Telit wireless solutions

QC Linux USB Driver - User Guide

1vv0300804 r5 11/19/2013

Making machines talk.

Disclaimer

Applicable Products

Product	Supported
UC864-E	√
UC864-G	√
UC864-WDU	√
UC864-WD	√
UC864-E-AUTO	√
UC864-K	√
CC864-DUAL	√
CC864-SINGLE	√
CC864-KPS	√
DE910-DUAL	√
CE910-DUAL	√
LE920	√
HE920	√
UE910-V2	√
HE910-V2	√

Contents

1 Introduction

1.1 Scope

This user guide serves the following purpose:

- Provides details about UC/CC/DE/CE/LE and HE920 modems
- Explains how to compile and install the Linux USB option and qmi_wwan drivers
- Describes how software developers can use the functions of Linux device drivers to configure, manage and use Telit modems

1.2 Audience

This User Guide is intended for software developers who develop applications using an UC/CC/DE/CE/LE modem and Linux.

1.3 Contact Information, Support

Our aim is to make this guide as helpful as possible. Keep us informed of your comments and suggestions for improvements.

For general contact, technical support, report documentation errors and to order manuals, contact Telit's Technical Support Center at:

TS-EMEA@telit.com
TS-NORTHAMERICA@telit.com
TS-LATINAMERICA@telit.com
TS-APAC@telit.com

Alternatively, use:

http://www.telit.com/en/products/technical-support-center/contact.php

Telit appreciates feedback from the users of our information.

1.4 Product Overview

UC864 modules contain a fully featured HSDPA modem, compatible with the other Telit GSM/GPRS modules.

CC864 modules contain a fully featured CDMA modem compatible with the other Telit CDMA modules.

The DE910-DUAL is a dual band 1xEV-DO Rev. A 3G wireless module supporting downlink data rates up to 3.1 Mbps and uplink data rates up to 1.8 Mbps.

The CE910-DUAL is a dual band 1x RTT wireless module supporting up/down link data rates up to 153.6kbps. It is designed to have the same form factor of its GSM/UMTS/HSPA/EVDO counterparts of the xE910family: GE910, HE910 and DE910-DUAL.

The Telit LE920 introduces a new generation of Telit automotive grade modules. It combines two high-speed cellular modes: LTE delivering 100Mbps downlink and 50Mbps uplink data rates and full fallback compatibility with HSPA+ delivering up to 42Mbs downlink and 5.76Mbps uplink rates.

The Telit HE920 is a tri-band HSPA module representative of the new Telit Automotive form factor xE920. The HE920 is a 3.5G wireless data module offering HSPA connectivity with download speeds up to 14.4 Mbps, upload speeds up to 5.76 Mbps and manufactured under ISO TS16949.

The UE910-V2 product family is a new member of the 910 family. It offers an HSDPA module, dual-band, designed on the xE910 LGA unified form factor. The UE910-V2 is a ”voice capable” device, featuring both analog or digital audio, but available also as data-only variant. In addition to pin-to-pin compatibility, also the AT command interface is fully compatible with the companion HE910 module.

The HE910 V2 modules are small, lightweight, low power consumption devices which combine the access to digital communication services in 2G and 3G networks with Land-Grid Array (LGA) form factor.

1.5 Document Organization

This manual contains the following chapters:

- “Chapter 1, Introduction” provides a scope for this manual, target audience, technical contact information, and text conventions.
- “Chapter 2, System Setup” describes how to setup the system before using the USB driver.
- “Chapter 3, Using the module” details USB device driver use and shows how software developers can use it to interact with the modem through shell commands and C programming.

How to Use

If you are new to the chosen product, it is recommended to start by reading the related Hardware User Guide and Product Description in their entirety to better understand how the modem driver works.

1.6 Text Conventions

This section lists the paragraph and font styles used for the various types of information presented in this user guide.

Format	Content
`Arial`	Linux shell commands, filesystem paths and example C source code

1.7 Related Documents

All documentation can be downloaded from Telit official web site www.telit.com if not otherwise indicated.

