# Contents

# Vorwort

Diese Anleitung ist fast vollständig. Sie ist auch keine Diskussionsgrundlage, ob Windows oder Linux besser ist. Auch wird hier nicht im Detail beschrieben, wie ich ein System zu installieren habe, mit dem ich *git, gcc, g++, make und qmake* nutzen kann. Diese Tools gibt es für Linux und Windows! Allerdings findet man deutlich mehr Anleitungen im Internet, die davon ausgehen, dass man diese Tools unter Linux nutzt, da sie sich unter Windows nicht bewährt haben. Um also Ärger zu vermeiden: Setze dich mit Linux (primär Archlinux) auseinander und installiere z.B. in einer virtuellen Maschine Archlinux.

Debian, Ubuntu, Suse, Mint sind als Linux-System nicht schlecht. Sie sind aber deutlich anders als Archlinux. Da in den nächsten Kapiteln eine SD-Karte für den Raspberry Pi B+ so vorbereiten wird, dass auf diesem Archlinux läuft, ist es nicht verkehrt, sich mit der Installation von Archlinux auf einem normalen PC (= Virtuelle Maschine) auseinander zu setzen. Plane als ungeübte Person

**zwei Tage** ein, bis du mit der Installation fertig bist (grafisches System mit funktionierendem Internet und QtCreator).

**Jochen Peters, 14.03.2015**

# Ziel

Ziel dieser Anleitung ist es ein *Raspberry Pi b+* mit einem Canberry soweit ans laufen zu bekommen, dass ein gesendetes Byte von einem Windows-PC (CAN-Bus USB) in einem grafischem Display als "Tacho" dargestellt wird.

## Warum Archlinux?

- Alles ist deutlich aktueller als die oben genannten Systeme
- Es ist minimalistisch: nur das nötigste installierst du von Hand
- Lerneffekt: worauf kommt es bei Linux an, damit es läuft?
- Erprobt: die beschriebenen Schritte wurden in den ersten zwei Wochen des Dezember 2014 erfolgreich durchgeführt.

## Unvollständig oder fehlerhaft?

Diese Anleitung kann nicht vollständig sein. Ebenso kann sie fehlerhaft sein. Dies liegt vor allem diesen Punkten:

- Du installierst und erstellst Software, die ständig erneuert und aktualisiert wird:
  - es kommen Fehler hinzu
  - es ändern sich Techniken in der Konfiguration
  - du orientierst dich an Anleitungen im Internet, die veraltet sind und machst damit mehr kaputt als richtig
- Widersprüche, Fehler, Lücken oder Unverständliches, weil ich meine Tätigkeiten zum Erfolg nicht tadellos notiert habe

# SD-Karte mit Archlinux erstellen

Vorab: von den deutschsprachigen http://archlinux.de Seiten kann ich nur abraten. Diese Seiten sind sehr oft veraltet. Da Archlinux fast täglich ein wenig ändert, sind aktuelle Wiki-Seiten sehr wichtig. Selbst auf http://archlinux.org sind immer wieder Stellen, wo auf die mangelnde Aktualität hingewiesen wird.

In der deutschen Seite fehlen die Hinweise oft oder führen nicht zu einem wiki-Artikel, der die Neuerung beschreibt.

Zur Installation von Archlinux auf dem Raspberry Pi (ARM-Prozessor) kann ich nur diese Seite empfehlen:

http://archlinuxarm.org/platforms/armv6/raspberry-pi

## Basiskonfiguration das Systems auf dem Raspberry Pi

Der Einfachheit halber: bitte erstmal dafür sorgen, dass dein Raspberry Pi mit einem Netzwerk-Kabel und DHCP eine IP-Adresse bekommt. Die Anleitung, besonders zu WLAN kann einen etwas überfordern:

- https://wiki.archlinux.org/index.php/Network_configuration
- https://wiki.archlinux.org/index.php/Wireless_network_configuration

Diese kommende Anleitung orientert sich an:

- https://wiki.archlinux.org/index.php/Installation_guide
- https://wiki.archlinux.org/index.php/Beginners%27_guide>

Ok, du bist nun als root (Passwort: root) auf deinem Raspberry Pi eingeloggt. Du hast auch bereits mit

```
ping -c 3 google.de
```

getestet, dass dein Netzwerk mit der Aussenwelt sprechen kann. Mit **passwd** kannst du als root jederzeit dein root-Passwort ändern. Weiter geht es mit

```
loadkeys de
```

für ein deutsches Tastatur-Layout. Umlaute funktionieren jetzt noch nicht unbedingt. Ausserdem willst du **de** immer beim Einloggen haben. Dazu später mehr. Mache nun ein komplettes Update deines Systems:

```
pacman -Syu
```

Im nächsten Schritt, um etwas angenehmer arbeiten zu können, installierst du dir den Midnight-Commander:

```
pacman -S mc
```

Damit du deine USB-Maus benutzen kannst, musst du *gpm* installieren:

```
pacman -S gpm
```

Um *gpm* zu nutzen, solltest du es in die Liste der Programme aufnehmen, die immer laufen sollten:

```
systemctl enable gpm
```

Und zum Starten von *gpm*:

```
systemctl start gpm
```

Nun solltest du mit deiner maus und der rechten Maustaste alles mögliche auf dem Bildschirm markieren und mit der mittleren Maustaste einfügen können.

