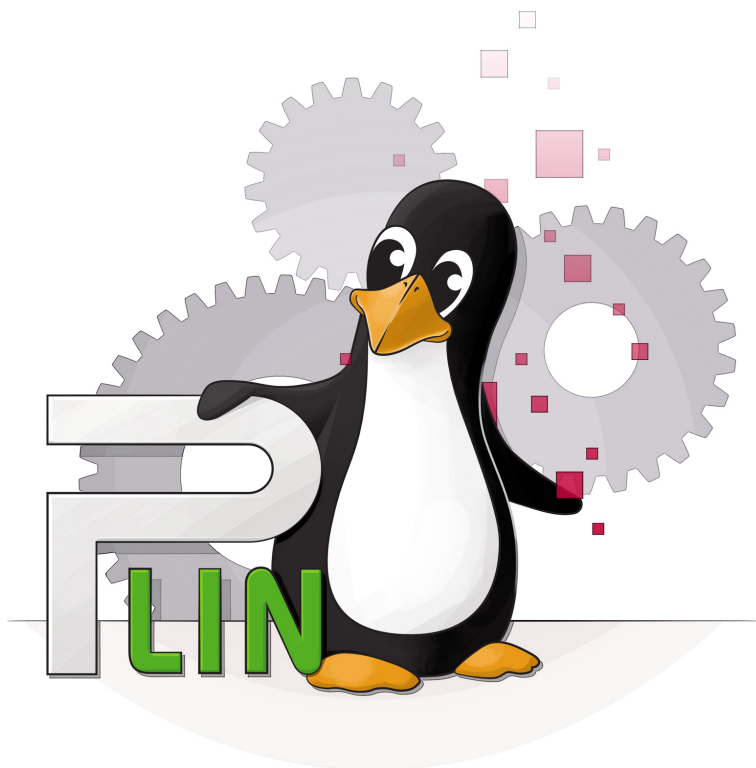


PLIN-Linux Driver

Quick Start Guide



Relevant Product

Product Name

PLIN-Linux Driver

Imprint

PCAN and PLIN are registered trademarks of PEAK-System Technik GmbH.

All other product names in this document may be the trademarks or registered trademarks of their respective companies. They are not explicitly marked by ™ or ®.

© 2024 PEAK-System Technik GmbH

Duplication (copying, printing, or other forms) and the electronic distribution of this document is only allowed with explicit permission of PEAK-System Technik GmbH. PEAK-System Technik GmbH reserves the right to change technical data without prior announcement. The general business conditions and the regulations of the license agreement apply. All rights are reserved.

PEAK-System Technik GmbH
Leydheckerstraße 10
64293 Darmstadt
Germany

Phone: +49 6151 8173-20
Fax: +49 6151 8173-29

www.peak-system.com
info@peak-system.com

Document version 1.0.0 (2024-06-04)

Contents

Imprint	2
Relevant Product	2
Contents	3
1 Introduction	4
2 Setup	5
2.1 Dependencies	5
2.2 How to compile the PLIN-Linux Driver	5
2.3 How to install and load the driver	7
3 Basics	8
3.1 Setting up a Master	8
3.2 Setting up a Slave	9
3.3 Frame Entries and Schedule Tables	10
3.4 Utils Folder	15
Appendix A ident-string / ident-num Parameter	19

1 Introduction

This Quick Start Guide will introduce you to the PLIN-Linux Driver package and the first steps necessary to get you up and running.



Note: The examples of this guide are tested on Kubuntu 19.04.

2 Setup

2.1 Dependencies

Some Linux distributions may need additional software packages installed to successfully compile the PLIN-Linux Driver package. The following requirements apply to the PLIN-Linux Driver package:

- The GCC compiler (GCC version 9.2.1 was used in this guide)
- Make utilities (GNU Make version 4.2.1 was used in this guide)
- linux-headers package (corresponding to your kernel version)
- dkms tool is used to install the PLIN driver

Please make sure the requirements are met on your system before compiling and installing the driver.