1.8 Document History

Revision	Date	Changes
ISSUE #0	2009-01-30	First Release
ISSUE #1	2010-01-13	Added CC864 Support Applied new layout
ISSUE #2	2012-02-13	Added DE910 Support New document organization
ISSUE #3	2012-06-12	Added CE910 Support
ISSUE #4	2013-02-20	Added LE920 and HE920 Support
ISSUE #5	2013-11-19	Added HE910-V2 and UE910-V2 Modified LE920 pid and interfaces

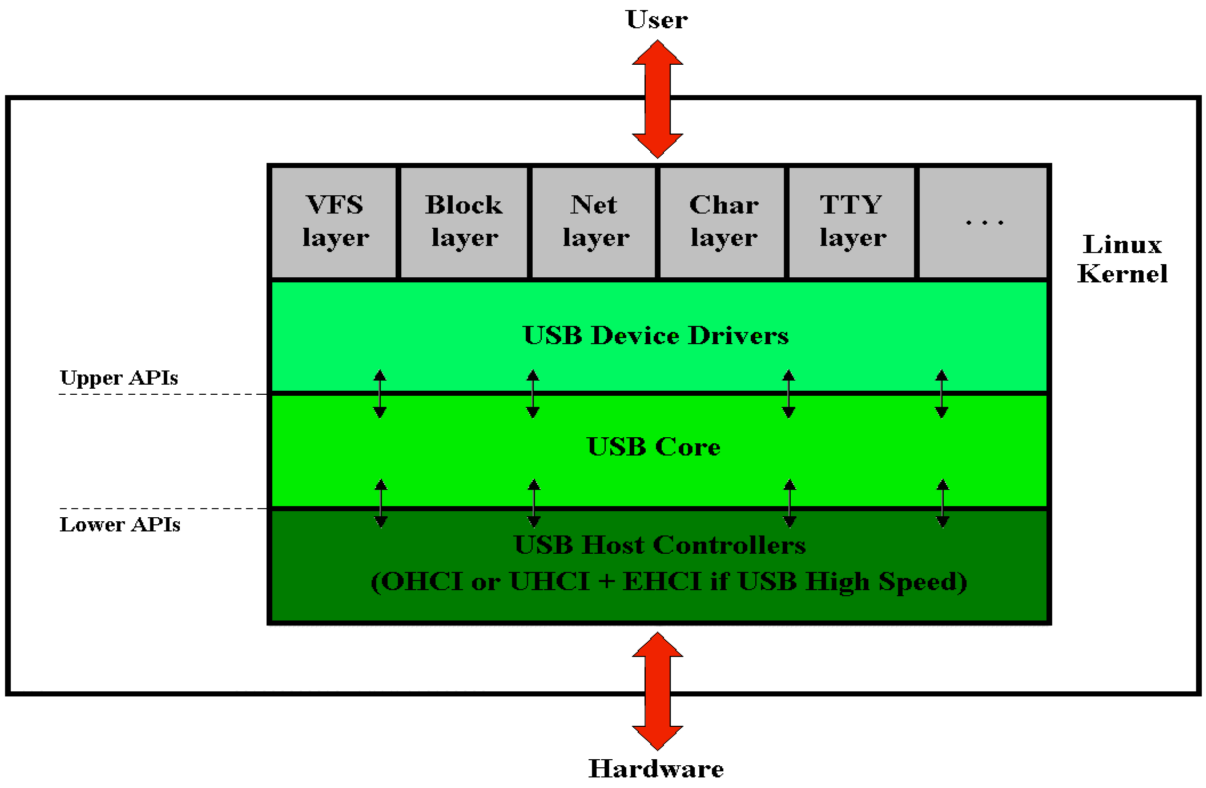
2 System Setup

In the first part of this chapter the general organization of the USB stack in Linux is described. In the second part it is explained how to setup the system for using the module.

Please note that all the instructions provided in this guide are generic: for further information refer to the documentation of your Linux distribution.

