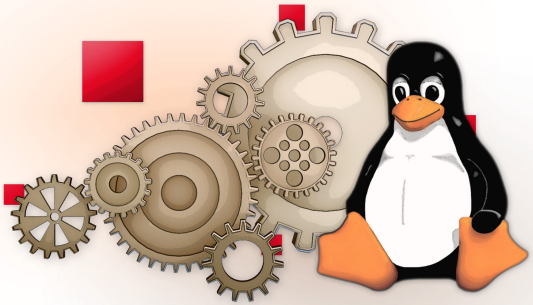
PCAN Driver for Linux

PCAN Driver for Linux

User Manual



PEAK System

Product names mentioned in this document may be the trademarks or registered trademarks of their respective companies. They are not explicitly marked by "™" and "®".

© 2010 PEAK-System Technik GmbH and Klaus Hitschler

PEAK-System Technik GmbH
Otto-Röhm-Straße 69
64293 Darmstadt
Deutschland

Phone: +49 (0)6151 8173-20
Fax: +49 (0)6151 8173-29

www.peak-system.com
linux@peak-system.com
klaus.hitschler@gmx.de

Issued 2010-04-19

Inhalt

1 Disclaimer 6

2 Changed Compilation Target since Kernel 2.6.25 7

3 Features of the 'pcan.o' or 'pcan.ko' Driver 8

3.1 Special Features of the “chardev driver” 9

3.2 Special Features of the “netdev driver” 9

4 Preluding Notes 11

5 Installing Manually 12

5.1 Unpacking the Files 12

5.2 Prerequisites for Compilation of the Sources 14

5.3 Manual Compilation and Installation 14

6 Loading the Driver 16

6.1 Driver Loading Specifics for chardev 17

6.2 Driver Loading Specifics for netdev 19

6.3 udev Support 23

6.4 pcan_make_devices 24

6.5 Specifics with the Driver Installation for PCAN-USB 24

6.6 Specifics with the Driver Installation for PCAN-PC Card 25

7 Customization of the modprobe Configuration File 26

7.1 Interface Hardware Type 27

7.2 I/O-Port and Interrupt Settings 27

7.3 Initial Bit Rate 27

7.4 Assign Parameter (netdev only) 28

8 Installation Test, Use of Test Programs (chardev only) 30

9 Most Important chardev Header Files 34

9.1 Manual Installation of the chardev Header Files 34

10 Most Important netdev Header Files 35

11 Compilation of the Driver Only 36

11.1 Use Cases 36

11.2 Support for Cross-Compilation 38

12 Features of the Shared Library 'libpcan.so' (chardev only) 39

13 Manual Installation of the chardev Shared Library 40

14 Message Filters (chardev only) 41

15 Error Handling 42

16 Socket-CAN Introduction and Basic Installation (netdev only) 43

17 Real-time Support with Xenomai 46

17.1 Installation 46

17.2 Compilation Environment 46

17.3 Runtime Environment 47

17.4 Troubleshooting 47

18 FAQ 49

Appendix A Usage of the Driver with chardev Mode 57

A.1 Polling 57

A.2 Blocking Wait 58
A.3 Blocking Wait with Timeout 58
A.4 Waiting for an IO-event with Select() 59

Appendix B Historical Parts 62
B.1 devfs 62
B.2 Compilation of Kernel Greater Than 2.5.x 62
B.3 Obfuscation 63
B.4 History of the Document 63

1 Disclaimer

Part of this program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License or LGPL Lesser General Public License as published by the Free Software Foundation version 2 of the License. This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301, USA

Attention! It is strictly prohibited to use the intellectual property from the provided source code for developing or producing a 'compatible' hardware. All rights are reserved by PEAK-System Technik GmbH

2 Changed Compilation Target since Kernel 2.6.25

Since Kernel version 2.6.25 the default compilation target has changed. Please take care if you have updated your target kernel.

The CAN Network subsystem PF_CAN (aka SocketCAN) became part of the mainline Linux Kernel since kernel version 2.6.25. For this reason the PEAK driver automatically compiles with the recommended netdev support, when CONFIG_CAN is set in the Kernel configuration.

The automatic configuration can furthermore be changed with NET=NETDEV_SUPPORT to force netdev configuration or NET=NO to force chardev configuration.

Attention! When using the netdev driver interface, the chardev driver test tools ("cat /dev/pcanXX", "receivetest", "transmittest") will not work correctly (receiving of messages won't be possible than).

To compile the PEAK Linux driver with the former (default) PEAK chardev interface, please invoke:

```
make clean; make NET=NO
```

3 Features of the 'pcan.o' or 'pcan.ko' Driver

The “pcan.o“ or “pcan.ko” driver is supporting PCAN-PCI, PCAN-miniPCI, PCAN-PCI Express, PCAN-ISA, PCAN-PC/104, PCAN-PC/104 plus, PCAN-Dongle, PCAN-PCCARD as well as PCAN-USB hardware. Depending on the type of hardware (PCI, ISA, Dongle, PC Card, and USB) 8 channels for each type can be supported. For PCAN-ISA, PCAN-PC/104 and PCAN-Dongle the base address and the used interrupt are settable. If no parameters are provided the driver uses factory defaults. A user program can communicate with the driver in three different ways:

- ASCII formatted data can be provided or accepted over a “read()“ or “write()“ interface. This data contains information from received or transmitted telegrams or about channel initializing parameters.
- With a “ioctl()“ interface. Through this interface user programs are able to receive and to transmit CAN-telegrams as well. However it is possible to get information about the channel status and to initialize the CAN-channel.
- As a network device with commonly used network socket calls. This kind of interface is alternative to the previous kinds. It is called the netdev drivers interface, in contrast to the chardev drivers interface. To use the netdev drivers interface with kernels before 2.6.25 you have to install the Socket-CAN support from socketcan.berlios.de.

Most of this manuals content covers the usage of the chardev driver, see chapter *Special Features of the “chardev driver”*. The netdev driver usage and installation is considered in depth in the chapter *Special Features of the “netdev driver”.* Parts of the following chapters are different for chardev or netdev implementations. Exceptions are marked to pinpoint the difference.

Since version 5.0 of the driver support for the XENOMAI real-time add-on to the Linux kernel is available. You can get more information in the chapter Real-time Support with Xenomai.

3.1 Special Features of the “chardev driver”

- The driver can be used from more than one user program at the same time and the simultaneous use of lots of interfaces and/or CAN channels is supported.
- The reading or writing of telegrams can block if the receive buffer doesn't contain data any more or if the write buffer is full. This behaviour can be disabled by opening the path as non-blocking.
- If more than one user program communicates with the same CAN-channel the input or output is shared. (e.g.: Many user programs are accidentally sharing the received telegrams)
- The “select“-method is supported.
- The interface to the driver (constants and structures) is defined in the header file `pcan.h`.

3.2 Special Features of the “netdev driver”

The netdev driver interfaces to the kernels network modules via a so called Socket-CAN framework. Socket-CAN aims to be a standard interface for access of CAN devices. This kind of interface enhances the features compared to the chardev driver. Especially it provides:

- Possibility for multiple paths reading/writing from/to the same CAN channel
- Built-in filter capabilities, e.g CAN-ID (range) filters with can-raw sockets

- Built-in broadcast manager for cyclic sending of CAN telegrams and content filtering in the CAN-data, e.g. updates to userspace only on data change
- Simple user API accessible from mostly all programming languages since normal network socket calls are used (creating PF_CAN analogue to PF_INET)
- Local loopback of sent CAN frames for network transparent applications
- Socket-CAN is accompanied from lots of useful tools and utilities (please have look at socketcan.berlios.de)

The interface to the driver (constants and structures) is defined in the header files `can.h` and `error.h` for kernels lower than 2.6.25.

4 Preluding Notes

Note: Before installing the software the hardware has to be installed in or at the computer (Exception: PCAN-Dongle, PCAN-PC Card or PCAN-USB [1]).

This manual is describing the installation process on an x86-Linux-computer with installed kernel 2.6.x. and a SuSE 10.1 system. Exceptions for kernel 2.4.x. and x86-RedHat systems are particularly mentioned.

Since driver version 6.0 support for kernels 2.2.x has ceased. If you need a driver for these kernels please have a look at the latest 5.x driver version.

During the installation sometimes a console and perhaps an editor as well will be used in the “root“- mode. If you want to login as “root“-user within a console or xterm-session you can use the 'su' command or the built in menu “file/root console“.

For Debian based systems (*ubuntu) you have to use the sudo command instead or replace “su” with “sudo su”.

To start an editor in the “root“–mode, just invoke from the menu “system/terminals/terminal (system manager mode)“ in KDE. From the console command line you might invoke the editor “kwrite“, “kate“ or “emacs“. For some distributions the standard path doesn't include the “/sbin“ folder. Then installation commands have to typed in with “/sbin/“ ahead.