Mit dem Midnight-Commander (*mc*) und seinem Editor (*mcedit*) kannst du die Maus ebenfalls nutzen.

```
mcedit /etc/locale.gen
```

Gehe nun in die Zeile, in der **#de_DE.UTF-8 UTF-8** steht. Entferne das Kommentarzeichen # und speichere deine Änderung ab. Gebe nun ein

```
locale-gen
```

um das System auf deutschen UTF-8 Zeichensatz um zu stellen. Damit alle Programme das mitbekommen, muss in die Datei **/etc/locale.conf** die Variable LANG mit de_DE.UTF-8 definiert werden. Die Datei kann erzeugt und mit LANG=de_DE.UTF-8 beschrieben werden, in dem man z.B. die Ausgabe vom Befehl **echo** mit dem > Operator in die Datei umleitet:

```
echo LANG=de_DE.UTF-8 > /etc/locale.conf
```

(Mit **mcedit /etc/locale.conf** hätte man die Datei natürlich auch erzeugen und darin LANG=de_DE.UTF-8 schreiben können.)

Um die Deutschsprachigkeit ab zu schließen, erstelle eine Datei

```
mcedit /etc/vconsole.conf
```

Mit dem Inhalt:

```
KEYMAP=de
```

Jetzt muss die Systemzeit noch auf *Berlin* gestellt werden. Hierzu wird ein Symbolischer Link auf die Zeitkonfiguration erstellt. Ersmal musst du die alte zeitkonfig-Datei löschen:

```
rm /etc/localtime
```

Jetzt einen symbolischen Link erstellen:

```
ln -s /usr/share/zoneinfo/Europe/Berlin /etc/localtime
```

Stimmt nun die Uhrzeit?

```
date
```

Wenn nicht, dann mit -s die Zeit und das Datum setzen:

```
date -s 2014-12-31
date -s 20:45
```

Nun ist ein guter Zeitpunkt, um das System neu zu starten! Gebe *exit* ein, um dich aus zu loggen und mit STRG+ALT+ENTF reboote das System.

Logge dich nach dem Reboot erneut als root ein. Nun installiere dir ein grafisches System. Optimal ist das lxde, aber xfce4 ginge auch. Bei Linux muss man klar trennen: X, oder der XServer (Xorg oder XF86) stellt nur eine Möglichkeit dar, in einem Bereich einen Fenster-Inhalt dar zu stellen. Streng genommen kann man über das Netzwerk seinen eigenen XServer nutzen, und das Programm läuft auf dem Raspberry Pi - wird aber auf deinem Linux-PC dargestellt. Mit http://x.cygwin.com/ gibt es auch einen XServer für Windows, so wie es viele andere Linux-Sachen bei cygwin für Windows gibt.

Für Fensterrahmen, Funktionen zum Schließen oder Verschieben von Fenstern gibt es unter Linux Fenstermanager oder ganze *Desktop Enviroments*, die etwas mehr können, eigene Programme/Apps haben, die nur unter Gnome, KDE oder XFCE4 laufen und sich stärker in das System "einmischen".

Wir nehmen aber **lxde**, was eigentlich nur ein paar eigenständige Programme sind und als Fenstermanager **openbox** verwendet.

Im ersten Schritt installieren wir den XServer, und ein paar Dinge, die man noch zusätzlich braucht:

```
pacman -S xorg-xinit xorg-server xorg-server-utils xterm
```

Nun kommt lxde hinzu:

```
pacman -S lxde
```

Wähle als Voreinstellung alle - also mach einfach “Return/Enter”. Weiter geht es mit dem (primitiven) Grafikkarten-Treibern:

```
pacman -S mesa xf86-video-fbdev xf86-video-vesa
```

So, eigentlich wären wir hier schon fast fertig. Mit **startx** könntest du den XServer bereits starten. Schöner wäre aber ein automatischer Start und ein grafisches Login-System. Hier empfehle ich **lightdm** und zwar in der gtk3 Version:

```
pacman -S lightdm-gtk3-greeter
```

Auch hier muss man dem System sagen, dass man es beim Systemstart automatisch haben will:

```
systemctl enable lightdm
```

und zum Starten:

```
systemctl start lightdm
```

Das kann nun ein paar Sekunden dauern. Möglicherweise kannst du dich auch nicht als root einloggen, da lightdm dies als Sicherheitsrisiko sieht. Achtung: leider ist nun im grafischen System wieder das englische Tastaturlayout aktiv, was beim Passwort Ärger machen könnte!

