const struct option_blacklist_info sim7500_sim7600_9018_blacklist = {
    .reserved = BIT(0)|BIT(1),
};

.....

{ USB_DEVICE(SIM7500_SIM7600_VID, SIM7500_SIM7600_PID_9001),
    .driver_info = (kernel_ulong_t)&sim7500_sim7600_9001_blacklist
},

{ USB_DEVICE(SIM7500_SIM7600_VID, SIM7500_SIM7600_PID_9011),
    .driver_info = (kernel_ulong_t)&sim7500_sim7600_9011_blacklist
},

{ USB_DEVICE(SIM7500_SIM7600_VID, SIM7500_SIM7600_PID_9018),
    .driver_info = (kernel_ulong_t)&sim7500_sim7600_9018_blacklist
},
```

After modifying the source code, you will recompile your kernel and use the new kernel,
The below kernel messages will print out if the Linux drives the SIM7500/SIM7600 correctly.

```
usb 1-1: new high speed USB device using rt3xxx-ehci and address 2
option 1-1:1.0: GSM modem (1-port) converter detected
usb 1-1: GSM modem (1-port) converter now attached to ttyUSB0
option 1-1:1.1: GSM modem (1-port) converter detected
usb 1-1: GSM modem (1-port) converter now attached to ttyUSB1
option 1-1:1.2: GSM modem (1-port) converter detected
```

```
usb 1-1: GSM modem (1-port) converter now attached to ttyUSB2
option 1-1:1.3: GSM modem (1-port) converter detected
usb 1-1: GSM modem (1-port) converter now attached to ttyUSB3
option 1-1:1.4: GSM modem (1-port) converter detected
usb 1-1: GSM modem (1-port) converter now attached to ttyUSB4
```

If your kernel code is different from the above, please be careful to modify it as appropriate. You will carefully study your source code in your Linux kernel version.

2.2. USB NDIS Driver

The SIM7500/SIM7600 is a Qualcomm-based module which supports the Qualcomm qmi. The Linux kernel (higher than version 3.3) supports the qmi_wwan driver for Qualcomm-based devices. The qmi_wwan driver includes the cdc-wdm subdriver which supports the qmi messages between the Linux and module. But the original qmi_wwan driver needs some modification for SIM7500/SIM7600.

2.2.1 Add VID/PID for interface 5.

The USB interface 5 is for USB NDIS with SIM7500/SIM7600. So you will add the VID(1E0E) PID(9001) and interface 5 in your qmi_wwan usb ids. Depending on the version of the Linux kernel, the change code may be a little different.

```
struct usb_device_id products[] = {
#ifdef 1 //Added for SIM7500/SIM7600
#ifndef QMI_FIXED_INTF
/* map QMI/wwan function by a fixed interface number */
#define QMI_FIXED_INTF(vend, prod, num) \
.match_flags = USB_DEVICE_ID_MATCH_DEVICE | \
USB_DEVICE_ID_MATCH_INT_INFO, \
.idVendor = vend, \
.idProduct = prod, \
.bInterfaceClass = 0xff, \
.bInterfaceSubClass = 0xff, \
.bInterfaceProtocol = 0xff, \
.driver_info = (unsigned long)&qmi_wwan_force_int##num,
```

```
#endif  
  
{ QMI_FIXED_INTF(0x1E0E, 0x9001, 5) }, /* SIM7500/SIM7600 */  
  
#endif
```

2.2.2 Add support for Raw IP mode

The SIM7500/SIM7600 only supports the raw ip mode. So the qmi_wwan.c will be changed for supports the raw ip mode. The changes in code refer to below

```
#include <linux/usb/usbnet.h>  
#include <linux/usb/cdc-wdm.h>  
  
#if 1 //Added for SIM7500/SIM7600  
#include <linux/etherdevice.h>  
  
struct sk_buff *qmi_wwan_tx_fixup(struct usbnet *dev, struct sk_buff *skb, gfp_t  
flags)  
{  
    if (dev->udev->descriptor.idVendor != cpu_to_le16(0x2C7C))  
        return skb;  
  