Examples:

```
/sbin/modprobe ...    instead of modprobe ...
/sbin/insmod ...      instead of insmod ...
/sbin/rmmod ...       instead of rmmod ...
/sbin/modinfo ...     instead of modinfo ...
```

5 Installing Manually

5.1 Unpacking the Files

Unpack the file `peak-linux-driver.x.y.tar.gz` into an optional folder into your home directory. Example:

```
tar  -xzf peak-linux-driver.x.y.tar.gz
```

Within this directory you get following tree of files:

```
peak-linux-driver-x.y/
|-- Documentation     Any documentary
|   |-- COPYING       GNU Public License
|   |-- Installation-en.pdf   this installation instructions
|   |-- template.c
|   |-- template.h
|   |-- template.make
|   `-- todo.txt
|-- Makefile   A 'global' makefile
|-- driver     the driver and its sources
|   |-- Makefile      the makefile for the driver
|   |-- pcan.h Application interface definitions of the driver
|   |-- pcan_make_devices     a script for generating device nodes
|   |-- udev   udev supporting stuff
|   |   `-- 45-pcan.rules          udev rules to modprobe driver
 for hotplug devices
|   |-- src    the driver sources
|   |   |-- ...
|   |   |-- ...
|   |   |-- ...
|   |   `-- ...
|   |-- test.txt
|   `-- wstress
|-- lib the access library for (chardev)
|   |-- Makefile      the makefile of the library (chardev)
|   |-- libpcan.h     the library call prototypes (chardev)
|   `-- src    the sources of the library (chardev)
|       `-- libpcan.c
`-- test       the test programs
|-- Makefile   the makefile to the test programs
|-- src        The sources to the test programs
|   |-- bitratetest.c to test bitrate calculations (chardev)
|   |-- common.c
```

```
|   |-- common.h
|   |-- filtertest.cppto test filter settings (chardev)
|   |-- parser.cpp
|   |-- parser.h
|   |-- receivetest.c to receive test (chardev)
|   |-- transmitest.cpp       to transmit test (chardev)
|   |-- pcan-settings.c       a tool to set the PCAN-USB device
number
|   |-- receivetest_rt.c      to receive test, realtime variant
(chardev)
|   `-- transmitest_rt.cpp    to transmit test, realtime variant
 (chardev)
`-- transmit.txt      Test transmit instructions
```

5.2 Prerequisites for Compilation of the Sources

Before compiling following prerequisites have to be considered:

1. Up from kernel version 2.6.x the completely configured kernel sources for your target system have to be installed. If it is a standard installation the sources are located under “/usr/src/linux“ in which this normally is a link e.g. to “/usr/src/linux-2.6.6“. For every kernel smaller or equal than 2.4.x only the “kernel-headers“ have to be installed. (It's not wrong to install the complete sources of the target system.) The kernel-headers are installed if the invocation of “cat /lib/modules/'uname -r'/build/include/linux/modversions.h“ is ending without an error. (“'uname -r'“ is a placeholder for the result of the invocation “uname -r“.)
2. The “/lib/modules/'uname -r'/build/include/linux/version.h“ file has to exist or being copied before from the path “/boot/vmlinuz.version.h“. With kernels 2.6.x this file is created during the kernel compilation process.
3. Tools like make, gcc, etc. have to be installed. One of the test programms uses the g++ compiler. The test programms need installed “libpopt” and “libstdc++” with their header files (-dev variants).
4. Your target kernel must have the proc-filesystem enabled. Dependent of your CAN interface hardware PCI, USB and/or PCCARD support must be enabled with your target kernel configuration.

5.3 Manual Compilation and Installation

A semi-automatic installation with compilation can also be done with little effort. Unpack the tar.gz-files to any directory and start compilation and installation. Please note the prerequisites for compilation of the sources.

As default the driver and its utilities are compiled for netdev usage from kernel versions up from 2.6.25, otherwise for chardev usage with support for all hardware interfaces from PEAK. If you need to customize the driver or switch to netdev usage please look at the chapter *Compilation of the Driver Only*.

```
cd peak-linux-driver-x.y
make clean
make
su -c "make install"
```

Installation itself needs to be root.

After building the sources and the following binary installation on your target computer it's enough to type in

```
/sbin/modprobe pcan
```

to install the driver and any other useful modules. Depending on your requirements you have to do this procedure every time you start your computer or your application. (This may be done in a startup script, too.)

6 Loading the Driver

Note: From kernel version 2.6.0 the kernel module extension is “ko“.

If the driver to be loaded is applicable for PCAN-Dongle you may need to load the parport subsystem support. To do this please invoke as “root” user:

```
/sbin/modprobe parport
```

This command installs the “Parport Subsystem” with Linux. It is necessary to support the PCAN-Dongle integration [2], [3].

Login as “root” user in a console e.g. console or xterm. Fork to the “.../peak-linux-driver-x.y/driver” directory and type in:

```
/sbin/insmod pcan.o type=xxx,yyy,... io=0x300,0x378,... irq=10,7,...
```

or from kernel 2.6.0 on

```
/sbin/insmod pcan.ko type=xxx,yyy,... io=0x300,0x378,... irq=10,7,...
```

if you need USB support you have to do the same steps as in *Specifics with the Driver Installation for PCAN-USB*. The place holder “xxx,yyy,...“ are standing for “isa“, “sp“ or “epp“. The parameter for “io“ and “irq“ can be left out, if the standard assignments are configured with your hardware. PCI-channels, USB-channels and PCCARD based channels are found and installed without any special details but ISA (PC/104) and parallel-port supported channels have to be registered.

The standard assignments for ISA and PC/104 interfaces are (io/irq): 0x300/10, 0x320/5. The standard assignments for Dongle in SP/EPP mode are (io/irq): 0x378/7, 0x278/5.

6.1 Driver Loading Specifics for chardev

With the following command you can see any error report after “insmod-ing” (as “root“ user):

```
tail  /var/log/messages
```

Normally the output looks similar to this (depending on your hardware interfaces and your configuration settings):

```
Mar  6 17:35:26 xxxx kernel: pcan: Release_20070221_n
Mar  6 17:35:26 xxxx kernel: pcan: driver config [mod] [isa] [pci] [dng] [usb]
   [pcc]
Mar  6 17:35:26 xxxx kernel: pcan: pci device minor 0 found
Mar  6 17:35:26 xxxx kernel: pcan: pci device minor 1 found
Mar  6 17:35:26 xxxx kernel: pcan: usb hardware revision = 28
Mar  6 17:35:28 xxxx kernel: pcan: usb device minor 32 found
Mar  6 17:35:28 xxxx kernel: pcan: PEAK PC_CAN_CARD 001 A1
Mar  6 17:35:28 xxxx kernel: pcan: pccard device minor 40 found
Mar  6 17:35:28 xxxx kernel: pcan: pccard device minor 41 found
Mar  6 17:35:28 xxxx kernel: pcan: pccard firmware 1.5
Mar  6 17:35:28 xxxx kernel: pcan: major 253.
```

To check the correct installations of the driver please invoke:

```
cat  /proc/pcan
```

You'll get a similar printout if there are no errors:

```
*------------ PEAK-Systems CAN interfaces (www.peak-system.com) -------------
*-------------------------  Release_20070221_n  --------------------------
*------------------ [mod] [isa] [pci] [dng] [usb] [pcc] --------------------
*---------------------- 8 interfaces @ major 253 found -----------------------
*n -type- ndev --base-- irq --btr- --read-- --write- --irqs-- -errors- status
 0    pci -NA- ec00b000 010 0x001c 00000000 00000000 00000000 00000000 0x0000
 8    isa -NA- 00000300 010 0x001c 00000000 00000000 00000000 00000000 0x0000
 9    isa -NA- 00000320 010 0x001c 00000000 00000000 00000000 00000000 0x0000
16     sp -NA- 00000378 007 0x001c 00000000 00000000 00000000 00000000 0x0000
32    usb -NA- ffffffff 255 0x001c 00000000 00000000 00000000 00000000 0x0000
33    usb -NA- 20003412 033 0x001c 00000000 00000000 00000000 00000000 0x0000
40 pccard -NA- 00000400 005 0x001c 00000000 00000000 00000000 00000000 0x0000
```

```
41 pccard -NA- 00000420 005 0x001c 00000000 00000000 00000000 00000000 0x0000
```

As default the driver is configured for the dynamic allocation of the "major" number [4].

Note: The contents of the ndev column. It differentiates between chardev and netdev drivers. netdev drivers show the network interface, chardev drivers do not have a network interface (-NA-).

With a properly configured and running UDEV system all devices files are generated on the fly. If your target system does not have a running UDEV system you must create the device files manually each time after driver installation. Please invoke the shell script for this:

```
./pcan_make_devices 2
```

This script invocation installs two device files for each type of CAN adapter. Please verify the result with the command:

```
ls  -l  /dev/pcan*
```

Note: When using the real-time flavor of the driver no device file is created. For a netdev device the device file is created but currently it is only necessary to set the bit rate for the interface.

