HE-GE Linux USB Driver - User Guide

1vv0300924 r2 03/25/2013

Disclaimer

SPECIFICATIONS SUBJECT TO CHANGE WITHOUT NOTICE

Notice

While reasonable efforts have been made to assure the accuracy of this document, Telit assumes no liability resulting from any inaccuracies or omissions in this document, or from use of the information obtained herein. The information in this document has been carefully checked and is believed to be entirely reliable. However, no responsibility is assumed for inaccuracies or omissions. Telit reserves the right to make changes to any products described herein and reserves the right to revise this document and to make changes from time to time in content hereof with no obligation to notify any person of revisions or changes. Telit does not assume any liability arising out of the application or use of any product, software, or circuit described herein; neither does it convey license under its patent rights or the rights of others.

It is possible that this publication may contain references to, or information about Telit products (machines and programs), programming, or services that are not announced in your country. Such references or information must not be construed to mean that Telit intends to announce such Telit products, programming, or services in your country.

Copyrights

This instruction manual and the Telit products described in this instruction manual may be, include or describe copyrighted Telit material, such as computer programs stored in semiconductor memories or other media. Laws in the Italy and other countries preserve for Telit and its licensors certain exclusive rights for copyrighted material, including the exclusive right to copy, reproduce in any form, distribute and make derivative works of the copyrighted material. Accordingly, any copyrighted material of Telit and its licensors contained herein or in the Telit products described in this instruction manual may not be copied, reproduced, distributed, merged or modified in any manner without the express written permission of Telit. Furthermore, the purchase of Telit products shall not be deemed to grant either directly or by implication, estoppel, or otherwise, any license under the copyrights, patents or patent applications of Telit, as arises by operation of law in the sale of a product.

Computer Software Copyrights

The Telit and 3rd Party supplied Software (SW) products described in this instruction manual may include copyrighted Telit and other 3rd Party supplied computer programs stored in semiconductor memories or other media. Laws in the Italy and other countries preserve for Telit and other 3rd Party supplied SW certain exclusive rights for copyrighted computer programs, including the exclusive right to copy or reproduce in any form the copyrighted computer program. Accordingly, any copyrighted Telit or other 3rd Party supplied SW computer programs contained in the Telit products described in this instruction manual may not be copied (reverse engineered) or reproduced in any manner without the express written permission of Telit or the 3rd Party SW supplier. Furthermore, the purchase of Telit products shall not be deemed to grant either directly or by implication, estoppel, or otherwise, any license under the copyrights, patents or patent applications of Telit or other

3rd Party supplied SW, except for the normal non-exclusive, royalty free license to use that arises by operation of law in the sale of a product.

Usage and Disclosure Restrictions

License Agreements

The software described in this document is the property of Telit and its licensors. It is furnished by express license agreement only and may be used only in accordance with the terms of such an agreement.

Copyrighted Materials

Software and documentation are copyrighted materials. Making unauthorized copies is prohibited by law. No part of the software or documentation may be reproduced, transmitted, transcribed, stored in a retrieval system, or translated into any language or computer language, in any form or by any means, without prior written permission of Telit

High Risk Materials

Components, units, or third-party products used in the product described herein are NOT fault-tolerant and are NOT designed, manufactured, or intended for use as on-line control equipment in the following hazardous environments requiring fail-safe controls: the operation of Nuclear Facilities, Aircraft Navigation or Aircraft Communication Systems, Air Traffic Control, Life Support, or Weapons Systems (High Risk Activities"). Telit and its supplier(s) specifically disclaim any expressed or implied warranty of fitness for such High Risk Activities.

Trademarks

TELIT and the Stylized T Logo are registered in Trademark Office. All other product or service names are the property of their respective owners.

Applicable Products

Product	Supported
HE863	√
HE910	√
GE910	√

Contents

Introduction

Scope

This user guide serves the following purpose:

- Provides details about HE/GE modems
- Explains how to compile and install the Linux USB cdc-acm driver
- Describes how software developers can use the functions of Linux device drivers to configure, manage and use USB modem

Audience

This User Guide is intended for software developers who develop applications using the HE HSPA and GE modems.

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.

Product Overview

HE modules contain a fully featured dual band HSPA and quad band GSM/GPRS module.
The HE Family provides 3.75G wireless data connectivity, combining the access to HSPA network, GSM, GPRS and EDGE services with a hosted A-GPS receiver in a compact form factor.