    // Skip Ethernet header from message  
    if (skb_pull(skb, ETH_HLEN)) {  
        return skb;  
    } else {  
        dev_err(&dev->intf->dev, "Packet Dropped ");  
    }  
  
    // Filter the packet out, release it  
    dev_kfree_skb_any(skb);  
    return NULL;  
}  
  
#include <linux/version.h>  
#if (LINUX_VERSION_CODE < KERNEL_VERSION( 3,9,1 ))  
static int qmi_wwan_rx_fixup(struct usbnet *dev, struct sk_buff *skb)  
{  
    __be16 proto;
```

```
if (dev->udev->descriptor.idVendor != cpu_to_le16(0x2C7C))
    return 1;
/* This check is no longer done by usbnet */
if (skb->len < dev->net->hard_header_len)
    return 0;
switch (skb->data[0] & 0xf0) {
case 0x40:
    proto = htons(ETH_P_IP);
    break;
case 0x60:
    proto = htons(ETH_P_IPV6);
    break;
case 0x00:
    if (is_multicast_ether_addr(skb->data))
        return 1;
/* possibly bogus destination - rewrite just in case */
    skb_reset_mac_header(skb);
    goto fix_dest;
default:
/* pass along other packets without modifications */
    return 1;
}
if (skb_headroom(skb) < ETH_HLEN)
    return 0;
skb_push(skb, ETH_HLEN);
skb_reset_mac_header(skb);
eth_hdr(skb)->h_proto = proto;
memset(eth_hdr(skb)->h_source, 0, ETH_ALEN);
fix_dest:
    memcpy(eth_hdr(skb)->h_dest, dev->net->dev_addr, ETH_ALEN);
return 1;
}
/* very simplistic detection of IPv4 or IPv6 headers */
static bool possibly_iphdr(const char *data)
```



```
{
    return (data[0] & 0xd0) == 0x40;
}
#endif
#endif
.....

/* if follow function exist, modify it as below */
static int qmi_wwan_bind(struct usbnet *dev, struct usb_interface *intf)
{
    .....
    #if 1 //Added for SIM7500/SIM7600
        if (dev->udev->descriptor.idVendor == cpu_to_le16(0x1E0E)) {
            dev_info(&intf->dev, "SIM7500/SIM7600 RawIP mode\n");
            dev->net->flags |= IFF_NOARP;
        }
        #if (LINUX_VERSION_CODE < KERNEL_VERSION( 3,9,1 ))
            /* make MAC addr easily distinguishable from an IP header */
            if (possibly_iphdr(dev->net->dev_addr)) {
                dev->net->dev_addr[0] |= 0x02; /* set local assignment bit */
                dev->net->dev_addr[0] &= 0xbf; /* clear "IP" bit */
            }
        #endif
        usb_control_msg(
            interface_to_usbdev(intf),
            usb_sndctrlpipe(interface_to_usbdev(intf), 0),
            0x22, //USB_CDC_REQ_SET_CONTROL_LINE_STATE
            0x21, //USB_DIR_OUT | USB_TYPE_CLASS | USB_RECIP_INTERFACE
            1, //active CDC DTR
            intf->cur_altsetting->desc.bInterfaceNumber,
            NULL, 0, 100);
    }
#endif
err:
return status;
}
```

```
.....

/* if follow function exist, modify it as below */
static int qmi_wwan_bind_shared(struct usbnet *dev, struct usb_interface *intf)
{
    .....