Now you already can do your first test with your CAN hardware. Just connect a CAN-transmitter to your CAN interface and send telegrams with bit rate setting of 500 kbit/sec. Login on an user-console and invoke:

- cat /dev/pcan0 for the first PCAN-PCI interface
- cat /dev/pcan8 for the first PCAN-ISA or PCAN-PC/104 interface
- cat /dev/pcan16 for the first PCAN-Dongle interface in SP-mode
- cat /dev/pcan24 for the first PCAN-Dongle interface in EPP-mode
- cat /dev/pcan32 for the first PCAN-USB interface [5]
- cat /dev/pcan40 for the first PCAN-PC Card interface

The received telegrams should be shown [6]. This invocation can be made at more than one console for more than one interface at the same time.

Output of data is already possible. With the call of:

```
echo „m s 0x234 2 0x11 0x22“ > /dev/pcan0
```

for example a CAN telegram with standard frame is transmitted with the identifier 0x234 and two data bytes “0x11“ and “0x22“ from the first PCAN-PCI interface. The bit rate of the interface can as well be set from the so called “write interface“.

6.2 Driver Loading Specifics for netdev

With the following command you can see any error report after “insmoding” (as “root“ user):

```
tail  /var/log/messages
```

Normally the output looks similar to this (depending on your hardware interfaces and your configuration settings):

```
Mar  6 17:39:51 xxxx kernel: pcan: Release_20070221_n
```

```
Mar  6 17:39:51 xxxx kernel: pcan: driver config [mod] [isa] [pci] [dng] [usb]
   [pcc] [net]
Mar  6 17:39:51 xxxx kernel: pcan: pci device minor 0 found
Mar  6 17:39:51 xxxx kernel: pcan: pci device minor 1 found
Mar  6 17:39:51 xxxx kernel: pcan: usb hardware revision = 28
Mar  6 17:39:53 xxxx kernel: pcan: registered netdevice can0 for pcan usb hw
   (minor 32)
Mar  6 17:39:53 xxxx kernel: pcan: usb device minor 32 found
Mar  6 17:39:53 xxxx kernel: usbcore: registered new driver pcan
Mar  6 17:39:53 xxxx kernel: pcan: PEAK PC_CAN_CARD 001 A1
Mar  6 17:39:53 xxxx kernel: pcan: pccard device minor 40 found
Mar  6 17:39:53 xxxx kernel: pcan: registered netdevice can1 for pcan pccard hw
   (minor 40)
Mar  6 17:39:53 xxxx kernel: pcan: pccard device minor 41 found
Mar  6 17:39:53 xxxx kernel: pcan: registered netdevice can2 for pcan pccard hw
   (minor 41)
Mar  6 17:39:53 xxxx kernel: pcan: pccard firmware 1.5
Mar  6 17:39:53 xxxx kernel: pcan: registered netdevice can3 for pcan pci hw
   (minor 0)
Mar  6 17:39:53 xxxx kernel: pcan: registered netdevice can4 for pcan pci hw
   (minor 1)
Mar  6 17:39:53 xxxx kernel: pcan: major 253.
```

After insmod you have to bring your network devices up, for example:

```
ifconfig can0 up
```

Depending on the configuration of your system (hotplugging, etc.) you might get a log entry like:

```
Mar  6 17:39:53 xxxx ifup:     can0
Mar  6 17:39:53 xxxx ifup:              No configuration found for can0
```

as the CAN network interface is currently not recognized and handled by these 'hotplugging' scripts. It's save to ignore these kind of log messages.

Before 'modprobing' the netdev prepared modules you should edit your /etc/modprobe.conf (or /etc/modprobe.conf.local or

/etc/modules.conf, depending on your kernel version and distribution) and add at least these lines:

```
# protocol family PF_CAN
   alias net-pf-29 can
# protocols in PF_CAN
   alias can-proto-1 can-raw
   alias can-proto-2 can-bcm
# protocol module options
# option can-tpgen printstats=1
# option can       stats_timer=0
# virtual CAN devices
   alias vcan0 vcan
   alias vcan1 vcan
   alias vcan2 vcan
   alias vcan3 vcan
# CAN hardware (uncomment the currently used)
#> Peak System hardware (ISA/PCI/Parallelport Dongle, USB, PC Card)
#> to set initial BTR-values set and hardware dependent settings
   alias can0 pcan
   alias can1 pcan
   alias can2 pcan
   alias can3 pcan
#options pcan assign=peak
#options pcan type=isa,isa io=0x2C0,0x320 irq=10,5 btr=0x4914
#options pcan type=epp btr=0x4914
#options parport_pc io=0x378 irq=7
```

Depending on the desired target hardware interfaces and transport protocols you may need more entries. Please look into the Socket-CAN documentation for more information.

With the special 'assign=peak' module load parameter you can assign the network device names of the CAN channels corresponding to the assigned minor numbers, e.g. network device name “can40” corresponds devices minor number 40.

If you omit this parameter the network device names are automatically assigned in channel initialization order.

A third option allows assigning individual network device names. To accomplish it please add a string similar to this:

```
assign=pcan32:can1,pcan41:can2
```

Note: With kernels 2.4.x you should use a plus (+) instead a comma (,) to separate the entries.

To check the correct installation of the driver please invoke:

```
cat  /proc/pcan
```

You'll get a similar printout if there are no errors:

```
*------------ PEAK-Systems CAN interfaces (www.peak-system.com) -------------
*-------------------------  Release_20070221_n  --------------------------
*---------------- [mod] [isa] [pci] [dng] [usb] [pcc] [net] -----------------
*--------------------- 6 interfaces @ major 253 found -----------------------
*n -type- ndev --base-- irq --btr- --read-- --write- --irqs-- -errors- status
 0    pci can2 f7d00000 209 0x001c 00000000 00000000 00000000 00000000 0x0000
 1    pci can3 f7d00400 209 0x001c 00000000 00000000 00000000 00000000 0x0000
32    usb can0 00000000 033 0x001c 00000000 00000000 000000a6 00000000 0x0000
33    usb can1 00000000 255 0x001c 00000000 00000000 00000000 00000000 0x0000
40 pccard can4 00000180 169 0x001c 00000000 00000000 00000000 00000000 0x0000
41 pccard can5 000001a0 169 0x001c 00000000 00000000 00000000 00000000 0x0000
```

The 'ndev' column shows the network device assignment. For assignment of bitrates to network devices you can either set the desired bit rate as module load parameter or you can assign the bit rate after loading the driver with a simple command line invocation like:

```
echo "i 0x4914 e" > /dev/pcan0
```

For this reason it is important to create the chardev-device-entries, too. If your target does no create the device files automatically with UDEV you have to run with the script

```
./pcan_make_devices 2
```

even if only the netdev driver is used.

To test the correct installation of the netdev driver you can use the Socket-CAN utility “candump”.

```
$ candump
Usage: candump [can-interfaces]
  (use CTRL-C to terminate candump)
Options: -m <mask>   (default 0x00000000)
         -v <value>  (default 0x00000000)
         -i <0|1>    (inv_filter)
         -e <emask>  (mask for error frames)
         -t <type>   (timestamp: Absolute/Delta/Zero)
         -c          (color mode)
         -a          (enable additional ASCII output)
         -s <level>  (silent mode - 1: animation 2: nothing)
         -b <can>    (bridge mode - send received frames to <can>)
         -l          (log CAN-frames into file)
         -L          (use log file format on stdout)
```

To receive from any CAN netdevice you might also invoke:

```
candump any
```

instead of working with specific CAN netdevices:

```
candump can0 can2
```

6.3 udev Support

From the driver version 6.8 udev support is provided. When the driver becomes installed the udev rules '45-pcan.rules' are copied into the directory “/etc/udev/rules.d”. After that each time PCAN device is recognized the driver is loaded and the device files are created automatically.

The script to install device files for non-UDEV target 'pcan_make_devices' is still provided. This script should not used together with udev.

Note: The device nodes are removed also automatically when a hotplug-device is removed. There is a compatibility issue with updates of a modprobe configuration file and 'pcan_make_devices' and the udev installation. Please look at 'Customization of 'modules.conf' or 'modprobe.conf'' for further details.

6.4 pcan_make_devices

Note: That pcan_make_devices is obsolete with systems supporting udev

Help invocation of 'pcan_make_devices'

```
$pcan_make_devices -help
Usage: pcan_make_devices n [--help | -h]
Info: Create n device nodes for each PCAN-type.
...
```

You must be root to invoke pcan_make_devices. To use 'pcan_make_devices' a successful insmod/modprobe pcan.o or pcan.ko must precede.

No device files must be created for Xenomai devices.