2.2 How to compile the PLIN-Linux Driver

Download the PLIN-Linux Driver package from the PEAK-System Linux website: www.peak-system.com/quick/PLIN-Linux-Driver

Unzip the folder you have just downloaded:

```
$ tar -xzf peak-lin-driver-x.y.z.tar.gz
```

Then change into the extracted directory and check the contents of the folder via:

```
$ ls
```

The extracted folder contains the following:

1. The driver folder: This folder contains the files necessary to compile the driver on your machine.
2. The utils folder: This folder will contain tools and utilities to be used with the driver once it is loaded.
3. The Makefile allows to build all the binaries from their source files ("make all") as well as to install them onto the current system ("sudo make install") or to remove them from it ("sudo make uninstall").

To build the driver, simply type:

```
$ make
```

```
~/Desktop/peak-lin-driver-1.2.0-rc6 $ make
make[1]: Entering directory '/home/peak/Desktop/peak-lin-driver-1.2.0-rc6/utils'
Building plin utilities:
gcc lin.c -c -I../driver -o lin.o
gcc lin.o -o lin
gcc linwrite.c -c -I../driver -o linwrite.o
gcc linwrite.o -o linwrite
gcc linread.c -c -I../driver -o linread.o
gcc linread.o -o linread
make[1]: Leaving directory '/home/peak/Desktop/peak-lin-driver-1.2.0-rc6/utils'
make[1]: Entering directory '/home/peak/Desktop/peak-lin-driver-1.2.0-rc6/driver'
Building driver module plin.ko from /usr/src/linux-headers-6.5.0-27-generic,
with x86_64-linux-gnu-gcc version 11, under Ubuntu 22.04:
make -C /usr/src/linux-headers-6.5.0-27-generic EXTRA_CFLAGS="" M=/home/peak/Desktop/peak-lin-driver-1.2.0-rc6/driver modules
make[2]: Entering directory '/usr/src/linux-headers-6.5.0-27-generic'
warning: the compiler differs from the one used to build the kernel
The kernel was built by: x86_64-linux-gnu-gcc-12 (Ubuntu 12.3.0-1ubuntu1-22.04) 12.3.0
You are using: gcc-12 (Ubuntu 12.3.0-1ubuntu1-22.04) 12.3.0
CC [M] /home/peak/Desktop/peak-lin-driver-1.2.0-rc6/driver/plin_main.o
CC [M] /home/peak/Desktop/peak-lin-driver-1.2.0-rc6/driver/plin_sysfs.o
CC [M] /home/peak/Desktop/peak-lin-driver-1.2.0-rc6/driver/plin_chrdev.o
CC [M] /home/peak/Desktop/peak-lin-driver-1.2.0-rc6/driver/plin_usb.o
LD [M] /home/peak/Desktop/peak-lin-driver-1.2.0-rc6/driver/plin.o
MODPOST /home/peak/Desktop/peak-lin-driver-1.2.0-rc6/driver/Module.symvers
CC [M] /home/peak/Desktop/peak-lin-driver-1.2.0-rc6/driver/plin.mod.o
LD [M] /home/peak/Desktop/peak-lin-driver-1.2.0-rc6/driver/plin.ko
BTF [M] /home/peak/Desktop/peak-lin-driver-1.2.0-rc6/driver/plin.ko
Skipping BTF generation for /home/peak/Desktop/peak-lin-driver-1.2.0-rc6/driver/plin.ko due to unavailability of vmlinux
make[2]: Leaving directory '/usr/src/linux-headers-6.5.0-27-generic'
make[1]: Leaving directory '/home/peak/Desktop/peak-lin-driver-1.2.0-rc6/driver'
```

```
~/Desktop/peak-lin-driver-1.2.0-rc6 $
```

2.3 How to install and load the driver

The driver installation is quite simple. Just run the following command:

```
$ sudo make install
```

And load the driver via:

```
$ sudo modprobe plin
```

To check if the driver was properly loaded and your device is recognized you can execute:

```
$ ls /sys/class/plin
```

```
~/Desktop/peak-lin-driver-1.2.0-rc6 $ ls /sys/class/plin  
pcan-usb_pro_fd-0 plin0 plin1 version
```



Note: Installation on a Raspberry PI needs additional steps which are outlined in chapter "Additional Information" on page 17.