    #if 1 //Added for SIM7500/SIM7600
        if (dev->udev->descriptor.idVendor == cpu_to_le16(0x2C7C)) {
            dev_info(&intf->dev, "SIM7500/SIM7600 work on RawIP mode\n");
            dev->net->flags |= IFF_NOARP;
        }
        #if (LINUX_VERSION_CODE < KERNEL_VERSION( 3,9,1 ))
            /* make MAC addr easily distinguishable from an IP header */
            if (possibly_iphdr(dev->net->dev_addr)) {
                dev->net->dev_addr[0] |= 0x02; /* set local assignment bit */
                dev->net->dev_addr[0] &= 0xbf; /* clear "IP" bit */
            }
        #endif

        usb_control_msg(
            interface_to_usbdev(intf),
            usb_sndctrlpipe(interface_to_usbdev(intf), 0),
            0x22, //USB_CDC_REQ_SET_CONTROL_LINE_STATE
            0x21, //USB_DIR_OUT | USB_TYPE_CLASS | USB_RECIP_INTERFACE
            1, //active CDC DTR
            intf->cur_altsetting->desc.bInterfaceNumber,
            NULL, 0, 100);
    }
    #endif

    err:
    return status;
}

.....

/* if follow struct exist, modify it as below */
static const struct driver_info qmi_wwan_info =
{
    .....
```

```
#if 1 //Added for SIM7500/SIM7600

.tx_fixup = qmi_wwan_tx_fixup,
.rx_fixup = qmi_wwan_rx_fixup,
#endif
}

.....

/* if follow struct exist, modify it as below */
static const struct driver_info qmi_wwan_force_int5 = {
.....

#if 1 //Added for SIM7500/SIM7600

.tx_fixup = qmi_wwan_tx_fixup,
.rx_fixup = qmi_wwan_rx_fixup,
#endif

};

/* if follow struct exist, modify it as below */
static const struct driver_info qmi_wwan_shared = {
.....

#if 1 //Added for SIM7500/SIM7600

.tx_fixup = qmi_wwan_tx_fixup,
.rx_fixup = qmi_wwan_rx_fixup,
#endif

};
```

2.2.3 Kernel messages

After modification on the qmi_wwan source code, you will recompile your Linux kernel and use it. If the module is attached correctly, the kernel messages will be like below:

```
[573509.735438] qmi_wwan 1-3:1.5: cdc-wdm0: USB WDM device
[573509.736052] qmi_wwan 1-3:1.5 wwan0: register 'qmi_wwan' at usb-0000:00:14.0-3,
WWAN/QMI device, 76:ec:f0:c2:f3:44
```

Depending on your Linux kernel version, you may study your Linux kernel source code for qmi_wwan. In the latest kernel version, may the raw ip mode can be dynamically configured.

Normally, the /dev/cdc-wdmX and wwanX will be generated, but some systems may be different.

2.3. USB RNDIS Driver

The SIM7500/SIM7600 can support the RNDIS mode. If you want to use the RNDIS mode, you will send an AT command first AT+CUSBPIDSWITCH=9011,1,1 and the module will reboot automatically and switch to RNDIS mode (PID 9011).

In RNDIS mode, the module will establish the dialup automatically and start the DHCP server with GW 192.168.225.1. The Linux host will start DHCP client to get IP address.

2.4. USB ECM Driver

The SIM7500/SIM7600 can support the ECM mode. If you want to use the ECM mode, you will send an AT command first AT+CUSBPIDSWITCH=9018,1,1 and the module will reboot automatically and switch to ECMmode (PID 9018).

In ECM mode, the module will establish the dialup automatically and start the DHCP server with GW 192.168.225.1. The Linux host will start DHCP client to get IP address.

3 Typical Usage

This chapter mainly introduces several commonly used dialing methods and their general processes.

USB devices must be recognized before use modem.

3.1. Test AT Commands

```
#cat /dev/ttyUSB2 &
#echo -e "at\r\n">/dev/ttyUSB2
#
OK
```

3.2. Use PPP Connection

3.2.1. How Does a PPP Dial-Up Connection Work?

You will need the right software and a couple of pieces of information before you start. First, check the pppd. If the programs do not exist, you can download the source code from <https://ppp.samba.org/download.html> and port them to your embedded development environment. Next you must write configuration file for pppd.

3.2.1.1. Chat Scription