6.5 Specifics with the Driver Installation for PCAN-USB

In order to use the driver the USB subsystem of the Linux kernel has to be enabled. USB support for Linux was first introduced with kernel 2.4. The use of PCAN-USB together with a Linux kernel version less than 2.4.7 is deprecated through the driver developer.

6.6 Specifics with the Driver Installation for PCAN-PC Card

The user side of the PCMCIA subsystem of the kernel has undergone heavy changes through subsequent releases of kernel 2.6.x. So we have to distinguish between the implementations:

Up from kernel 2.6.13 the vendor and card IDs are not any more registered in the configuration file `/etc/pcmcia/config` since the relationship between manufacturer-ID and device-ID of the card and the associated driver is registered inside the driver. The driver should be loaded before card plug in.

With these kernel versions you can use the pccardctl utility from the pcmcia-package, especially “pccardctl eject” and “pccardctl insert” to prepare a card for ejection or to notify an inserted card. The former utility to do the same was called cardctl.

For earlier kernel versions than 2.6.13 you need entries in “/etc/pcmcia/config” like:

```
device “pcan”
class “none” module “pcan”
card “PCAN-PCCARD”
manfid 0x0377, 0x0001
bind “pcan”
```

With these entries you can use your configured hotplug system to load the driver when the card is plugged in.

Also for kernels lower than 2.6.13 you can use the cardctl utility from the pcmcia-package, especially “cardctl eject” and “cardctl insert” to prepare a card for ejection or to notify an inserted card.

Note: Removing a card with still open channels is not recommended.

7 Customization of the modprobe Configuration File

Note: modprobe installation options are put into a modprobe configuration file named either `/etc/modprobe.conf` or `/etc/modprobe.conf.local` or `/etc/modules.conf` or currently `/etc/modprobe.d/pcan`. The name depends on kernel version and distribution.

Former modprobe installation options configured an invocation of 'pcan_make_devices' in the modprobe configuration file. These entries will not be modified by a fresher installation to avoid overwriting user made entries. Please remove a potential invocation of 'pcan_make_devices' inside this configuration files if your system supports udev.

The current entry locates in '/etc/modprobe.d/pcan':

```
# pcan - automatic made entry, begin --------
# if required add options and remove comment
# options pcan type=isa,sp
install pcan /sbin/modprobe --ignore-install pcan
# pcan - automatic made entry, end ----------
```

At the end of the modprobe configuration file a marked addition is added through the installation. The respective type and the eventually non-standard base addresses and interrupt numbers are to be given there, if the driver has to support the PCAN-ISA or PCAN-Dongle interfaces. (Have a look at the description about the manual loading of the driver)

Note: As soon as there already is an entry with the word 'pcan' the current entry is not to be changed. If you wish to have a new entry you have to delete the old one first. To do that you have to use an editor as “root“ user.

7.1 Interface Hardware Type

To register non-hot pluggable devices permanently you have to use these settings:

```
type=type1,type2[,...]
```

where type* is “isa”, or “sp”, or “epp” depending on your interface hardware. Example:

```
type=isa,sp
```

7.2 I/O-Port and Interrupt Settings

If you like to use non-standard settings for IO-ports or interrupt number assignments you have to add:

```
io=io1,io2[,...] irq=irq1,irq2[,...]
```

where io* is the hexadecimal number of the corresponding IO-port and irq* is the decimal number of the assigned interrupt line.

Examples:

```
io=0x300,0x378 irq=10,7
```

The standard assignments for ISA and PC/104 interfaces are (io/irq): 0x300/10, 0x320/5.

The standard assignments for Dongle in SP/EPP mode are (io/irq): 0x378/7, 0x278/5.

7.3 Initial Bit Rate

To prevent a wrong initial bit rate when loading the driver we added a driver load parameter which sets the initial bit rate of the related interfaces. For example to set the initial bit rate to 1 Mbit/sec the syntax is:

```
bitrate=0x0014
```

In order to make this setting persistent you have to modify /etc/modules.conf or /etc/modprobe.conf (depending on your kernel version).

Note: Please consider that all interfaces handled with this driver get the same initial bit rate.

The corresponding (common) bit rate codes are:

```
#define CAN_BAUD_1M     0x0014  //   1 Mbit/s
#define CAN_BAUD_500K   0x001C  // 500 kBit/s
#define CAN_BAUD_250K   0x011C  // 250 kBit/s
#define CAN_BAUD_125K   0x031C  // 125 kBit/s
#define CAN_BAUD_100K   0x432F  // 100 kBit/s
#define CAN_BAUD_50K    0x472F  //  50 kBit/s
#define CAN_BAUD_20K    0x532F  //  20 kBit/s
#define CAN_BAUD_10K    0x672F  //  10 kBit/s
#define CAN_BAUD_5K     0x7F7F  //   5 kBit/s
```

7.4 Assign Parameter (netdev only)

There are three choices to assign a network device name to CAN interfaces:

```
assign=peak
assign=device-node-name:network-device-name[, ... ]
```

With the special 'assign=peak' module load parameter you can assign the network device names of the CAN channels corresponding to the assigned minor numbers, e.g.

Network device name “can40” corresponds devices minor number 40.

A third option allows assigning individual network device names. To accomplish it please add a string similar to this:

```
assign=pcan32:can1,pcan41:can2
```

Note: With kernels 2.4.x you should use a plus (+) instead a comma (,) to separate the entries.

If you omit this parameter the network device names are automatically assigned in channel initialization order.

8 Installation Test, Use of Test Programs (chardev only)

Two test programs are attached for the initial test of the interfaces. For using the programs please change to the directory ".../peak-linux-driver-x.y/test". Please invoke:

```
./receivetest --help
```

You get following response:

```
receivetest Version "Release_20040412_a"  (www.peak-system.com)
--------- Copyright (C) 2004 PEAK System-Technik GmbH ---------
receivetest comes with ABSOLUTELY NO WARRANTY.     This is free
software  and you are welcome  to redistribute it under certain
conditions.   For   details   see    attached   COPYING   file.
Receivetest - a small test program which receives and prints CAN messages.
Usage: receivetest{[-f=devicenode]|{[-t=type] [-p=port [-i=irq]]}}[-
   b=BTR0BTR1][-e]
options: -f - devicenode - path to device file, default=/dev/pcan0
-t - type of interface, e.g. 'pci', 'sp', 'epp', 'isa' or 'usb' (default: pci).
-p - port in hex notation if applicable, e.g. 0X378 (default: 1st port of type).
-i - irq in dec notation if applicable, e.g. 7 (default: irq of 1st port).
-b - BTR0BTR1 code in hex, e.g. 0X001C (default: 500 kbit).
-e - accept extended frames. (default: standard frames)
-? or --help - this help
receivetest: finished (0)
```

The default bit rate settings for all CAN-interfaces are set to 500 kbit/sec. The bit rate can be customized through the command line parameters.

For using an ISA or Dongle interface with the standard settings following detail is enough:

```
receivetest -t=isa  or  receivetest -t=sp or receivetest  -t=epp
```

or

```
receivetest  -f=/dev/pcan0
```

to get to the first PCAN-PCI-interface.

Every received telegram of the interface will be print out. Entering Ctrl-C cancels the program.

The program “transmitest“ can be used to transmit data. Please invoke:

```
./transmitest -help
```

You get following output similar to receivetest:

```
transmitest Version “Release_20040412_a”  (www.peak-system.com)
--------- Copyright (C) 2004 PEAK System-Technik GmbH ---------
transmitest comes with ABSOLUTELY NO WARRANTY.     This is free
software  and you are welcome  to redistribute it under certain
conditions.   For   details   see    attached   COPYING   file.
Transmitest – a small test program which sends CAN messages.
Usage: transmitest filename {[-f=devicenode]|{[-t=type][-p=port[-i=irq]]}} ...
options: filename – mandatory name of message description file.
-f – devicenode – path to device file, default=/dev/pcan0
-t – type of interface, e.g. ‘pci’, ‘sp’, ‘epp’ ,’isa’ or ‘usb’ (default: pci).
-p – port in hex notation if applicable, e.g. 0X378 (default: 1st port of type).
-i – irq in dec notation if applicable, e.g. 7 (default: irq of 1st port).
-b – BTR0BTR1 code in hex, e.g. 0X001C (default: 500 kbit).
-e – accept extended frames. (default: standard frames)
-? or --help – this help
transmitest: finished (0)
```

This program expects similar parameters as receive test with the exception of a file name. In the file with this name the program expects a list of telegram descriptions which are to be send. Those telegrams will be send cyclic until the ending by pressing Ctrl-C. The syntax of the description of the telegrams is:

Descriptions of the first column of the telegram of the read/write interface.

Tag / Column #1	Description
m	Message string follows
r	RTR-message string follows
i	Initalisation string follows (write only)
#	Comment follows (write only)

Descriptions of the following columns of 'normal' and RTR-telegrams.