2.1 Linux USB Drivers Structure

USB drivers lie between the different kernel subsystems (block, net, char, etc.) and the USB hardware controllers. The Linux USB core provides an interface for USB drivers to use and control the hardware, without having to worry about the different types of controllers that are present on the system.



The USB Core subsystem provides specific APIs to support USB devices and host controllers. Its purpose is to abstract all hardware or device dependent parts by defining a set of data structures, macros and functions. These functions can be grouped into an upper and a lower API layer: the upper layer APIs are used by device drivers, while the lower APIs are used by the host controllers. When a device driver or a host controller is loaded in Linux using the **modprobe** command, the USB core is also automatically loaded.

2.2 Loading the drivers

2.2.1 Loading the driver: option

Linux OS includes a generic USB driver for GSM/CDMA modems in the form of a kernel module (called option): according to the considered kernel version it is possible that the driver should be customized for working well with Telit modules. Refer to chapter 2.3.1 for detailed instructions.

Most recent Linux distributions do not require any user action in order to load this driver: it is enough to simply plug the USB cable.

If the modem is recognized by the operating system some devices named /dev/ttyUSBx will be created. The number of devices varies according to the used module. For example the UC864-G presents the following devices:

/dev/ttyUSB0
/dev/ttyUSB1
/dev/ttyUSB2
/dev/ttyUSB3

Please note that AT commands are allowed on Modem and Auxiliary ports. Refer to the following table for port setup details:

UC864-E, UC864-E-AUTO, UC864-K, UC864-WD, UC864-WDU, CC864-SINGLE, CC864-KPS	ttyUSB0 → Modem port ttyUSB1 → Diagnostic port ttyUSB2 → Auxiliary port
UC864-G, CC864-DUAL,	ttyUSB0 → Modem port ttyUSB1 → Diagnostic port ttyUSB2 → NMEA port ttyUSB3 → Auxiliary port
DE910-DUAL, HE920, HE910-V2	ttyUSB0 → Diagnostic port ttyUSB1 → NMEA port ttyUSB2 → Auxiliary port ttyUSB3 → Modem port

CE910-DUAL	ttyUSB0 → Diagnostic port ttyUSB1 → Modem port
LE920	ttyUSB0 → Diagnostic port ttyUSB1 → NMEA port ttyUSB2 → Modem port 1 ttyUSB3 → Modem port 2 ttyUSB4→ SAP port
UE910-V2	ttyUSB0 → Diagnostic port ttyUSB1 → Auxiliary port ttyUSB2 → Modem port

If no devices are created in your system check for the existence of the kernel module:

```
# lsmod | grep option
```

If no entries are found, load the kernel module, with root privileges:

```
# modprobe option
```

If an error response is returned, such as:

```
# FATAL: Module option not found
```

the meaning is that the kernel module is not present in your system and it should be built. Refer to the next chapter for generic instructions.

2.2.2 Loading the driver: qmi_wwan

The LE920 exports also a network interface that can be used when the driver qmi_wwan is available (since the 3.4 version of the vanilla kernel). According to the considered kernel version it is possible that the driver should be customized for working well with Telit modules. Refer to chapter 2.3.2 for detailed instructions.

If the driver is correctly loaded by the operating system a network interface named wwanx (e.g. wwan0) will be created.

For checking the network interface presence, the ifconfig command can be used:

```
ifconfig –a
```

If no network interface is created in your system check for the existence of the kernel module:

```
# lsmod | grep qmi_wwan
```

If no entry is found, load the kernel module, with root privileges:

```
# modprobe qmi_wwan
```

If an error response is returned, such as:

```
# FATAL: Module qmi_wwan not found
```

the meaning is that the kernel module is not present in your system and it should be built. Refer to the next chapter for generic instructions.

2.3 Building the drivers

2.3.1 Customizing the driver: option

If the modem is not recognized by the system it is possible that the driver should be customized for working correctly with Telit modules.