Es ist ratsam, nun einen User pi an zu legen. Dafür wechselt du mit der Tastenkombination

```
STRG+ALT+F1 .. F6
```

auf eine freie Textkonsole. Auf einer (F1 = tty1) hast du den XServer gestartet, auf einer anderen läuft der bereits (lightdm Login). Logge dich einfach nochmal auf tty4 (F4) als root ein, und erstelle den user pi:

```
useradd pi
```

Gebe dem user ein Passwort und evtl. weitere Infos, die nicht notwendig sind. Mit **passwd** kannst du später als *pi* das Passwort neu setzen. Als root geht das sogar ohne Abfrage des alten Passworts mit **passwd pi**

Der User pi braucht ein home-Verzeichnis. Das wird möglicherweise nicht automatisch erstellt. Wechsel als *root* in den home-Ordner:

```
cd /home
```

Lasse dir den Inhalt (leer?) mit *ls* anzeigen. Dieser Befehl, zeigt ein paar zusätzliche Infos (mit Farben) an:

```
ls -lah --color
```

Da der Home-Ordner leer ist, hier ein Beispiel, was dieser Befehl so kann:

```
ls -lah --color /lib/
```

Ordner sind dunkelblau. Da **ls -lah –color** etwas lang ist, kannst du ein *alias* definieren, der immer beim Nutzen der Bash-Shell (die Text-Eingabeaufforderung, in der du nach dem Login auf einer Text-Konsole landest) aktiv wird. Dazu musst du den Befehl:

```
alias la='ls -lah --color'
```

in die Datei /etc/bash.bashrc am Ende schreiben:

```
mcedit /etc/bash.bashrc
```

Erst beim erneuten Login wird das dann ausgeführt. Alternativ kannst du aber den Befehl jetzt eingeben, damit du mit **la** den Befehl *ls -lah –color* nutzen kannst.

Also: ist nun kein **pi** im /home Ordner angelegt, mache du es mit

```
mkdir pi
```

oder

```
mkdir /home/pi
```

falls du nicht mehr mit *cd* im /home Ordner bist. Gebe dem Ordner nun die Rechte des users pi und der Gruppe pi:

```
chown pi /home/pi
chgrp pi /home/pi
```

Nun ist ein guter Zeitpunkt, den Raspberry Pi neu zu starten. Er sollte nach einer Weile (< 1 Min.) den graphischen Login (englisches Tastaturlayout) zeigen. Logge dich nun als user pi ein!

Weitere Hinweise, falls etwas nicht klappen sollte, findest du evtl. hier:

- http://blog.adityapatawari.com/2013/05/arch-linux-on-raspberry-pi-running-xfce.html
- https://wiki.archlinux.org/index.php/Raspberry_Pi

## Phase 2: Webbrowser, qt5, Texteditor und mehr

Wechsel in eine Textkonsole (z.B. mit STRG+ALT+F3) und logge dich als root ein. Um ein brauchbares System zum Erzeugen von Programmen zu haben und evtl. auch mal mit dem User pi ins Internet gehen zu können, sollte man dies noch installieren:

```
pacman -S qtcreator gcc make midori geany wget git
```

qtcreator ist eine IDE um Windows-Programme mit c++ entwickeln zu können. Derzeit macht die etwas Ärger auf dem Raspberry Pi und stützt ab. Trotzdem bitte installieren, damit wir die Programme des normalen Linux-PC, auf dem wir auch qtcreator installieren und nutzen werden, auch auf dem Raspberry Pi nutzen zu können.

Geany ist ein sehr guter Text-Editor, den es übrigends wie den Browser Midori und die IDE qtCreator auch für Windows gibt! *wget* ist ein Tool für die Text-Konsole, um aus dem Internet z.B. Dateien herunter zu laden. Das *git* ist sowas ähnliches, aber ein ganzes Versions-Management System steckt dahinter. Auf dem Linux-PC wirst du es später nutzen, um Tools zur Erstellung des neuen Linux-Kernels zu installieren und auch um den neuen Linux-Kernel für den Raspberry Pi zu bekommen.