3 Basics

In this chapter we will cover how to use the PLIN-Linux Driver package.

We will cover the `lin` application which basically handles all configuration of our LIN nodes. Additionally we will use `linwrite` and `linread` to use our LIN nodes once they are set up.

3.1 Setting up a Master

Every LIN bus needs a Master to handle communication. Setting up the master node with the PLIN-Linux Driver is quite simple. First, we want to know which LIN channels are available to us. We can check this via:

```
$ ls /sys/class/plin
```

In this example, we will use `plin0` as our master node.

```
$ lin start master 9600 /dev/plin0
```

To test our master node, we can write a frame to the bus using `linwrite`:

```
$ linwrite -i=0x22 -b="1 2 1 2" -c=C -d=P /dev/plin0
```

```
~/Desktop/peak-lin-driver-1.2.0-rc6 linwrite -i=0x22 -b="1 2 1 2" -c=C -d=P /dev/plin0
```

If you, for example, run Windows application PLIN-View Pro on the other end as Slave, you should be able to see this message now. But let's assume you do not have a Windows PC running PLIN-View Pro available to check this.

We now must set up a Slave to communicate to.

3.2 Setting up a Slave

Start another terminal session.

We can use the `lin` application in the same manner we already did with our Master node. It's basically the same command for both, except for the node type. We will use `plin1` as our slave node:

```
$ lin start slave 9600 /dev/plin1
```

```
~/Desktop/peak-lin-driver-1.2.0-rc6 lin start slave 9600 /dev/plin1
```

By default, the slave will not relay incoming messages to our applications, such as `linread`, so we need to open its filter first (for all frames):

```
$ lin set id-filter all-opened /dev/plin1
```

```
~/Desktop/peak-lin-driver-1.2.0-rc6 lin set id-filter all-opened /dev/plin0
```



Note: `lin set id-filter` command normally waits for a 64-bit mask made of 8 numeric values, separated with ':', each bit defining if the filter is opened (1) or closed (0) for the corresponding ID. `all-opened` as well as `all-closed` are shortcuts to open (close) all filters on every ID.

Now, reading a frame from the LIN node is done with:

```
$ linread /dev/plin1
```

If you now write a frame to the bus using your Master node like described in chapter before, you will be able to see that frame:

```
~/Desktop/peak-lin-driver-1.2.0-rc6 linread /dev/plin1
count    timestamp d ID data
-----
1 5556116016 d SLEEP
2 5565894009 S 22 01 02 01 02
3 5566997009 S 22 01 02 01 02
4 5568516009 S 22 01 02 01 02
5 5570394009 S 22 01 02 01 02
6 5571411009 S 22 01 02 01 02 _
```

3.3 Frame Entries and Schedule Tables

Let's switch back to our Master node session.

To add a specific frame to our Schedule table we need to add an entry for it first:

```
$ lin set pub-frm-entry 0x21 --cs-enhanced --len 4 /dev/plin0
```

We now have a new entry for 0x21, you can check this via:

```
$ lin get frm-entry 0x21 /dev/plin0
```

```
~/Desktop/peak-lin-driver-1.2.0-rc6 lin get frm-entry 0x21 /dev/plin0
ID (hex)  l D C FLGS DATA
-----+-----+-----+-----+-----
33 (21)  4 p e 0001 00 00 00 00
```

Let us add this frame to the schedule table. Our master will publish it:

```
$ lin add unc-schd-slot 0 100 0x21 /dev/plin0
```

Now check if the entry is there:

```
$ lin get schd-slot 0 /dev/plin0
```

```
~/Desktop/peak-lin-driver-1.2.0-rc6 lin get schd-slot 0 /dev/plin0
SLOT T   c/r delay   handle I0 I1 I2 I3 I4 I5 I6 I7
-----
0 u      0    100 200111e8 21
```