Retrieve the appropriate kernel source code version for your system (preferably with the distribution package system, if any) and unpack/install it. From the unpacking root directory open the file:

/drivers/usb/serial/option.c

Check for the existence of the proper #define statement related to your module, according to the following table:

UC864-E, UC864-E-AUTO, UC864-K, UC864-WD, UC864-WDU	#define TELIT_PRODUCT_UC864E 0x1003
UC864-G	#define TELIT_PRODUCT_UC864G 0x1004
CC864-DUAL	#define TELIT_PRODUCT_CC864_DUAL 0x1005
CC864-SINGLE, CC864-KPS	#define TELIT_PRODUCT_CC864_SINGLE 0x1006
CC864-K	
DE910-DUAL, HE920, HE910-V2	#define TELIT_PRODUCT_DE910_DUAL 0x1010
CE910-DUAL	#define TELIT_PRODUCT_CE910_DUAL 0x1011
LE920	#define TELIT_PRODUCT_LE920 0x1201
UE910-V2	#define TELIT_PRODUCT_UE910_V2 0x1012

If not present, add the statement.

Then add the proper line in the usb_device_id option_ids[] structure:, according to the following table:

UC864-E, UC864-E-AUTO, UC864-	{ USB_DEVICE(TELIT_VENDOR_ID, TELIT_PRODUCT_UC864E) },

K, UC864-WD, UC864-WDU	
UC864-G	{ USB_DEVICE(TELIT_VENDOR_ID, TELIT_PRODUCT_UC864G) },
CC864-DUAL	{ USB_DEVICE(TELIT_VENDOR_ID, TELIT_PRODUCT_CC864_DUAL) },
CC864-SINGLE, CC864-KPS	{ USB_DEVICE(TELIT_VENDOR_ID, TELIT_PRODUCT_CC864_SINGLE) },
DE910-DUAL, HE920, HE910-V2	{ USB_DEVICE(TELIT_VENDOR_ID, TELIT_PRODUCT_DE910_DUAL) },
CE910-DUAL	{ USB_DEVICE(TELIT_VENDOR_ID, TELIT_PRODUCT_CE910_DUAL) },
UE910-V2	{ USB_DEVICE(TELIT_VENDOR_ID, TELIT_PRODUCT_UE910_V2) },

Save the changes and close the file.

2.3.1.1 Additional customization for the LE920

The LE920 requires specific customization for not letting the driver option claim other USB interfaces needed by different drivers.

The driver option should not be bound to the interfaces 1 and 5. According to the kernel version, different changes should be done in the option source code.

As an example we can consider the source code of the 3.7 kernel version.

First, add the following struct:

```
static const struct option_blacklist_info telit_le920_blacklist = {
		.reserved = BIT(1) | BIT(2),
};
```

Then, the following line should be added to the usb_device_id option_ids[] structure:

```
		{ USB_DEVICE(TELIT_VENDOR_ID, TELIT_PRODUCT_LE920),
				.driver_info = (kernel_ulong_t)&telit_le920_blacklist },
```

2.3.2 Customizing the driver: qmi_wwan

When using the LE920, if the network interface is not recognized by the system it is possible that the driver should be customized.

Retrieve the appropriate kernel source code version for your system (preferably with the distribution package system, if any) and unpack/install it. From the unpacking root directory open the file:

/drivers/net/usb/qmi_wwan.c

LE920 VID, PID and interface info should be added to the driver source code. According to the kernel

version, different changes should be done in the source code.