Um z.B. mit dem *Midnight-Commander* zip, tar.gz und andere Dateien öffnen zu können, solltest du noch folgendes installieren:

```
pacman -S unzip zip gzip tar p7zip bzip2
```

Um im graphischen System eine deutsche Tastatur zu haben, musst du auf das X Symbol (“Startmenu”) unten links klicken, und unter *Systemwerkzeuge* findest du *LXTerminal*. Gebe in das Terminalfenster

```
setxkbmap de
```

ein. Damit du die deutsche Tastatur auch schon beim Login-Manager *lightdm* nutzen kannst, gebe als *root* diesen Befehl ein:

```
localectl set-x11-keymap de
```

Nach einem Reboot soltest du dann immer ein deutsches Layout haben. Leider stellen die archlinux Entwickler gerade alles etwas um, so dass es derzeit wenig zu dem Thema gibt. Eine Anpassung in xorg-Dateien ist zumindest nicht mehr "state of the art". Allerdings macht das der obige Befehl.

Der nächste Punkt wäre der das network-manager-applet. Das Ding hockt später neben der Uhr uns sogt dafür, dass man mit der Network-Konfiguration etwas leichter zurecht kommt (z.B. VPN, oder WLAN).

```
pacman -S networkmanager network-manager-applet networkmanager-openvpn
```

hierzu muss der der networkmanager dem System bekannt gemacht werden, damit auch alles zu beginn läuft:

```
systemctl enable NetworkManager
systemctl start NetworkManager
```

Als letzten Schritt sollte dafür gesorgt werden, dass USB-Speichermedien automatisch eingebunden werden. Dafür brauchst du

```
pacman -S udev autofs gvfs
```

Wenn du willst, kümmere dich nun um die Soundkarte und dessen Einrichtung, sowie den Betrieb von bluetooth, einem Mediaplayer wie smplayer oder vlc. Im wiki von archlinux.org wirst du bestimmt etwas finden. Ebenfalls interessant sind die Anpassung von sshd_config und ssh_config (goggle mal nach X11 forwarding) sowie die Einrichtung einer Firewall (google nach ufw).

Damit wärst du mit dem System auf dem Raspberry Pi schon mal fertig.

## Treiber und Tools für den Raspberry Pi

Nun geht es weiter mit deinem Archlinux-PC, oder mit deiner Virtuellen Maschine, in der du (evtl. mit der Hilfe aus dem letzen Kapitel) ein Archlinux zurechtgemacht hast. Sollte mal etwas fehlen, dann einfach mit

```
pacman -S packetname
```

nachinstallieren. Zum Suchen des Packets, kannst du

```
pacman -Ss suchwort
```

benutzen. Praktisch sind z.B. audio-Tools, Flash und Medienplayer:

```
pacman -S alsa-oss alsa-utils alsa-tools flashplugin smplayer vlc
```

Ebenfalls nützlich ist ein Pack/Entpack Tool, was später im Kontextmenu des Dateimanagers (pcmanfm) von lxde zu finden ist:

```
pacman -S xarchiver
```

Und wenn man PDFs lesen und ein wenig bearbeiten will, installiert man am besten

```
pacman -S xournal
```

## Dein Linux-PC vorbereiten

Da der Raspberry Pi nicht der schnellste ist, macht es Sinn deinen richtigen Rechner für die kommenden Aufgaben zu benutzen. Mache dir ein Terminal Fenster auf, in der du nun etwas länger "unterwegs" sein wirst.

Wechsel in der Home Verzeichnis:

```
cd ~
```

Erstelle dir für die Kommenden Aufgaben ein raspi-Verzeichnis:

```
mkdir raspi
```

Nun solltest du für dieses Terminal eine BASEDIR Variable definieren:

```
export BASEDIR=$(pwd)/raspi
```

du brauchst diese und weitere Variablen hinterher, um dem Kompiler ein paar Informationen zu übergeben. Diese Variablen gelten NUR so lange, wie dieses Terminal Fenster geöffnet ist!

Wechsel nun in deine BASEDIR:

```
cd $BASEDIR
```

Nun kann es etwas längern dauern, da du den aktuellen Linux-Kernel des Raspberry Pi mit git aus dem Internet holen musst (ca 650 MB):

```
git clone --depth 100 https://github.com/raspberrypi/linux.git
```

Versichere dich, dass du danach einen linux-Ordner hast. Die nächste Variable ist:

```
export KERNEL_SRC=$BASEDIR/linux
```

Weiter geht es mit dem auschecken / clone der Cross-Compiler-Tools (ca. 330 MB):

```
git clone git://github.com/raspberrypi/tools.git
```

In der Zwischenzeit kümmere dich wieder um deinen Raspberry Pi. Versuche mit ssh/scp oder mit einem USB-Stick an die gZippte-Datei /proc/config.gz zu kommen. Diese Datei enthält alle Konfigurationen des aktuell laufenden Systems auf deinem Raspberry Pi (ich meine damit: Nur des Kernels!). Lege die Datei in deine BASEDIR raspi. In einem 2. Terminal kannst du während die "tools" auschecken, die config.gz entpacken und als .config Datei in den linux Ordner legen:

```
zcat config.gz > linux/.config
```