Now let us start the scheduler. Once started this runs on the hardware. Remember slot 0 is our new entry!

```
$ lin start schedule 0 /dev/plin0
```

```
~/Desktop/peak-lin-driver-1.2.0-rc6 lin start schedule 0 /dev/plin0
```

Our Master node will now publish 0x21 every 100 milliseconds. We can now switch back to our Slave node session and see the requests from our Master node coming in:

```
~/Desktop/peak-lin-driver-1.2.0-rc6 linread /dev/plin1
count    timestamp d ID data
-----
1 5772559011 S 21 00 00 00 00
2 5772659010 S 21 00 00 00 00
3 5772759010 S 21 00 00 00 00
4 5772859011 S 21 00 00 00 00
5 5772959011 S 21 00 00 00 00
6 5773059010 S 21 00 00 00 00
7 5773159011 S 21 00 00 00 00
8 5773259011 S 21 00 00 00 00
9 5773359010 S 21 00 00 00 00
10 5773459010 S 21 00 00 00 00
11 5773559011 S 21 00 00 00 00
12 5773659011 S 21 00 00 00 00
13 5773759010 S 21 00 00 00 00
14 5773859010 S 21 00 00 00 00
15 5773959010 S 21 00 00 00 00
16 5774059010 S 21 00 00 00 00
17 5774159011 S 21 00 00 00 00
18 5774259011 S 21 00 00 00 00
19 5774359010 S 21 00 00 00 00
20 5774459010 S 21 00 00 00 00
21 5774559011 S 21 00 00 00 00
22 5774659010 S 21 00 00 00 00
23 5774759010 S 21 00 00 00 00
24 5774859011 S 21 00 00 00 00
25 5774959010 S 21 00 00 00 00
26 5775059010 S 21 00 00 00 00
27 5775159011 S 21 00 00 00 00
28 5775259011 S 21 00 00 00 00
29 5775359010 S 21 00 00 00 00
30 5775459011 S 21 00 00 00 00
31 5775559010 S 21 00 00 00 00
32 5775659010 S 21 00 00 00 00
33 5775759011 S 21 00 00 00 00
34 5775859010 S 21 00 00 00 00
35 5775959011 S 21 00 00 00 00
36 5776059010 S 21 00 00 00 00
37 5776159010 S 21 00 00 00 00
38 5776259010 S 21 00 00 00 00
39 5776359010 S 21 00 00 00 00
```

Finally let us subscribe our master to 0x22 using the scheduler:

First make sure you add 0x22 to the frame entries on your Slave node:

```
$ lin set pub-frm-entry 0x22 --cs-enhanced --len 4 /dev/plin0
```

Add the frame entry to your master node:

```
$ lin set sub-frm-entry 0x22 --cs-enhanced --auto-len /dev/plin0
```

Add the frame entry to a slot, in our example slot 0, a cycle time of 100 ms and the ID of the frame 0x22h:

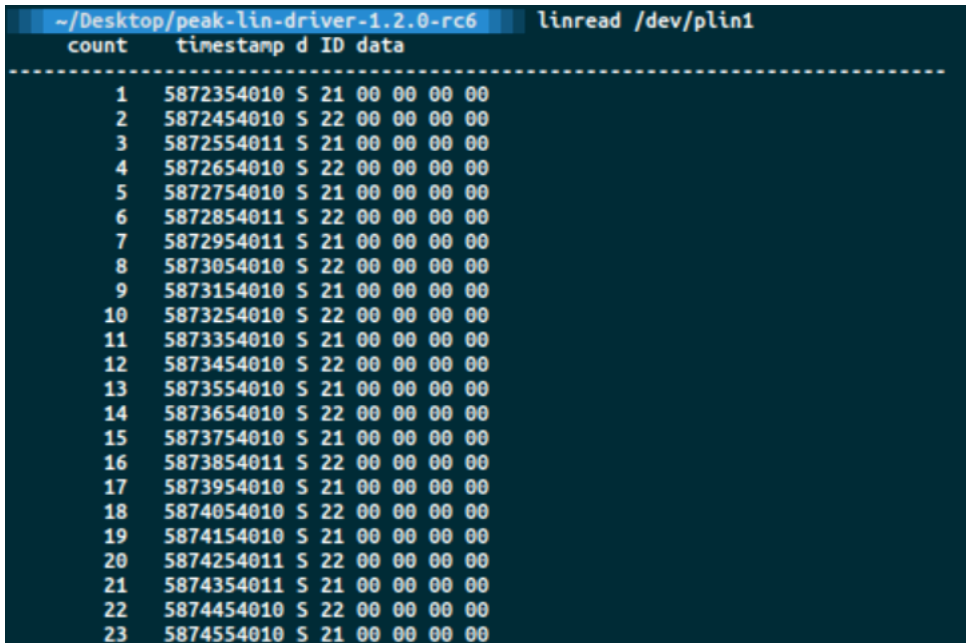
```
$ lin add unc-schd-slot 0 100 0x22 /dev/plin0
```