```
#named simcom-connect-chat and place in /etc/ppp/peers
ABORT "BUSY"
ABORT "NO CARRIER"
ABORT "NO DIALTONE"
ABORT "ERROR"
ABORT "NO ANSWER"
TIMEOUT 30
"" AT
OK ATE0
OK ATI;+CSQ;+CSQ;+CPIN?;+COPS?;+CGREG?;&D2
# Insert the APN provided by your network operator, default apn is 3gnet
OK AT+CGDCONT=1,"IP","3gnet",,0,0
OK ATD*99#
CONNECT
```

```
#named simcom-disconnect-chat and place in /etc/ppp/peers
ABORT "ERROR"
ABORT "NO DIALTONE"
SAY "\nSending break to the modem\n"
"" +++
"" +++
"" +++
SAY "\nGoodbay\n"
```

3.2.1.2. Configure dialing and AT port

```
# named simcom-pppd and place in /etc/ppp/peers
/dev/ttyUSB3 115200

#Insert the username and password for authentication, default user and password are
test
user "test" password "test"

# The chat script, customize your APN in this file
connect 'chat -s -v -f /etc/ppp/peers/simcom-connect-chat'

# The close script
disconnect 'chat -s -v -f /etc/ppp/peers/simcom-disconnect-chat'

# Hide password in debug messages
hide-password

# The phone is not required to authenticate
noauth

# Debug info from pppd
debug

# If you want to use the HSDPA link as your gateway
defaultroute

# pppd must not propose any IP address to the peer
noipdefault

# No ppp compression
novj
novjccomp
noccip

ipcp-accept-local
ipcp-accept-remote
local

# For sanity, keep a lock on the serial line
lock

modem

dump

nodetach

# Hardware flow control
```

```
nocrtscts
remotename 3gppp
ipparam 3gppp
ipcp-max-failure 30
# Ask the peer for up to 2 DNS server addresses
usepeerdns
```

3.2.1.3. Dial-Up Connection

```
# pppd call simcom-pppd &
```

When you see the output below, it shows that dial-up succeeded.

```
Connect: ppp0 <--> /dev/ttyUSB3
sent [LCP ConfReq id=0x1 <asynmap 0x0><magic 0x5107d141><pcomp><accomp>]
rcvd [LCP ConfReq id=0x0 <asynmap 0x0><auth chap MD5><magic
0x9a5c1936><pcomp><accomp>]
sent [LCP ConfAck id=0x0 <asynmap 0x0><auth chap MD5><magic
0x9a5c1936><pcomp><accomp>]
rcvd [LCP ConfAck id=0x1 <asynmap 0x0><magic 0x5107d141><pcomp><accomp>]
sent [LCP EchoReq id=0x0 magic=0x5107d141]
rcvd [LCP DiscReq id=0x1 magic=0x9a5c1936]
rcvd [CHAP Challenge id=0x1 <dd93b9f04d75e2bbba3786f6d24df3d7>, name =
"UMTS_CHAP_SRVR"]
sent [CHAP Response id=0x1 <498d4d7cf3b59dacfc07a45ce6eb7e26>, name = "test"]
rcvd [LCP EchoRep id=0x0 magic=0x9a5c1936 51 07 d1 41]
rcvd [CHAP Success id=0x1 ""]
CHAP authentication succeeded
CHAP authentication succeeded
sent [IPCP ConfReq id=0x1 <addr 0.0.0.0><ms-dns1 0.0.0.0><ms-dns2 0.0.0.0>]
rcvd [IPCP ConfReq id=0x0]
sent [IPCP ConfNak id=0x0 <addr 0.0.0.0>]
rcvd [IPCP ConfNak id=0x1 <addr 10.51.68.23><ms-dns1 222.66.251.8><ms-dns2
116.236.159.8>]
sent [IPCP ConfReq id=0x2 <addr 10.51.68.23><ms-dns1 222.66.251.8><ms-dns2
```