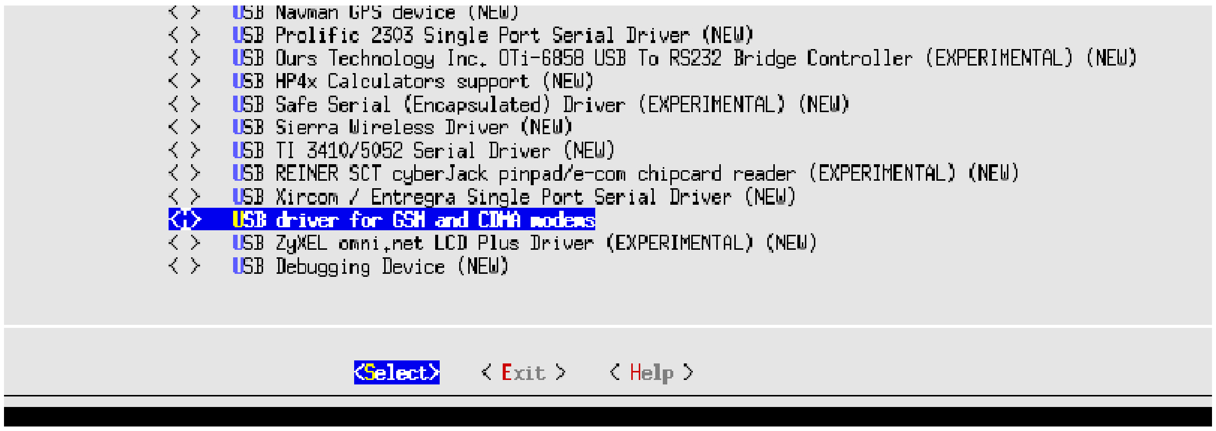
As an example we can consider the source code of the 3.7 kernel version. The following line should be added to the usb_device_id products[] structure:

```
{QMI_FIXED_INTF(0x1bc7, 0x1201, 2)},     /* Telit LE920 */
```

2.3.3 Compiling the drivers

From the unpacking root directory of your kernel run the kernel configuration system (e.g. menuconfig).

Configure the kernel according to the considered system configuration; then browse through the menus *"Device Drivers"* → *"USB Support"* → *"USB Serial Converter support"* and choose to build *"USB driver for GSM and CDMA modems"* as a module:



```
< >   USB Navman GPS device (NEW)
< >   USB Prolific 2303 Single Port Serial Driver (NEW)
< >   USB Ours Technology Inc. OTi-6858 USB To RS232 Bridge Controller (EXPERIMENTAL) (NEW)
< >   USB HP4x Calculators support (NEW)
< >   USB Safe Serial (Encapsulated) Driver (EXPERIMENTAL) (NEW)
< >   USB Sierra Wireless Driver (NEW)
< >   USB TI 3410/5052 Serial Driver (NEW)
< >   USB REINER SCT cyberJack pinpad/e-com chipcard reader (EXPERIMENTAL) (NEW)
< >   USB Xircom / Entregra Single Port Serial Driver (NEW)
<*>   USB driver for GSM and CDMA modems
< >   USB ZyXEL omni.net LCD Plus Driver (EXPERIMENTAL) (NEW)
< >   USB Debugging Device (NEW)


                    <Select>    < Exit >    < Help >
```

When using the LE920 browse through the menus *"Device Drivers"* → *"Network Device Support"* → *"USB Network Adapters"* and choose to build *"QMI WWAN driver for Qualcomm MSM based 3G and LTE modems"* as a module.

Once configured, start the build.

The kernel module option.ko can be found in the directory drivers/usb/serial, while the module qmi_wwan.ko can be found in the directory drivers/net/usb.

If the kernel has been previously already built, the module can be compiled simply typing:

```
# make M=drivers/usb/serial
```

for option or

```
# make M=drivers/net/usb
```

for qmi_wwan.

The modules can then be loaded using modprobe or insmod.

In order to avoid runtime loading, the drivers should not be built as a modules, but as parts of the kernel. **This is the suggested configuration for Android devices.**

3 Using the module

3.1 Using the driver: option

3.1.1 Shell commands

For testing the serial ports created by the driver, type in a shell (replace the name of the device with the proper one):

```
# cat /dev/ttyUSBx &
# echo –en "ATE0\r" > /dev/ttyUSBx[1]
# echo –en "AT\r" > /dev/ttyUSBx
```

in order to print on standard output the answer of the modem to the command “ATE0\r” and “AT\r”.

Please note that sending the command “ATE0” is mandatory, otherwise there could be issues in the terminal output.