Das Programm **zcat** ist das selbe wie das oben erwähnte **cat**, aber es kann eine z-Komprimierte Datei entpacken. Ist das Auschecken der Tools mit git nun fertig? Dann mache nun dort weiter:

```
export KERNEL_SRC=$BASEDIR/linux
export CCDIR=$BASEDIR/tools/arm-bcm2708/arm-bcm2708-linux-gnueabi/bin
export CCPREFIX=$CCDIR/arm-bcm2708-linux-gnueabi-
```

Schlimm, was der Cross-compiler so alles an Variablen gesetzt haben will, nicht wahr? Wechsel nun in das linux-Verzeichnis:

```
cd $KERNEL_SRC
```

Dort sollte nun deine .config Datei liegen, die du jetzt zu einer neuen Version des neuen, ausgecheckten Kernels umwandeln musst:

```
ARCH=arm CROSS_COMPILE=${CCPREFIX} make oldconfig
```

Es werden dir eine Menge Fragen zu neuen Konfigurationsmöglichkeiten gestellt, die du mit dem Default-Wert *durch-entern* kannst (hoffentlich).

Jetzt, wo die Config-Datei zum neuen Kernel angepasst ist, geht es darum den Kernel an zu passen:

```
ARCH=arm CROSS_COMPILE=${CCPREFIX} make xconfig
```

Orientiere dich bitte an diesen Einstellungen:

```
[*] Networking support --->
....<M> CAN bus subsystem support --->
........<M> Raw CAN Protocol (raw access with CAN-ID filtering)
........<M> Broadcast Manager CAN Protocol (with content filtering)
............CAN Device Drivers --->
..............<M> Platform CAN drivers with Netlink support
..............[*] CAN bit-timing calculation
..............<M> Microchip MCP251x SPI CAN controllers
..............[*] CAN devices debugging messages
....Device Drivers --->
........[*] SPI support --->
............<M> BCM2798 SPI controller driver (SPI0)
............<M> User mode SPI driver support
.......-*- GPIO Support --->
............[*] /sys/class/gpio/... (sysfs interface)
```

Stelle am Besten zu erst die SPI-Sachen ein, damit ein paar andere Sachen im CAN-Support auch aktivierbar sind. Speichere das ganze wieder in der .config Datei (default) ab.

Nun kommt der etwas häßliche Teil, da ich mir nicht sicher bin, ob ich wirklich in dem Code des Modul (=Treiber) vom SPI-Chip (bcm2708) des Raspberry Pi etwas gravierendes geändert habe. In der Datei *linux/arch/arm/mach-bcm2708/bcm2708.c* habe ich glaube ich nichts verändert, da der 3.12.y Kernel, den ich hatte, bereits für den bcm2515 und das Modul spi-config angepasst war.

Deswegen geht es weiter mit dem Compilieren des Kernels (kann bis zu 2h dauern):

```
ARCH=arm CROSS_COMPILE=${CCPREFIX} make
```

Weiter geht es mit einem Modul-Ordner, weiteren Variablen und dem Kompilieren der Module:

```
mkdir $BASEDIR/modules
export MODULES_TEMP=$BASEDIR/modules
ARCH=arm CROSS_COMPILE=${CCPREFIX} INSTALL_MOD_PATH=${MODULES_TEMP} \
    make modules_install
```

Wechsel wieder in deine BASEDIR:

```
cd $BASEDIR
```

und hole dir das Modul zu spi-Konfiguration:

```
git clone https://github.com/msperl/spi-config
```

Wechsel in das Verzeichnis:

```
cd spi-config
```

und erstelle das Modul:

```
ARCH=arm CROSS_COMPILE=${CCPREFIX} make KDIR=$KERNEL_SRC
ARCH=arm CROSS_COMPILE=${CCPREFIX} INSTALL_MOD_PATH=${MODULES_TEMP} \
    make KDIR=$KERNEL_SRC install
```

Gehe in das Modul-Verzeichnis und finde heraus, zu welcher Kernel-Version Module erstellt wurden:

```
cd $MODULES_TEMP
cd lib/modules/
ls -lah
```