Column	Description
2	s = Standard frame, e = Extended frame
3	Identifier (hex)
4	ILenght of telegram (dec)
5..12	Telegram bytes (hex)
13	Timestamp milli seconds (dec, read only)
14	Timestamp milli seconds (dec, read only)
13	Comment (write only)

Descriptions of the following columns for initializing strings.

Column	Description
2	BTR0BTR1 Init data (hex, write only)
3	E = allow extended frames (write only)

For example:

```
# a comment
# a message, standard frame, id=0x123, 0 Data bytes
m s 0x123 0
# a message, standard frame, id=0x123, 1 Data byte, Data
m s 0x123 1 0x11
# a message, standard frame, id=0x123, 1 Data byte, Data
m s 0x123 2 0x11 0x22
# a message, extended frame, id=0x123, 3 Data byte, Data
```

```
m e 0x123 3 0x11 0x22 0x33
# a RTR, standard frame, id=0x123, 0 Data byte, comment
r s 0x123 0   # a comment
# a RTR, extended frame, id=0x123, 0 Data byte, comment
r e 0x123 0   # a comment
# initialize
i 0x1234 e
```

9 Most Important chardev Header Files

Any sources of the driver, of the library and the test programs are fit in underneath the “peak-linux-driver-x.y“ directory tree. Two files are to be mentioned especially: The file `.../peak-linux-driver/driver/pcan.h` describes the interface to the driver and command constants. You have to include this file if you want to have direct access to the driver with “open(), ioctl(), read(), close() etc.“. The file `.../peak-linux-driver/lib/libpcan.h` describes the interface to the dynamic linkable library libpcan.so. You have to include this file if your application needs to use the dynamic library.

Note: libpcan.h is including the pcan.h file

9.1 Manual Installation of the chardev Header Files

Please copy the header files into the directory “/usr/include“ as “root” user as followed and customize the access rights.

```
cp peak-linux-driver/driver/pcan.h  /usr/include/pcan.h
chmod 644 /usr/include/pcan.h
cp peak-linux-driver/lib/libpcan.h  /usr/include/libpcan.h
chmod 644 /usr/include/libpcan.h
```

10 Most Important netdev Header Files

For kernel versions lower than 2.6.25 the header file `can.h` and other includes important for application programs should be taken from the Socket-CAN source. Appropriate Makefiles can be found, e.g, at `socketcan/trunk/can-utils`.

11 Compilation of the Driver Only

11.1 Use Cases

In some cases of using the installed driver it doesn't fit to the configured operating system. This could be following cases:

1. Your kernel sources are not to be found in the standard directory `/usr/src/linux`. You still can compile with following invocations:

```
cd ~/peak-linux-driver-x.y/driver
make clean
make KERNEL_LOCATION=<path-to-my-kernel>
```

2. You don't need or don't want support for some interface types. Then you have to compile the driver with disabled support for some interfaces. The makefile provides to enable or disable partial compilation with this switches:

```
USB=USB_SUPPORT    or USB=NO
PCI=PCI_SUPPORT    or PCI=NO
DNG=DONGLE_SUPPORT or DNG=NO
ISA=ISA_SUPPORT    or ISA=NO
PCC=PCCARD_SUPPORT or PCC=NO
NET=NETDEV_SUPPORT or NET=NO
```

   As default all interfaces are enabled. For example if you want to suppress PCAN-Dongle and PCAN-ISA Interfaces you have to compile:

```
cd ~/peak-linux-driver-x.y/driver
make clean
make DNG=NO ISA=NO
```

   As default for kernels lower than 2.6.25 the netdev driver is disabled. If you like to use the netdev driver please compile with

```
NET=NETDEV_SUPPORT.
```

3. The following section only applies to kernel versions 2.4.x or lower: The driver is to be installed on RedHat Linux. RedHat in contrast to SuSE as default uses the "CONFIG_MODVERSIONS" modifier to prohibit the installation of improper drivers. If your operating system was generated with enabled "CONFIG_MODVERSIONS" you can see this easily by invoking "cat /proc/ksyms". If there is cryptic number-letter combinations at the end of the symbols your system was generated with enabled "CONFIG_MODVERSIONS".

   A simple new compilation on your system is enough:

```
cd ~/peak-linux-driver-x.y/driver
make clean
make
```

4. Your operating system doesn't support the PARPORT_SUBSYSTEM or you have a parallel interface which is unknown to the operating system and you like to use a PCAN-Dongle with this interface. Here you have to compile with the "PAR=NO" switch:

```
cd  ~/peak-linux-driver-x.y/driver
make clean
make PAR=NO_PARPORT_SUBSYSTEM
```

5. You get runtime errors and want to see more information for diagnosis at "/var/log/messages". Here you have to translate with the "DBG=DEBUG" switch:

```
cd  ~/peak-linux-driver-x.y/driver
make clean
make DBG=DEBUG
```

Combinations of these switches are also possible.

11.2 Support for Cross-Compilation

The driver, its library and the test programs sometimes are used on other systems than the development system. The process of generating the binary driver for another target is called Cross-Compilation. Partially the driver and its helper programs are supporting Cross-Compilation. Therefore the driver has to be compiled with the option.

```
make KERNEL_LOCATION=<path-to-my-kernel>
```

Whether the driver can be compiled for every possible target system and is working there can't be guaranteed because of so many different architectures. The installation of the driver and the library an on other target than the host system can't be supported through the given installation scripts. We are glad about hints how to improve the Cross-Compilation support. Especially we are looking for successful ports of the driver to document the knowledge for other engineers.

12 Features of the Shared Library 'libpcan.so' (chardev only)

The “libpcan.so“ library is providing an easier interface for using the features of the PCAN drivers. The library interface is similar to the one from MS-Windows and makes the porting of applications easier. There are also some calls which adapt better to the specifics of Linux. They are marked with a 'LINUX_..' as prefix. (Example: HANDLE LINUX_CAN_Open(char *szDeviceName, int nFlags);)

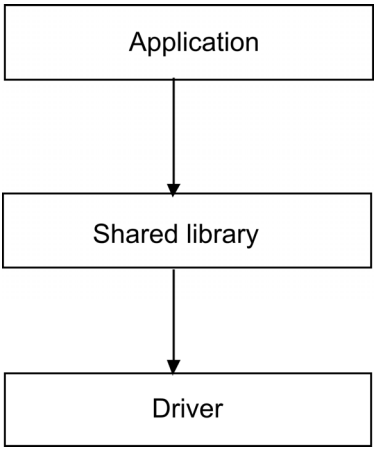


Figure 1: Calling hierarchy

The interface of the library is described within the C/C++-header file `libpcan.h`. Please note that `libpcan.h` includes the `pcan.h` file. The source code is under LGPL so it is for utilizing within proprietary software without any exceptions.

13 Manual Installation of the chardev Shared Library

Please change to the directory "../peak-linux-driver-x.y/lib". Login as "root" user and invoke:

```
cp libpcan.so.0.1 /usr/lib/libpcan.so.0.1
ln -sf /usr/lib/libpcan.so.0.1 /usr/lib/libpcan.so.0
ln -sf /usr/lib/libpcan.so.0 /usr/lib/libpcan.so
```

This process is only necessary if you install this program for the first time or if you want to update the library. To use the test programs the library has to be installed.

Note: The version minor number of the library can change as a result of minor or major improvements.

14 Message Filters (chardev only)

Up from driver version 6.0 message filters are supported for the chardev device. With the netdev device message filters are part of Socket-CAN.

To maintain backward compatibility, after first open of a path to a device no message filter is set. With the first time a message filter is set or reset the message filter chain will be activated.

It is possible to set more than one filter condition in a message chain for each path. After a message is received the filter chain is walked trough to check for a pass condition. If no pass condition is found the message is rejected.

15 Error Handling

The CAN error handling for the chardev driver has changed from version 5.x to version 6.x of the driver. Now the handling is more compatible to the MS-DOS implementation.

If a CAN error occurs a status message is enqueued into the channels receive queue. A status message is marked with MSGTYPE_STATUS set in the MSGTYPE field of the TPCANMsg structure.

Depending on the kind of error detailed CAN_ERR_... information is set in the DATA[3] field of the same structure.

Error handling with netdev devices is similar, however the error information is fit into the can_frame structure. For details please look into the Socket-CAN documentation.

16 Socket-CAN Introduction and Basic Installation (netdev only)

The Socket-CAN package is an implementation of CAN protocols (Controller Area Network) for Linux. CAN is a networking technology which has wide-spread use in automation, embedded devices, and automotive fields. While there have been other CAN implementations for Linux based on character devices, Socket-CAN uses the Berkeley socket API, the Linux network stack and implements the CAN device drivers as network interfaces. The CAN socket API has been designed as similar as possible to the TCP/IP protocols to allow programmers, familiar with network programming, to easily learn how to use CAN sockets. (Taken from the README file from the Socket-CAN package).