3.1.2 Create a PPP connection

Most recent Linux distributions have GUI tools for creating PPP connections; the following instructions are for creating a PPP connection through command line interface.

PPP support needs to be compiled into the kernel; pppd and chat programs are also needed.

pppd needs two scripts: the first script performs the environment setting and calls the second script, used by the chat program. For creating a PPP connection type:

```
# pppd file /etc/pppd_script &
```

[1] The use of “ATE0” (echo disabled) before any other AT command is necessary because it prevents the sending/receiving of spurious characters to/from the modem when used in interaction with the Linux commands “echo” and “cat”.

Example of *pppd_script* :

```
# Debug info from pppd
debug
#kdebug 4
# Most phones don't reply to LCP echos
lcp-echo-failure 3
lcp-echo-interval 3
# Keep pppd attached to the terminal
# Comment this to get daemon mode pppd
nodetach
# The chat script (be sure to edit that file, too!)
connect "/usr/sbin/chat -v -f /etc/chatscripts/hsdpa_connect"
# Serial Device to which the phone is connected
/dev/ttyUSBx
# Serial port line speed
115200
dump
# The phone is not required to authenticate
#noauth
user <insert here the correct username for authentication>
name <insert here the name of the connection>
password <insert here the correct password for authentication>
# If you want to use the HSDPA link as your gateway
defaultroute
# pppd must not propose any IP address to the peer
#noipdefault
ipcp-accept-local
ipcp-accept-remote
# Keep modem up even if connection fails
#persist
# Hardware flow control
crtscts
# Ask the peer for up to 2 DNS server addresses
usepeerdns
# No ppp compression
novj
nobsdcomp
novjccomp
nopcomp
noaccomp
# For sanity, keep a lock on the serial line
lock
# Show password in debug messages
show-password
```

This script calls the option “*connect*” using the script “*hsdpa_connect*”; following there is an example of this script:

```
#!/bin/sh
# Connection to the network
" AT+CGDCONT=1,"IP","<insert here the correct APN provided by your network operator>"
# Dial the number.
OK ATD*99***1#
# The modem is waiting for the following answer
CONNECT "
```

After launching a PPP connection is possible to use ftp protocol or other utilities that allow the access to the Internet.

3.1.3 C programming

The following paragraphs show all the functions that can be used from C source code to perform read/write operations on the serial devices.

3.1.3.1 open()

The *open*() function shall establish the connection between a file and a file descriptor. The file descriptor is used by other I/O functions to refer to that file.

Header file:
fcntl.h

Prototype:
int open(const char *pathname, int flags)

Parameters:
pathname – file name with its own path
flags – is an *int* specifying file opening mode: is one of O_RDONLY, O_WRONLY or O_RDWR which request opening the file read-only, write-only or read/write, respectively

Returns:
The new file descriptor *fildes* if successfull, -1 otherwise

Example:
Open the /dev/ttyUSBx.

```
int fd;      // file descriptor for the /dev/ttyUSBx entry

if((fd = open("/dev/ttyUSBx", O_RDONLY) < 0)
{
        /* Error Management Routine */
} else {
        /* ttyUSBx Device Opened */
}
```

3.1.3.2 read()

The *read*() function reads *nbyte* bytes from the file associated with the open file descriptor, *fildes*, and copies them in the buffer that is pointed to by *buf*.

Header file:
unistd.h

Prototype:
ssize_t read(int fildes, void *buf, size_t nbyte)

Parameters:
fildes – file descriptor
buf – destination buffer pointer
nbyte – number of bytes that read() attempts to read

Returns:
The number of bytes actually read if the operation is completed successfully, otherwise it is -1.

Example:
Read *sizeof(read_buff)* bytes from the file associated with *fd* and stores them into *read_buff*.

```
char read_buff[BUFF_LEN];

if(read(fd, read_buff, sizeof(read_buff)) < 0)
{
      /* Error Management Routine */
} else {
      /* Value Read */
}
```

3.1.3.3 write()

The *write*() function writes *nbyte* bytes from the buffer that are pointed by *buf* to the file associated with the open file descriptor, *fildes*.