```
116.236.159.8>]
rcvd [IPCP ConfReq id=0x1]
sent [IPCP ConfAck id=0x1]
rcvd [IPCP ConfAck id=0x2 <addr 10.51.68.23><ms-dns1 222.66.251.8><ms-dns2
116.236.159.8>]

Could not determine remote IP address: defaulting to 10.64.64.64
local IP address 10.51.68.23
remote IP address 10.64.64.64
primary DNS address 222.66.251.8
secondary DNS address 116.236.159.8
Script /etc/ppp/ip-up started (pid 6616)
Script /etc/ppp/ip-up finished (pid 6616), status = 0x0
```

Now PPP call is set up successfully. Please use following commands to check IP/DNS/Route.

```
# ifconfig ppp0
ppp0      Link encap:Point-to-Point Protocol
          inet addr:10.216.159.39 P-t-P:10.64.64.64 Mask:255.255.255.255
          UP POINTOPOINT RUNNING NOARP MULTICAST MTU:1500 Metric:1
          RX packets:9 errors:0 dropped:0 overruns:0 frame:0
          TX packets:9 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:3
          RX bytes:362 (362.0 B) TX bytes:316 (316.0 B)

# cat /etc/resolv.conf
nameserver 221.180.132.108

# route -n
Kernel IP routing table
Destination        Gateway            Genmask           Flags Metric Ref    Use Iface
0.0.0.0            0.0.0.0           0.0.0.0           U        0      0        0 ppp0
10.64.64.64       0.0.0.0           255.255.255.255  UH       0      0        0 ppp0

# ping baidu.com
PING baidu.com (220.181.57.216) 56(84) bytes of data.
```



```
64 bytes from 220.181.57.216: icmp_seq=1 ttl=50 time=84.0 ms
64 bytes from 220.181.57.216: icmp_seq=2 ttl=50 time=34.2 ms
```

Following commands can be used to terminate PPPD process to disconnect a PPP call:

```
# killall pppd
```

3.3. Use ECM Connection

Enable ECM

```
# cat /dev/ttyUSB2 &
# echo -e "AT+CUSBPIDSWITCH=9018,1,1\r"/dev/ttyUSB2
# OK
```

Please use following commands to check IP/DNS/Route.

```
# ifconfig usb0
usb0      Link encap:Ethernet  HWaddr ae:68:46:d6:b2:80
          inet addr:192.168.225.100  Bcast:192.168.0.255  Mask:255.255.255.0
          inet6 addr: fe80::ac68:46ff:fed6:b280/64  Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:45 errors:0 dropped:0 overruns:0 frame:0
          TX packets:104 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:4237 (4.2 KB)  TX bytes:13148 (13.1 KB)

# ping baidu.com
PING baidu.com (123.125.114.144) 56(84) bytes of data.
64 bytes from 123.125.114.144: icmp_seq=1 ttl=56 time=114 ms
64 bytes from 123.125.114.144: icmp_seq=2 ttl=56 time=58.6 ms
64 bytes from 123.125.114.144: icmp_seq=3 ttl=56 time=45.1 ms
```

3.4. Use RNDIS Connection

```
# cat /dev/ttyUSB2 &
```

```
# echo -e "AT+CUSBPIDSWITCH=9011,1,1\r"/dev/ttyUSB2#  
# OK
```

Please use following commands to check IP/DNS/Route.