The development on Socket-CAN started end of 2002 at Volkswagen AG to have a common CAN-IT-interface for HMI- and Car2Car prototypes. Nowadays Socket-CAN is maintained by a group of developers at socketcan.berlios.de. One of the Socket-CAN maintainers, Oliver Hartkopp, did most of the netdev integration into the plain PCAN driver.

Note: Socket-CAN modules are part of the mainline kernel since version 2.6.25. So the generation of the socket-CAN modules do not apply to newer kernels. For newer kernels you appropriately have to configure your kernel (CONFIG_CAN, CONFIG_CAN_RAW, CONFIG_CAN_BCM, CONFIG_CAN_VCAN) to use the necessary modules.

To use Socket-CAN you first have to fetch the Socket-CAN sources and compile and load them. To fetch the sources you might invoke:

svn co http://svn.berlios.de/svnroot/repos/socketcan/trunk /where/it/should/load

Then please follow the instructions given in the README file to compile and load Socket-CAN. With Socket-CAN you get a driver for “virtual CAN channels”. They are called vcan0, vcan1 and so on.

To get the Socket-CAN loaded you can invoke as root-user:

```
modprobe can
modprobe can-raw
modprobe can-bcm
modprobe vcan
```

to load the necessary modules. You might look into your kernel-log (e.g. /var/log/messages) to see the module banners printed at load time.

After loading you’ll get this lsmod output

```
lsmod | grep can
can_bcm                16392  0
can_raw                 9856  0
can                    36072  2 can_bcm,can_raw
vcan                    4360  0
```

Now you only have to bring up your new network nodes. For example please invoke:

```
ifconfig vcan0 up
```

This starts the network device vcan0. To see if the network device is installed please call:

```
ifconfig
vcan0 Protokoll:UNSPEC  Hardware Adresse 00-00-00-00-00-00-00-00-00-00-00-00-
   ...RX packets:0 errors:0 dropped:0 overruns:0 frame:0 TX packets:0 errors:0
   dropped:0 ...carrier:0 collisions:0 Sendewarteschlangenlänge:1000 RX bytes:0
   (0.0 b)  TX bytes:0 (0.0 b)
eth0      ...
lo        ...
```

With this basic installation you can test your Socket-CAN setup. To test it you must have the Socket-CAN can-utilities compiled. In a first console invoke:

```
./cangen vcan0
```

If you invoke in a second console “candump vcan0” then you’ll get the generated (and loop backed) CAN messages from your first invocation:

```
./candump vcan0
vcan0  567   [6] 69 98 3C 64 73 48
vcan0  451   [8] 4A 94 E8 2A EC 58 55 62
vcan0  729   [8] BA 58 1B 3D AB D7 7E 50
vcan0  1F2   [8] E3 A9 E2 79 46 E1 45 75
```

This output shows that your basic Socket-CAN installation was successful. Next you have to install the PCAN netdev driver.

Note: Currently most of the Socket-CAN utilities can only be used as root-user. Due to the standard NET-CAPABILITIES the access to the RAW-sockets and to the BCM-sockets are only granted to the user root (analogue to the promiscuous mode on ethernet devices). But you may define CONFIG_CAN_RAW_USER and/or CONFIG_CAN_BCM_USER in the Kconfig (when integrated into the Kernel) to override this.

You may also try to put

```
#define CONFIG_CAN_RAW_USER
#define CONFIG_CAN_BCM_USER
```

into your local /usr/src/linux/include/linux/autoconf.h before compiling the Socket-CAN source code or change the files `raw.c` and `bcm.c` after downloading Socket-CAN. This is a disadvantage of the 'external' module compilation.

For additional support on Socket-CAN you may also check the mailing-lists at: developer.berlios.de/projects/socketcan/ or developer.berlios.de/mail/?group_id=6475

17 Real-time Support with Xenomai

Since Release 'release_20061029_*' real-time support for the Xenomai real-time environment (www.xenomai.org) was merged from a previous rt-branch into the main branch. Most of the work was done by Laurent Bessard and Edouard Tisserant.

The Xenomai project has been launched in August 2001. It has merged in 2003 with the RTAI project [www.gna.org/projects/rtai] to produce an industrial-grade real-time Free Software platform for GNU/Linux called RTAI/fusion, on top of Xenomai's abstract RTOS core.

17.1 Installation

First of all, in order to install PCAN-rt-driver, a Xenomai real-time environment must be installed on your workstation. Download the Xenomai source code from the Xenomai website and install it by following the installation manual provided. The Linux kernel of your workstation must be patched for using Adeos (patches for a lot of architectures are included in the source code).

Before using Xenomai, the list of dynamic libraries of your workstation must be updated. If not, the driver compilation will abort. For that, you can use 'ldconfig', ensuring that the path to Xenomai libraries have been added to `/etc/ld.so.conf` file.

17.2 Compilation Environment

To compile the PCAN real-time driver, you must provide in the make command the option RT=XENOMAI. By default, the PCAN driver will be compiled for non-real-time environment.

Because USB and PCCARD are still not supported on PCAN real-time driver, it is highly recommended to not compile PCAN real-time driver with USB_SUPPORT or PCCARD_SUPPORT flags.

An example of command for compiling PCAN real-time driver:

```
cd ~/peak-linux-driver/driver
make clean
make RT=XENOMAI
```

Possible other make switches are RT=RTAI for a RTAI skin or RT=NO_RT for no 'real-time' support at all.

17.3 Runtime Environment

You can load PCAN real-time driver module as usual, there are no changes compared to the standard loading procedure. The PCAN-API is the same in real-time and non-real-time, so you can open, close, read from or write to your PCAN device like usual. But be careful, reading and writing must be made in a real-time task to be really effective. For an example of how to use Xenomai for creating and using a real-time task, please refer to the examples (transmitest and receivetest) located in the ‘test’ folder. For compiling your program, Xenomai libraries path and flags must be included in your ‘Makefile’. They can’t be obtained by using the ‘xeno-config’ script provided with Xenomai. You can have an example of how using ‘xeno-config’ in the ‘Makefile’ file located in the ‘test’ folder.

17.4 Troubleshooting

Attention! On an old station, after compiling the kernel patched for Adeos, a problem that makes the kernel couldn’t load can appear. To solve this problem, you can add ‘lapic=1’ to the

kernel command line parameters in ‘grub.conf’ or ‘lilo.conf’ file, depending on your bootloader.

If your kernel still crashes at start, a solution for finding what is blocking your kernel is to compile it with Xenomai support in module and try to insert it after kernel was loaded. It will then indicate in the log messages what the problem is.

The most common mistake when using a real-time task is trying to print some text in the user terminal with ‘printf’ command. It is highly recommended to avoid doing that. In most cases, it will result in a kernel crash.

18 FAQ

Q The compilation and installation worked well. I can write messages to the CAN-Bus but I cannot read any message.

A Probably you have a kernel version greater or equal than 2.6.25 and you compiled for NET_DEV_SUPPORT since this is the default compilation target for kernels greater and equal than 2.6.25.

Q The driver for my PCAN-ISA card is not being installed. What's wrong?

A One possibility is that the SJA-1000 chip on the card is setting back while checking the PeliCAN-mode. If the card with a PLD that has got a red point and the jumper JP11 is set “on“ this may be the reason. Remedy: To set the jumper to “off“.

Q My PCAN-ISA, CAN-PC/104 or PCAN-Dongle doesn't receive any data though I can send. What's wrong?

A A possible reason is that a wrong interrupt is configured for the device. Then single telegrams can be send however none can be received. Not only the wrong assertion of the interrupt number but often the missing release of the interrupt in BIOS („legacy ISA“) for the PCAN-ISA card can be the reason [2].

Q My PCAN-Dongle is being rejected while using it even if the installation was successful. What's wrong?

A Normally the reason is a not assigned interrupt for the „parport_pc“ while using it together with the PARPORT_SUBSYSTEM [2]

Q What possibilities do I have to diagnose if I have problems?

A Linux has a variety of choices. First it's useful to look at recent messages with:

```
"tail -f /var/log/messages"
```

If the driver is loaded you can see (as root) with:

```
„/sbin/lsmod | grep pcan"
```

More informations about the parameter of the driver can be asked with:

```
„/sbin/modinfo pcan"
```

Closer information about assignment of major- and minor-numbers and about the use of resources can be queried with:

```
"cat /proc/pcan"
```

Information about assignment of resources can be queried with:

```
"cat /proc/interrupts"
"cat /proc/ioports"
"cat /proc/iomem"
„cat /proc/pci" or „cat /proc/bus/pci/devices" depending on the kernel version.
„cat /proc/bus/usb/devices"
```

Note: The interrupts of the hardware aren't registered until its use.