Angenommen, es ist **3.12.34-ARCH**, dann gehe nochmal in das Modul-Verzeichnis, und Packe alle Module in eine *rpi-can-modules.tar.bz2* Datei:

```
cd $MODULES_TEMP
tar -cjf rpi-can-modules.tar.bz2 \
 lib/modules/3.12.34-ARCH/kernel/net/can/can.ko \
 lib/modules/3.12.34-ARCH/kernel/net/can/can-raw.ko \
 lib/modules/3.12.34-ARCH/kernel/net/can/can-gw.ko \
 lib/modules/3.12.34-ARCH/kernel/net/can/can-bcm.ko \
 lib/modules/3.12.34-ARCH/kernel/drivers/net/can/mcp251x.ko \
 lib/modules/3.12.34-ARCH/kernel/drivers/net/can/can-dev.ko \
 lib/modules/3.12.34-ARCH/kernel/drivers/spi/spi-bcm2708.ko \
 lib/modules/3.12.34-ARCH/extra/spi-config.ko
```

Fahre deinen Raspberry Pi nun herunter und nehme die Micro-SD Karte heraus. Das Boot-Verzeichnis, in dem der Kernel liegt, wird nun die Datei *rpi-can-modules.tar.bz2* sowie den neuen Kernel bekommen. Benenne die Datei *kernel.img* um in *kernel.img.orginal*.

Um den neuen Kernel zu nutzen, Wechsel in das tools/mkimage Verzechnis:

```
cd $BASEDIR/tools/mkimage
```

Führe dort folgendes aus:

```
./imagetool-uncompressed.py $KERNEL_SRC/arch/arm/boot/zImage
```

... und kopiere den neuen Kernel *kernel.img* auf die Micro-SD Karte.

Die Micro-SD Karte wieder unmounten/auswerfen und in den Raspberry Pi tun und Power an. Startet/Bootet der Raspberry Pi, müssen nun die Treiber entpackt und dann geladen werden (geht nur als root-User):

```
cd /
tar -xjf /boot/rpi-can-modules.tar.bz2

modprobe spi-bcm2708
modprobe can
modprobe can-dev
modprobe can-raw
modprobe can-bcm
```

Die SPI-Bus Konfiguration wird so eingespielt:

```
modprobe spi-config devices=bus=0:cs=0:modalias=mcp2515:speed=10000000:\
gpioirq=25:pd=20:pds32-0=16000000:pdu32-4=0x2002:force_release
```

Und zum Schluss wird der can-Bus als "Netzwerk" Interface aktiviert:

```
ip link set can0 up type can bitrate 500000
```

## Testen

Den Quellcode der Can-Utils findest du hier:

- https://gitorious.org/linux-can/can-utils

Am Besten auf dem **Raspberry Pi** mit *git* holen:

```
git clone https://gitorious.org/linux-can/can-utils.git
```

ins Verzeichnis wechseln:

```
cd can-utils
```

und mit *make* und *make install* installieren:

```
make
make install
```

Tipps zum Testen:

- http://skpang.co.uk/catalog/pican-canbus-board-for-raspberry-pi-p-1196.html

# Startprozess

Dieses Kapitel soll grob in die Tiefen des Archlinux Systems auf dem Raspberry Pi gehen, der die CAN-Bus Daten darstellen soll. Es beschreibt alle beteiligten Dateien, die *beim Betrieb* des Systems eine Rolle spielen.

## Tipp

Beim Benutzer *automat* ist im */home/automat/* Verzeichnis ein Link in den Ordner */var/opt/Display* hinterlegt. Sollte man die .h und .cpp Dateien ändern und mit *make* kompilieren, dann muss man diese Dateien nicht hin und her kopieren. Sehr praktisch. Man muss auch dafür nicht root sein.

# Ablauf des Bootvorgangs

Das System bootet einen Kernel und ein Boot-Image, was die nötigen Treiber und Anpassungen enthält, um auf den SPI-Bus Chip vom Pi und Canberry zugreift.

### Schritt 1

Nachdem der *NetworkManager.service* gestartet ist, wird der *canbus.service* gestartet. Mit dem Befehl

```
systemctl enable canbus.service
```

habe ich die Datei */usr/lib/systemd/system/canbus.service* dem Startvorgang bekannt gemacht. Dieser Befehl muss also in Zukunft nicht nochmal per Hand ausgeführt werden.

```
[Unit]
Description=CAN bus drivers and create CAN interface
After=NetworkManager.service

[Service]
ExecStart=/var/opt/Display/canbus.sh

[Install]
WantedBy=multi-user.target
```

Unter *ExecStart* sieht man das Skript (es muss Executeable sein!), welches die Befehle enthält, um die Treiber zu laden und den CAN-Bus als Netzwerk-Interface zu erstellen:

```
#!/bin/bash

modprobe spi-bcm2708
modprobe can
modprobe can-dev
modprobe can-raw
modprobe can-bcm
modprobe spi-config devices=bus=0:cs=0:modalias=mcp2515:speed=10000000:\
  gpioirq=25:pd=20:pds32-0=16000000:pdu32-4=0x2002:force_release
ip link set can0 up type can bitrate 500000
```