```
# ifconfig usb0  
usb0      Link encap:Ethernet  HWaddr ae:68:46:d6:b2:80  
          inet addr:192.168.225.100  Bcast:192.168.0.255  Mask:255.255.255.0  
          inet6 addr: fe80::ac68:46ff:fed6:b280/64  Scope:Link  
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1  
          RX packets:45 errors:0 dropped:0 overruns:0 frame:0  
          TX packets:104 errors:0 dropped:0 overruns:0 carrier:0  
          collisions:0 txqueuelen:1000  
          RX bytes:4237 (4.2 KB)  TX bytes:13148 (13.1 KB)  
  
# ping baidu.com  
PING baidu.com (220.181.38.148) 56(84) bytes of data.  
64 bytes from 220.181.38.148: icmp_seq=1 ttl=50 time=94.8 ms  
64 bytes from 220.181.38.148: icmp_seq=2 ttl=50 time=135 ms  
64 bytes from 220.181.38.148: icmp_seq=3 ttl=50 time=61.9 ms
```

3.5. Use NDIS Connection by AT\$QCRMCALL

SIM7500/SIM7600 can support NDIS data call by AT command (AT\$QCRMCALL).
The description of this AT command is below.

```
AT$QCRMCALL=<action>, <instance> [,<IP type> [,<Tech pref> [,<umts_profile>  
[,<cdma_profile> [,<APN> [,<username>,<password> [,<auth pref]]]]]]
```

<action>

0- Stop the NDIS data call

1- Start the NDIS data all

<instance>

1- Rmnet instance // please set 1 if you do not need multi-PDN

<Ip type>

```
1- IPV4
2- IPV6
3- IPV4V6
<Tech pref>
1-3GPP2
2-3GPP
<umts profile>
1-16
<cdma profile>
1-200
<APN>
String of APN
<username>
username for authentication,
<password>
password for authentication,
<auth pref>
0-NONE
2-PAP
3-CHAP
4-PAP or CHAP
```

If the user want to NDIS setup, please following the steps:

- 1 Check the module if it is registered the network
- 2 If the module is registered network, using AT+QCRMCALL=1,1 to establish the data call.
- 3 Start the DHCP client to get the ip address.

3.6. QMI

SIM7500/SIM7600 supports Qualcomm QMI messages. The Linux can send and receive the QMI Messages by the /dev/cdc-wdmX.

The user would better get the open-source QMI applications like libqmi and uqmi.

3.7. Concurrence of the internal TCPIP and external data call

Some users would use multi-PDN with the usage of concurrence of the internal TCPIP and external data call. In this case please read the below notes:

~~~The module prebuilt profile 2 and 3 is reserved for ims(volte), so please skip these two profiles  
 ~~~The module prebuilt profile 6 is used for RNDIS or ECM data call. So if please do not delete it if you want to use RNDIS or ECM mode.

4 Troubleshooting

If Linux does not create devices, check for the kernel module:

```
# lsmod | grep option
```

If entries aren't found, load the kernel module with root privileges:

```
# modprobe option
```

Check dmesg output to see that the radio was detected:

```
# dmesg | grep option
```

Check dmesg output to see that the radio was detected:

```
# dmesg | grep option

[ 16.672003] usbcore: registered new interface driver option
[ 16.672105] option 2-1.2:1.0: GSM modem (1-port) converter detected
[ 16.672216] option 2-1.2:1.1: GSM modem (1-port) converter detected
[ 16.672292] option 2-1.2:1.2: GSM modem (1-port) converter detected
[ 16.672365] option 2-1.2:1.3: GSM modem (1-port) converter detected
[ 16.672438] option 2-1.2:1.4: GSM modem (1-port) converter detected
```

If this returns an error response, the kernel module is not on your system. You will need to build the driver

5 Appendix and Abbreviations

Table 1: Terms and Abbreviations

| Abbreviation | Description |
|-----------------------------------|-------------|
| A7600_Linux_USB_User_Guide19 / 21 | |

| | |
|------|--|
| USB | Universal Serial Bus |
| PPP | Point-to-Point Protocol. The Point-to-Point Protocol is designed for simple links which transport packets between two ports. These links provide full-duplex simultaneous bi-directional operation, and are assumed to deliver packets in order. It is intended that PPP provides a common solution for easy connection of a wide variety of hosts, bridges and routers. |
| IPCP | IP Control Protocol |
| IP | Internet Protocol |
| DNS | Domain Name Server |