And start the schedule for this slot:

```
$ lin start schedule 0 /dev/plin0
```

Our Master node now requests 0x22 every 100 millisecond.

As you can see on our Slave node, both the published frame 0x21 as well as the requests for 0x22 can be read:



count	timestamp	d	ID	data
1	5872354010	S	21	00 00 00 00
2	5872454010	S	22	00 00 00 00
3	5872554011	S	21	00 00 00 00
4	5872654010	S	22	00 00 00 00
5	5872754010	S	21	00 00 00 00
6	5872854011	S	22	00 00 00 00
7	5872954011	S	21	00 00 00 00
8	5873054010	S	22	00 00 00 00
9	5873154010	S	21	00 00 00 00
10	5873254010	S	22	00 00 00 00
11	5873354010	S	21	00 00 00 00
12	5873454010	S	22	00 00 00 00
13	5873554010	S	21	00 00 00 00
14	5873654010	S	22	00 00 00 00
15	5873754010	S	21	00 00 00 00
16	5873854011	S	22	00 00 00 00
17	5873954010	S	21	00 00 00 00
18	5874054010	S	22	00 00 00 00
19	5874154010	S	21	00 00 00 00
20	5874254011	S	22	00 00 00 00
21	5874354011	S	21	00 00 00 00
22	5874454010	S	22	00 00 00 00
23	5874554010	S	21	00 00 00 00

Tip: If you want to find out which (physical) channel you are using, you can identify it like this:

```
$ lin identify /dev/plin0
```

The channel LED on the device will flash rapidly 2 times.

3.4 Uutils Folder

Here a short description of the utilities contained within the utils folder:

3.4.1 lin

The lin utility will act as the main configuration utility for your LIN devices.

Usage:

```
lin COMMAND [OPTIONS] [/dev/plinX]
```

Command:

lin set -s	set frames entries, ID filters, LEDs ...
lin get -g	get values set in the device
lin add -a	add a row in frames and scheduler tables
lin del -d	delete a row in frames and scheduler tables
lin start go	start master/slave mode, scheduler ...
lin disconnect stop	disconnect the LIN device
lin wake-up	wake-up the LIN bus (slave mode)
lin suspend pause	suspend scheduler, keep-alive ...
lin resume restart	resume scheduler, keep-alive ...
lin identify id	flash the LED of the given LIN device
lin reset rst	hard/soft reset of the device
lin help -h	display this help
lin version -v	display the version

3.4.2 linread

Will open up a readwindow on a specified device and display the incoming (filtered) LIN traffic.

Usage:

```
linread -h|-v
linread [OPTIONS] /dev/plinX
```

Command:

-e --eol=x	set EOL char value (0xa 0xd)
-f --fmt=FMT	set digits output format (x X d u)
-n --count=N	read N msgs then quit
-h --help	display help
-V --verbose[=x]	verbose mode (level x)
-v --version	display the version of the software

3.4.3 linwrite

Will write a frame to the LIN bus from a specified device.