To diagnose USB there are also programs with a graphical user interface like e.g. „USBView".

With garbled installations sometimes paths for lation of the driver are set wrong. That's why while translating the driver a message like this appears:

```
*** Host machine kernel version=2.4.18-4GB, Driver kernel version=2.6.5, \ Path
   to kernel sources=/mnt/usr/src/linux, use KBUILD=yes
```

With this message it is possible to compare if the driver is really compiled for the used kernel. The "KBUILD" mechanism is used from kernel 2.6.x on.

Q How do I interpret the output of "cat /proc/pcan"?

A The second line gives information about the release of the driver. In the third line the number of the found CAN-channels and the assigned major-number are shown. From line four on the characteristics of the each CAN-channel are described.

The first column shows the minor-number assignment of the channel, the second column shows the type of the channel, the third column shows the network device assigment i available. The fourth column shows the current BTR0BTR1 (bit rate) setting, the next 2 columns show the assignment to base addresses (ports) and interrupt-numbers. The following four columns count the number of processed read-, write-, interrupt- and errors transactions. The last column shows the last occurred error status of the channel. The status is a bit-field with an interpretation like this table:

Bit	Description
0	Chip-send-buffer full
1	Chip-receive-buffer overrun
2	BUS warning
3	BUS passive
4	BUS off
5	Receive buffer is empty
6	Receive buffer overrun
7	Send-buffer is full
14	Illegal parameter

Both columns „base“ and „irq“ are misused with USB channel types. „base“ is showing the serial number of the module and „irq“ the programmed device number of the module. The column ”irq“ in case of type USB counts the number of received or sent USB-packages.

Q What happens if I plug out the interface while using PCAN-USB?

A First of all your programs will stall and a re-plug-in of the PCAN-USB doesn’t change anything. First you have to stop and after

plugging-in of the interface you can invoke your programs again. It's not guaranteed that the same minor number as the previous PCAN-USB minor-number is allocated. Minor-number assignment depends on the order of plug-ins of PCAN-USB devices.

Q How can I find out what minor-number the PCAN-USB is assigned to?

A With help of the individual pre-programmed „device-number", which can be experienced with help of the „/proc/pcan" interface, a relationship to a special physical connection can be made.

Q I want to open my PCAN-USB device however I get an error message. What's wrong?

A There is more than one possibility: 1) The appropriate device is not registered as „/dev/pcan...". 2.) The appropriate device is not plugged in.

Q The compilation on my system doesn't work. What can be wrong?

A The most common reason is not installed Linux kernel sources. Often sources of another kernel than the one installed are used. Sometimes both files

„/boot/vmlinuz.autoconf.h" and
„/boot/vmlinuz.version.h"

of the distribution are not linked to the files they are assigned to

„/lib/modules/'uname-r'/build/include/linux/autoconf.h" and
„/lib/modules/'uname -r'/build/include/linux/version.h"

Q The transmission performance of CAN-messages is low, if I send a telegram with help of the write-interface out of a script. What is the problem?

A Every line in a script opens the path to a device and immediately closes this path again. This opening and closing take time. The

transmission performance can be improved by letting another path open. For example:

```
cat /dev/pcan32 >/dev/null &
echo „m s 0x123  2 0x11 0x12“ > /dev/pcan32
echo „m s 0x123  2 0x11 0x12“ > /dev/pcan32
```

Q I got a lot of error messages when compiling the driver for a kernel 2.6.x system. What's wrong?

A To compile for kernel 2.6 target system you need to have a pre-configured kernel. To accomplish this you need to install the target kernel sources. Then do

```
cd /usr/src/linux # for example
su                # you need to be root
make cloneconfig  # create a configuration suitable for your running kernel
make scripts      # create the necessary scripts
```

That's all. Now you can – as ordinary user – compile your driver.

Q While compiling the driver following message appears:

```
*** “Can't find /usr/src/linux-2.4.24.SuSE/include/linux/version.h !”.  End.
```

What's the problem?

A Within the driver makefile up from version 3.3 the target kernel version is not being won out of the interpretation of the command 'uname -r', but extracted out of contents of the file “$(KERNEL_LOCATION)/include/linux/include/version.h“. (If there wasn't any special “KERNEL_LOCATION“ given at the command line of „make“ “KERNEL_LOCATION=/usr/src/linux“ is being used as default.) While translating the kernel sources the file `version.h` is generated. Normally this message has its cause in not translated kernel sources. Until kernels 2.4.x it was enough if the distribution had installed this file. From kernel 2.6.x on it is necessary that the kernel sources for the target system are configured and translated completely. Then the file `version.h` is being created.

With ubuntu systems you should set KERNEL_LOCATION=/usr/src/Linux-headers-‘uname-r’.

Q I like to have interrupt sharing with PCAN-ISA or PCAN-PC/104

A No problem since version 3.30 of the driver. But you should be aware that interrupt sharing with ISA-BUS is only possible with devices which are supported from the same driver, e.g. the PCAN driver. For example it is not possible to share a serial device with a PCAN device but two PCAN devices can share the same interrupt level.

Q I am trying to use the PCAN driver with Xenomai support. My problem is, when I load the module it registers the device with major 000 and doesn't do an entry into /proc/devices, only into /proc/xenomai and /proc/pcan. Therefore pcan_make_devices doesn't work. Is that an error or supposed to be like that?

A It seems you want to use the Xenomai RTDM driver as a standard driver. You have to know that, with Xenomai's Real-time Driver Model, there is no real /dev/pcanX device node created. This is the RTDM skin that provide the appropriates syscalls to user space libs (libnative, librtdm,...). The pcan_make_devices script is not needed and do not work with Xeno.

pcanlib is also compiled differently with xenomai. It is linked with xenomai user space libs, and does not use real /dev/pcanX, but calls RTDM system calls.

In other world you have to fully recompile peak Linux-driver, pcanlib and examples with RT=XENOMAI, reinstall and reload module to be able to use PCAN with Xenomai. Do not expect to have some real /dev/pcanX nodes working if PCAN have been compiled with RT=XENOMAI, they only "virtually" exist for pcanlib with RTDM.

You also have to notice that in real-time, tests are not as verbose as in non-real-time. This is due to the fact that printing to screen is not

recommended from a real-time thread. Tests do just display statistics at the end of the test (after Ctrl-C).

For a minimal “hello word” example see in “test/minimal_xenomai” directory.

For a more concrete example of use of PEAK-Linux-driver with Xenomai, please consult source code of the CanFestival project (OpenSource CanOpen stack, www.canfestival.org).

Q I have problems to compile or load the driver. What can I do?

A First, read this manual to see if you got a well known problem. Second, if it is a compiling problem then redirect your compiler output into a file and send the contents to linux@peak-system.com. If it is a loading problem do the same for the output of your loading procedure. With loading or runtime problems it is very helpful to run a driver with debug mode enabled. Then please attach the relevant parts of “/var/log/messages”.

Q I got other questions or problems with the driver.

A Please consult linux@peak-system.com.

Q I am enthusiastic about the driver or the documentation and want to contribute something to make it better. Who do I have to consult?

A Please also contribute linux@peak-system.com. Peak and the authors are happy about every positive response.

Q When the socketCAN driver “peak_pci.ko” was installed the driver “pcan.ko” didn’t load anymore. What can I do?

A If your system runs udev you only need to copy the file `blacklist-peak_pci.conf` into “/etc/modprobe.d”. The file `blacklist-peal_pci.conf` is located in the Documentation folder. You need to do this as “root” user, e.g. with

```
sudo cp Documentation/blacklist-peak_pci.conf /etc/modprobe.d
```

After copying please restart your computer and “peak_pci.ko” will not load anymore. To remove this option you simply need to erase “/etc/modules.d/blacklist-peak_pci.conf” and restart.

Appendix A Usage of the Driver with chardev Mode

There are 4 options to 'talk' with the chardev driver. Some code snippets explain the options.

A.1 Polling

Polling is the worst of all possibilities. But sometimes it becomes the easiest choise.