### Schritt 2

Wie in Schritt 1 habe ich die Datei *canWSSd.service* in den Bootvorgang eingebunden:

```
[Unit]
Description=can bus WebSocket Server Daemon (canWSSd)
```

```
After=canbus.service syslog.target

[Service]
User=automat
Group=automat
Type=forking
ExecStart=/var/opt/Display/canWSServer/canWSSd start can0 61000
ExecStop=/var/opt/Display/canWSServer/canWSSd stop
Restart=on-failure
RestartSec=6

[Install]
WantedBy=multi-user.target
```

Diese Datei sorgt dafür, dass der Can-Bus WebSocket Server Daemon nach dem *canbus.service* und der Systemprotokolierung (syslog.target) gestartet wird. Wie man leicht erkennen kann, wird mir dem *start* Parameter auch die Bezeichnung des Canbus Interface (can0 oder dummy) übergeben. Außerdem ist der Port (61000) angegeben, auf dem der Server auf Client-Anfragen wartet. Dieser Port muss mit dem Port übereinstimmen, den man beim Client (inkl. der IP-Adresse des Servers) angibt.

Wie man hoffentlich in der Datei sehen kann, wird bei einem Fehler des Startens von *canWSSd* nach 6 Sekunden ein weiterer Versuch unternommen (Schlimmstenfalls endlos oft). Das ist leider nötig, weil *canbus.service* eine paar Sekunden zum Erzeugen von *can0* braucht. Existiert *can0* nicht, dann versagt auch *canWSSd.service* und versucht es wie gesagt nach 6 sec erneut. Tragisch wir es, wenn es *canWSSd.service* nicht schafft, bevor *./tacho* (Schritt 4) gestartet wird. Dann stürzt leider der "Tacho" ab.

Bei Problemen ist folgender Befehl als root-User (su root) ein guter Wegbegleiter:

```
journalctl -r
```

Da aus Performance-Gründen in Archlinux nur selten Log-Dateien geschrieben werden, wir ein anderes Konzept genutzt, um Systemmeldungen auf zu zeichnen und darzustellen. Alle Meldungen, die *canWSSd* betreffen, sind mit dem obigen Befehl zu finden. Achtung: Da der Rasperry Pi die Uhr bei Power off stehen lässt, kann man den Zeiten und dem Datum in der Datei nicht ganz trauen! Das System ist aber so eingerichtet, dass über einen Ethernet Anschluss (wird via DHCP automatisch konfiguriert) aus dem Internet eine aktuelle Zeit geholt wird. Diese ist aber beim Booten noch nicht *da*.

Das "Type=forking" ist sehr wichtig. Da *canWSSd* ein fork() macht, um einen Prozess zu starten, der nebenher läuft und sich das "main.cpp" Programm beendet, muss man das angeben. Sonst würde das "systemd" Boot-System nicht erkennen, ob der Prozess wirklich korrekt läuft.

#### Hintergrund

Besonders interessant ist, dass *canWSSd* aus folgenden Teilen besteht:

- main.cpp : Das Daemon Konzept mit process() im Hintergrund und main() welches sich beendet.
- canWSS.cpp :
  - Horcht auf Client Anfragen
  - Schaut alle 0,5 Sekunden, ob es sich beenden soll
  - terminiert canThread und stellt ihn ggf. auf "dummy" Modus um
  - beobachtet canThread, und ließt neue Daten ein, falls diese ankommen
- canThread.cpp : Ist für das lesen der CAN-Bus Daten verantwortlich und enthält auch den "dummy" Modus.

### Schritt 3

#### Veraltet

Das System ist so eingerichtet, dass der Login-Manager *lightdm* den User "automat" automatisch einloggt.

#### Seit 12.3.2015

In der Datei */etc/systemd/system/getty.target.wants/getty@tty1.service* ist das Starten der tty-Login Terminals (strg + alt + F1 .. F6) so eingerichtet, dass der User *automat* sofort eingeloggt wird:

```
...
ExecStart=-/sbin/agetty --autologin automat --noclear %I $TERM
...
```

(–**autologin automat** wurde hinzugefügt)

### Schritt 4

#### Veraltet

Beim User "automat" ist im Ordner */home/automat/.config/autostart/* die Datei *tacho.desktop* abgelegt.

```
# auch gut fuer Startmenu/Sonstiges : ablegen dieser Datei
# in /usr/share/applications
# -> NUR DANN Type und Terminal einkommentieren !!!

[Desktop Entry]
Name=Tacho
Exec=/var/opt/Display/runTacho.sh
# Type=Applications
# Terminal=false
```