Usage:

```
linwrite -h|-v
linwrite [OPTIONS] /dev/plinX
```

Command:

-b --bytes[="x y z ..."]	change default data bytes
-c --cstype[=U C E A]	change CS type: cUstom Classic Enhanced Auto
-d --dir[=D P S A]	change dir: Disabled, Publisher, Subscriber, Subscriber Auto len
-h --help	display help
-i --id[=x]	change default LIN frame ID

-l --len[=x]	change default LIN frame data length
-n --num[=s]	define the number of times the frame is sent to the driver
-p --pause[=s]	change default pause time after transmitting frame to driver
-v --version	display the version of the software

3.4.4 stop_lin.sh

Will close the connection on a specified device.

3.4.5 start_lin_master.sh & start_lin_slave.sh

These utilities can be used to test your devices. You can use `start_lin_master.sh` on a device and it will be completely set up a LIN master and will send the frame `id=0x1f` periodically. The corresponding `start_lin_slave.sh` started on a different device will set up a matching LIN slave. Both utilities will use `linread` to display the communication between both devices.

You'll see that closing the `start_lin_master` utility will result in a `SLEEP` notification on the active `start_lin_slave` utility. Closing the `start_lin_slave` with an active Master will result in a `Slave not responding error` message.

3.4.6 Additional Information

The kernel headers need to be installed for driver compilation!

Raspberry-PI:

- download the latest Raspbian image and install it on a SD card using Win32DiskImager or any other tool
- boot your RPI and setup all necessary settings (language, network etc.)
- now update your PI to the latest version and reboot once:
- `sudo apt get update`

- `sudo apt-get upgrade`
- `reboot`
- install the kernel headers:

```
$ sudo apt-get install raspberrypi-kernel-headers
```

Appendix A ident-string / ident-num Parameter

A channel can be identified by 2 user-defined identification fields: a 24-character string and a numerical value, initially empty:

For example, ident-string:

```
$ lin get ident-string /dev/plin0
```

```
$ lin set ident-string "wheel" /dev/plin0
$ lin get is /dev/plin0
wheel
```

Example using the user-defined numerical value, ident-num (32-bit coded):

```
$ lin get ident-num /dev/plin0
```

```
$ lin set ident-num 33 /dev/plin0
$ lin get in /dev/plin0
33
```

Please note that these LIN channel identification values can also be read/modified via the PLIN driver's sysfs interface. For example, the identification values read/modified above by the "lin get ident-xxx/lin set ident-xxx" commands have the following equivalent:

Ident-string:

```
$ cat /sys/class/plin/plin0/id_str
```

```
$ echo wheel | sudo tee /sys/class/plin/plin0/id_str
wheel
```

Using the user-defined numerical value, ident-num (32-bit coded):

```
$ cat /sys/class/plin/plin0/id_num
```

```
$ echo 33 | sudo tee /sys/class/plin/plin0/id_num
33
```

The point of identifying a channel is to be able to distinguish it regardless of the order in which USB LIN interfaces are connected. To this end, the driver installs the new Udev rule 45-plin.rules that will create a symbolic link to the LIN channel whose name will use (for example) the numerical identification value. In the example above, a symbolic link `"/dev/plin33"` will be created, pointing (in this case) to `/dev/plin0`, or whatever name is assigned by the system when the device is connected.

Note that any change in these identification values requires the driver to be unloaded and then reloaded so that Udev can take the new value into account:

```
$ sudo rmmod plin
$ sudo modprobe plin
$ ls -l /dev/plin*
crw-rw-rw- 1 root root 507, 0 May 27 16:18 /dev/plin0
crw-rw-rw- 1 root root 507, 1 May 27 16:18 /dev/plin1
lrwxrwxrwx 1 root root 5 May 27 16:18 /dev/plin33 -> plin0
```

From now on, accessing the `/dev/plin33` channel will be equivalent to accessing the `/dev/plin0` channel:

```
$ lin get is /dev/plin33
wheel
```

```
$ lin get in /dev/plin33
33
```