To poll the driver path have to be opend with the option O_NONBLOCK:

```
#include <fcntl.h>
#include <libpcan.h>

TPCANRdMsg msg;
int ret;

HANDLE h = LINUX_CAN_Open("/dev/pcan0", O_RDWR | O_NONBLOCK);

do
{
  ret = LINUX_CAN_Read(h, &msg);
} while (ret == -EAGAIN);

if (!ret)
{
  /* use 'msg' */
}
else
  ...
```

A.2 Blocking Wait

Blocking wait should be preferred because it preserves system resources. Normally blocking wait must run inside a separate thread to avoid blocking of the program.

```
#include <fcntl.h>
#include <libpcan.h>

TPCANRdMsg msg;
int ret;

HANDLE h = LINUX_CAN_Open("/dev/pcan0", O_RDWR);

/* at this point the thread blocks until a message is available */
/* or an error occurs */
ret = LINUX_CAN_Read(h, &msg);

if (!ret)
{
  /* use 'msg' */
}
else
  ...
```

A.3 Blocking Wait with Timeout

This method is a mix between the two predecessors. Reading of the data blocks until a message is available, an error occurs or the given timeout elapses. It is more or less a skyhook to create simple programs without wasting system resources.

```
#include <fcntl.h>
#include <libpcan.h>

TPCANRdMsg msg;
```

```
int ret;

HANDLE h = LINUX_CAN_Open("/dev/pcan0", O_RDWR);

do
{
  /* at this point the thread blocks until a message is available, */
  /* an error occurs or the timout elapses */
  ret = LINUX_CAN_Read_Timeout(h, &msg, 1000000); /* one second */
} while (ret == CAN_ERR_QRCVEMPTY);

if (!ret)
{
  /* nutze 'msg' */
}
else
  ...
```

A.4 Waiting for an IO-event with Select()

Waiting for an IO-event with select() is very resource efficient but complicated to use.

With select() you can wait for multiple read-, write- and exceptions events simultaneously. Additionaly you can give a timeout as a maximum waiting time.

You can get more information with ‘man select’.

The following example monitors 2 path’s for an input event:

```
#include <fcntl.h>
#include <unistd.h>
#include <libpcan.h>

TPCANRdMsg msg;
```

```
int ret;
int maxfd;
fd_set watchset, testset;   /* create 2 sets to carry file descriptors;

HANDLE h = LINUX_CAN_Open("/dev/pcan0", O_RDWR);
/* request the associated file descriptor */
int fdpcan = LINUX_CAN_FileHandle(h);

FD_ZERO(&watchset);         /* clear the set of file descriptors */
FD_SET(fdpcan, &watchset);  /* add the pcan descriptor */
FD_SET(fdother, &watchset); /* add another input descriptor */

/* determine the highest used file descriptor */
maxfd = (fdpcan > fdother) ? fdpcan : fdother;

/* test if the file descriptors still in the set */
while ((FD_ISSET(fdpcan,  &watchset) ||
       (FD_ISSET(fdother, &watchset)
{
  /* watchset will be used later on again */
  testset = watchset;

  /* wait until a input is ready */
  err = select(maxfd + 1, &testset, NULL, NULL, NULL);
  if (err < 0)
  {
    /* an error is occured */
    return ...;
  }

  if (FD_ISSET(fdpcan, &testset)
  {
    /* read a message */
    ret = LINUX_CAN_Read(h, &msg);

    if (!ret)
    {
```

```
      /* use 'msg' */
    }
    else
      ...
  }

  if (FD_ISSET(fdother, &testset)
  {
    /* do what has to be done to delete the IO-event */

    /* e.g. delete a file descriptor from the set */
    close(fdother);
    FD_CLR(fdother, &watchset);
  }
}
```

Appendix B Historical Parts

B.1 devfs

Up from Release_20030622_x the device file system is being supported optionally. From kernel version 2.6.x the kernel developers depreciate the use of devfs so for the time being the driver is not supporting this. Up from release release_20060501_a (version 4.0) the parts for supporting devfs are removed from the driver.

Using the devfs it's not necessary to create device files manually any more. This is taken over by the kernel in cooperation with the driver and the daemon devfsd. The device file system has to be enabled within the kernel.

Compiling for devfs-file system support. If you translate the sources for a system with devfs-support (CONFIG_DEVFS_FS = y) this will be considered automatically:

```
cd ~/peak-linux-driver/driver
make clean
make DEVFS=DEFS_SUPPORT
```

Q I compiled for device file system support. Though it’s not working.

A Is your devfsd – the “device file system daemon“ – active? Please check that.

B.2 Compilation of Kernel Greater Than 2.5.x

Q While translating kernel 2.6.x following warning appears:

```
*** Warning: Overriding SUBDIRS on the command line can cause
```

```
*** inconsistencies
```

Is something not right?

A I am afraid that the discussion within the kernel-developer community about the right way of translating external modules hasn't found an end until this release. The current way generates this warning and at worst translates **all** of the modules again. Though there are approaches that the generation of external modules like pcan.ko is getting easier. This message can be ignored.

B.3 Obfuscation

Q The source code module "pcan_usb_kernel.c" only contains cryptic letters. Is the file destroyed?

A No, the contents of the files is obfuscated to save intellectual property. Though it's still a translatable source code. This feature does only apply for releases earlier than 20061029.

Q During installation of the driver with USB-support I get following message: "Warning: loading pcan.o will taint the kernel: non-GPL license – Proprietary". What's the meaning of this message?

A This message points out that the driver for the PCAN-USB module was not released under the GPL license. The use of the driver as a module within Linux though is legal.

B.4 History of the Document

Text	Datum
First draft	Hi 01/13/2002
Rescue after loss of data	Hi 02/10/2002
Format revision, read/write tables	Hi 02/20/2002

Removed typos	Hi 02/21/2002
PCAN-USB integrated	Hi 02/09/2003
Obfuscation of „pcan_usb_kernel.c“ is described	Hi 02/23/2003
devfs, kernel-2.5 support, LGPL	Hi 08/04/2003
1st revision for kernel 2.6.x driver	Hi 05/02/2004
Enlarge FAQ, corrections	Hi 05/13/2004
Simple correction for RPM make procedure	Hi 08/14/2004
Additional compiler switches	Hi 07/19/2005
Replaced cat “i ...” with echo “i ...”	Hi 11/30/2005
Added a hint from Uwe Bonnes	Hi 03/14/2006
Added PCAN-PC Card support, removed devfs	Hi 05/14/2006
Corrected serious typographic error	Hi 05/16/2006
Removed obfuscation, added real-time support	Hi 10/29/2006
Minor improvements	Hi 11/04/2006
netdev, message filters and error handling described	Hi 02/27/2007
Added a special paragraf to explain NETDEV compilation target	Hi 01/18/2009
udev support explained, support up to kernel 2.6.29	Hi 02/03/2009
udev issues detailed	Hi 02/14/2009
Rework of the manual	Hi 01/16/2010
Added a hint how to blacklist peak_pci.ko.	Hi 03/28/2010

[1] CAN hardware is only accepted based on Philips chip SJA1000. PCAN-ISA or -PCI hardware is immediately verified during installation. In contrast to that the PCAN-Dongle hardware is not verified until an „open()“ call. That's the reason why no error is reported if a parallel port without plugged-in PCAN-Dongle is recognized.

[2] The „Parport Subsystem“ has to be configured to use a interrupt. Therefore following lines have to be entered into the file „/etc/modules.conf“ or „/etc/modprobe.conf“ depending on your kernel version:

```
alias parport_lowlevel parport_pc
options parport_pc io=0x378 irq=7
```

Normally „irq=none“ stands in the „options“-line. The assigned interrupt number must match the assigned interrupt number of the device. Those works can only be done as 'root“. If the „Parport Subsystem“ is already installed it can be removed with:

```
rmmod lp
rmmod pcan
rmmod parport_pc
rmmod parport
```

[3] You are not forced to use the „Parport Subsystem“. Then the alternately use of the parallel port through PCAN-Dongle and e.g. a printer is not possible any more. For it the driver has to be translated with the PAR=NO_PARPORT_SUBSYSTEM option. Then the previous detail of „modprobe parport“ is not necessary while installing.

[4] In some cases the dynamic assignment of „major“- device numbers are unwelcome. However this can be changed in the driver sources (file „pcan_main.h“, constant „PCAN_MAJOR“). Afterwards the driver has to be translated and installed again. The best is to use the number „91“. This already is intended for CAN-devices. Please note that conflicts to other installed CAN-drivers can be possible.

[5] From kernel version 2.6 on the behaviour during assignment of the „minor“-numbers is being changed through the kernel parameter „CONFIG_USB_DYNAMIC_MINORS“. When „CONFIG_USB_DYNAMIC_MINORS“ is set, the first free „minor“-number is assigned to from 0 on. Mandrake distributions use this configuration.

[6] The „read“- and „write“-interface of the driver print out ASCII formatted data or accept ASCII formatted data as input. The format of the telegrams for „read“ and „write“ is defined identically, so that a redirection of data is possible:

```
cat  /dev/pcan0 > /dev/pcan8
```

The format of the data is like the previously mentioned description of the telegrams. Additionally to the description of the telegrams the „read“-output supplies a time stamp in milliseconds and microseconds. If there is a redirection like above the time stamp is being ignored.

The „write“-output also accepts details about initializing:

```
echo „i 0x1234 e“  > /dev/pcan8
```

The first parameter pinpoints the initialization, the second parameter names an input value for both BTR0/BTR1 registers of the SJA1000 chip. The optional third parameter „e“, enables the accept ion of extended frames.

Naturally a faster „ioctl()“-interface is also defined.