Header file:
unistd.h

Prototype:
ssize_t write(int fildes, const void *buf, size_t nbyte)

Parameters:
fildes – file descriptor
buf – destination buffer pointer
nbyte – number of bytes that write() attempts to write

Returns:
The number of bytes actually written if operation is completed successfully (this number shall never be greater than *nbyte*), otherwise it is -1.

Example:
Write *strlen(value_to_be_written)* bytes from the buffer pointed by *value_to_be_written* to the file associated with the open file descriptor, *fd*.

```
char value_to_be_written[] = “dummy_write”;

if (write(fd, value_to_be_written, strlen(value_to_be_written)) < 0)
{
        /* Error Management Routine */
} else {
        /* Value Written */
}
```

3.1.3.4 close()

The *close*() function shall deallocate the file descriptor indicated by *fildes*. To deallocate means to make the file descriptor available for return by subsequent calls to *open*() or other functions that allocate file descriptors.

Header file:
unistd.h

Prototype:
int close(int fildes);

Parameters:
fildes – file descriptor

Returns:
0 if successfull, -1 otherwise

Example:
Close the ttyUSBx file.

```
if(close(fd) < 0)
{
        /* Error Management Routine */
} else {
        /* File Closed */
}
```

3.1.3.5 Test Program

The following simple C program is useful to test the modem issuing an AT command. The program opens the /dev/ttyUSBx device and calls the write() and the read() function to send an AT command and receive the subsequent output.

```
#include <stdio.h>   /* Standard input/output definitions */
#include <string.h>  /* String function definitions */
#include <unistd.h>  /* UNIX standard function definitions */
#include <fcntl.h>   /* File control definitions */
#include <errno.h>   /* Error number definitions */
#include <termios.h> /* POSIX terminal control definitions */

#define USB                     "/dev/ttyUSBx"
#define BUFSIZE         1000
#define BAUDRATE                B115200

int open_port(char *port)
{
        struct termios options;
        int fd;

        fd = open(port, O_RDWR | O_NOCTTY | O_NDELAY);

        if (fd == -1)
        {
                printf("open_port: Unable to open the port - ");
        }
        else
        {
                printf ( "Port %s with file descriptor=%i",port, fd);
                fcntl(fd, F_SETFL, FNDELAY);

                tcgetattr( fd, &options );

                cfsetispeed( &options, BAUDRATE );
                cfsetospeed( &options, BAUDRATE );

                options.c_cflag |= ( CLOCAL | CREAD);
                options.c_cflag &= ~(CSIZE | PARENB | CSTOPB | CSIZE);
                options.c_cflag |= CS8;
                options.c_cflag &= ~CRTSCTS;
                options.c_lflag &= ~(ICANON | ECHO | ECHOE | ISIG);
                options.c_iflag &= ~(IXON | IXOFF | IXANY | ICRNL | INLCR | IGNCR);
                options.c_oflag &= ~OPOST;

                if ( tcsetattr( fd, TCSANOW, &options ) == -1 )
                        printf ("Error with tcsetattr = %s\n", strerror ( errno ) );
                else
                   printf ( "%s\n", "succeed" );
        }
        return (fd);
}


int main()
{
        int serialFD = open_port(USB);
        char buf[BUFSIZE];
        memset(buf,0,BUFSIZE);

        write(serialFD, "AT\r" , strlen("AT\r"));
        sleep(1);
        read( serialFD, buf, BUFSIZE );

        printf("The string is: %s\n", buf);

        close(serialFD);

        return 0;
}
```

The “sleep” instruction is necessary because the response of the modem after issuing the command “AT” is not immediate, so you need to wait a bit before reading. Obviously there are more efficient ways to do this, that is, for example, put the “read” call in a while loop and exit when the read buffer contains a certain string.

3.2 Using the driver qmi_wwan

To use the network interface exported by the LE920 a modem manager with support for QMI is recommended.

Refer to the libqmi project (cgit.freedesktop.org/libqmi) for an open-source QMI modem protocol helper library.