The HE863 Family features HSDPA 7.2 Mbps (Cat 8), HSUPA 5.8 Mbps (Cat 6), EGPRS Class 33, SMS support, analog and digital audio, embedded TCP/IP protocol stack with standard and custom Telit AT commands and more. Several HE863 variants are available for different regional coverage and custom needs. Moreover, the HE Family incorporates features such as UARTs, USB 2.0 HS, 3xADC, 1xDAC and programmable GPIOs.
The HE910 Family features HSDPA 21 Mbps (Cat 14), HSUPA 5.8 (Cat6), GPRS/EDGE Class 33, SMS, GPIO digital audio and well as embedded GPS receiver.
HE910 is available in 5 HSPA+ bands for global coverage as well as in tri-band low-end regional variant.
The GE910 is the GSM/GPRS product line of Telit's xE910 Unified Form Factor Family. The GE910 provide USB 2.0 full speed interface. The GE910-QUAD features quad-band GPRS wireless data connectivity, as well as analog and digital voice. The GE910-GNSS variant adds a competitively priced GSM/GPRS plus GNSS combo solution supporting both GPS and GLONASS.
See Product description for more details.

Document Organization

This manual contains the following chapters:

- Introduction provides a scope for this manual, target audience, technical contact information, and text conventions.
- System Setup describes how to setup the system before using the USB driver.
- 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 this product, it is recommended to start by reading Telit HE863, Telit GE910 and HE910 Hardware User Guide, Telit HE863, Telit GE910 and HE910 Product Description and this document in their entirety in order to understand how the HE-GE driver works.

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

Related Documents

The following documents are related to this user guide:
[1] Telit HE863, GE910 and HE910 Hardware User Guide
[2] Telit HE863, GE910 and HE910 Product Description

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

Document History

Revision	Date	Changes
ISSUE #0	02/14/2011	First Release
ISSUE #1	11/15/2011	Second Release; support for HE910 module
ISSUE #2	03/25/2013	Third Release; support for GE910 module; modified date convention

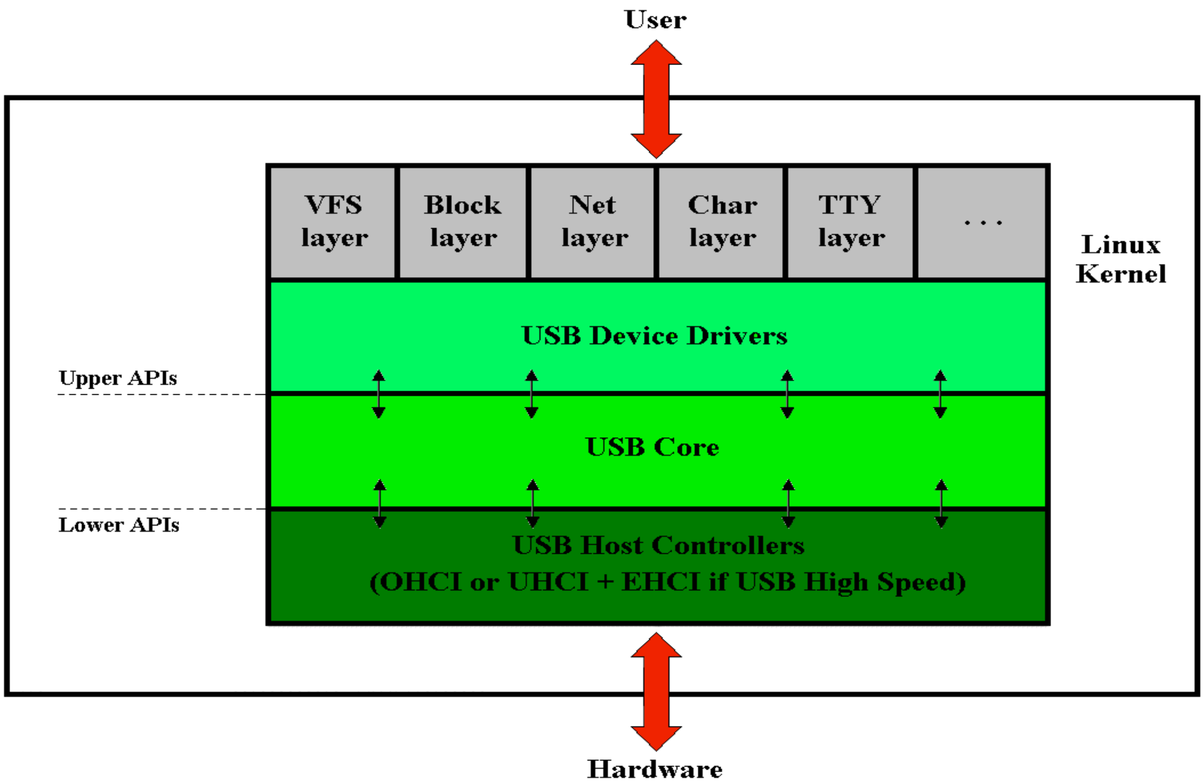
System Setup

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 HE-GE module.

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

Linux USB Drivers Structure

USB drivers live between the different kernel subsytems (block, net, char, etc.) and the USB hardware controllers. The Linux USB Core provides an interface for USB drivers to use and control the USB hardware, without having to worry about the different types of USB hardware 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.

Loading the driver

Linux OS includes a generic USB driver for modems supporting the CDC ACM specification in the form of a kernel module (called cdc-acm): this driver works well without customization for HE-GE module.

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 seven devices will be created:

`/dev/ttyACM0`
`/dev/ttyACM1`
`/dev/ttyACM2`[1]
`/dev/ttyACM3`[1]
`/dev/ttyACM4`[1]
`/dev/ttyACM5`[1]
`/dev/ttyACM6`[1]

Of those only the following devices can be used: generic port for AT commands

`/dev/ttyACM0`: data port for PPP connections and AT commands
`/dev/ttyACM3`[2]: generic port for AT commands

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

lsmod | grep cdc_acm

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

modprobe cdc-acm

If an error response is returned, such as:

FATAL: Module cdc-acm not found

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

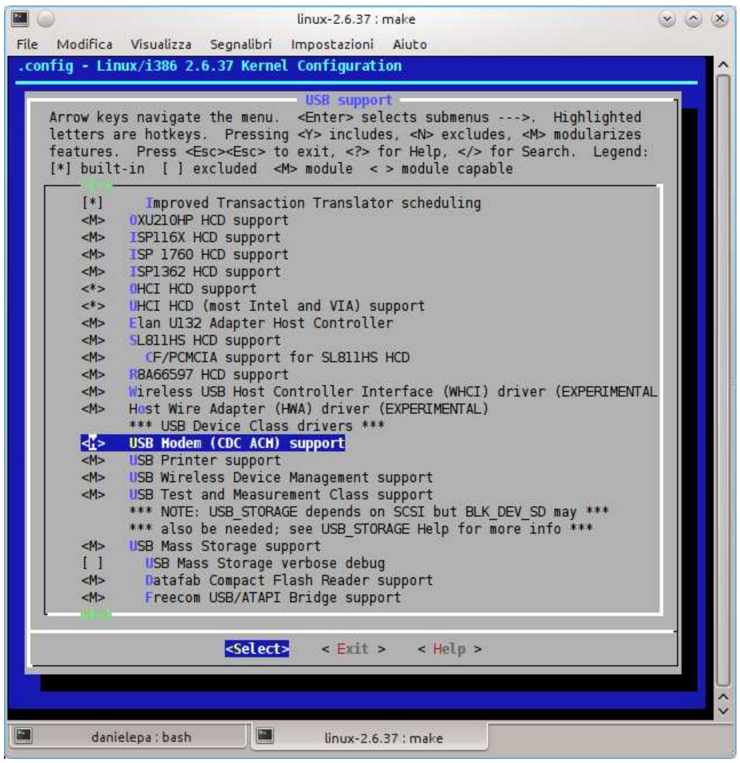
[1] Not available on GE910
[2] `/dev/ttyACM1` on GE910

Building the driver

First, retrieve the appropriate kernel source code version for your system (preferably with the distribution package system, if any). Unpack it and in its root directory type:

```
# make menuconfig
```

Configure the kernel according to the considered system configuration; then browse through the menus *“Device Driver”* → *“USB Support”* and choose to build *“USB Modem (CDC ACM) support”* as a module:



Once configured, start the build typing:

```
# make
```

The kernel module cdc-acm.ko can be found in the directory drivers/usb/class.

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

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

The module can then be loaded using modprobe or insmod.

Using the module

Using the driver

Shell commands

For testing the serial ports created by the driver, type in a shell:

```
# cat /dev/ttyACM0 &
# echo –en "ATE0\r" > /dev/ttyACM0[3]
# echo –en "AT\r" > /dev/ttyACM0
```

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.

The same test can be performed using the other interface (`ttyACM1` into GE910 or `ttyACM3` into HE).

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 &
```

[3] 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 modem is connected
/dev/ttyACM3
# 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.

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.

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/ttyACM0`.

```
int fd;      // file descriptor for the /dev/ttyACM0 entry

if((fd = open("/dev/ttyACM0", O_RDONLY) < 0)
{
       /* Error Management Routine */
} else {
```


```
        /* ttyACM0 Device Opened */
}
```

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 */
}
```

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 */
}
```

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 ttyACMx file.

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

A Test Program()

The following simple C program is useful to test the modem issuing an AT command. The program opens the /dev/ttyACM0 interface 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/ttyACM0"
#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.