**Seit 12.3.2015**

Leider stimmt etwas nicht mit der *lxsession*, die eine Reihe von kleinen Programmen startet, welche eine grafische Oberfläche etwas "bedienbarer" machen. Daher wird nun auf einen grafischen Login-Manager wie *xdm-archlinux* oder *lightdm* verzichtet und wie oben erwähnt der User *automat* via Komandozeilen-Login automatisch eingeloggt. Beim Einloggen wird */home/automat/.bash_profile* gestartet:

```
#!/bin/bash

[[ -z $DISPLAY && $XDG_VTNR -eq 1 ]] && startx
```

Diese Zeile dafür sorgt, dass nur *ein* grafisches System mit *startx* gestartet wird. *startx* arbeitet dann die Datei *.xinitrc* ab:

```
#!/bin/bash

/var/opt/Display/runTacho.sh &
exec openbox
```

(mit *chmod +x /home/automat/.xinitrc* bitte die Datei ausführbar machen)

Das *exec openbox* sorgt dafür, dass die Fenster verschiebbar sind und ein rechts-Klick auf den Hintergrund ein Menu z.B. zum Logout öffnet. Die Zeile mit */var/opt/Display/runTacho.sh* sorgt dafür, dass automatisch beim Starten der grafischen Oberfläche auch das *Tacho*-Programm gestartet wird. Das Skript *runTacho.sh* ist etwas umständlich:

```
#!/bin/bash

cd /var/opt/Display/CanWSClient
./tacho
```

Leider gelang es mir in c++ nicht, dass *tacho* sein eigenes Verzeichnis erkennt und so in den Unter-Ordner *config/* zugreift. Daher muss man zum Starten von *tacho* erst in den Ordner von *tacho* wechseln. Das obige Bash-Skript erledigt das für uns (muss Executeable sein!).

Mit *x* kann man das *tacho* Fenster einfach schließen. Für einen neuer Start ist ein Link auf dem Desktop hinterlegt. Alternativ kann man auch einen Terminal öffnen und mit:

```
cd /home/automat/Display/CanWSClient/
```

oder

```
cd /var/opt/Display/CanWSClient/
```

oder

```
cd Display/CanWSClient/
```

in das Verzeichnis des Qt-Client wechseln und ihn mit *./tacho* von Hand starten. Dies hat 3 Vorteile:

- man sieht mögliche Fehlermeldungen
- man kann kontrollieren, ob die Konfiguration richtig erkannt wurde
- bei einem schnellen Abbruch: fehlt die Ausgabe, die sagt, dass sich tacho mit dem canWS Server verbunden hat? läuft canWSS?

Die Config-Datei des Qt-Client (/home/automat/Display/CanWSClient/Default.ini) ist hoffentlich selbsterklärend. Trotzdem ein paar Tipps:

- Die 3 Spalten MÜSSEN mit Tabulator (KEIN LEERFELD) getrennt sein
- Die Reihenfolge der Zeilen ist egal
- Die Bezeichnungs Spalte am Ende darf nicht verändert werden, da nach genau diesen Bezeichnungen in c++ gesucht wird. Eine Leerfeld am Ende der Zeile IST EINE ÄNDERUNG !
- nur am ENDE der Datei darf eine (oder mehr) leere Zeile stehen
- Die color Werte sind Hex-Werte. Beispiel: 0xff00ff10 bedeutet:
  - Rot 255
  - Grün 0
  - Blau 255
  - Transparenz 16

Bei den Drehzahlen bin ich mir bei der Berechung unsicher gewesen. Daher sind hier die Werte der 2 Bytes 1:1 von mir übernommen worden. Unklar war mir: Wenn das Steuergerät als max. Drehzahl 5000 für den Motor gesagt bekommt, aber der Motor 12000 kann, was bedeutet dann z.B. der Byte (Num-Wert in der Anleitung) 3276 ? Sind das 3276 RPM oder 500 RPM (10 % von 5000) oder 1200 RPM (10% von 12000) ? Im HTML-Client ist dies auch nicht von mir gelöst worden.

Bisher ist nur bei der Spannung von mir ein minimal-Wert in Verwendung. Es macht keinen Sinn, einen "Balken" von 0 V bis 552 V zu machen, da wohl bereits 402 V als "Akkus sind leer" definiert ist.

### Schritt 5

Wenn man mir CTRL+ALT+F2 und dann CTRL+ALT+ENTF den Pi ausschalten will, wird automatisch *canWSSd stop* ausgeführt. Dieser Befehl löscht die Datei

```
/tmp/canWSSd.run
```

Sie wir mit "start" angelegt und durch das Löschen erkennt im 0,5 Sekunden Takt der Prozess *canWSSd*, dass er sich beenden soll.

## HTML-Client

Neben dem Qt-Client */var/opt/Display/canWSClient/tacho* existieren auch 2 HTML/Javascript Clients. Diese haben ein paar Vorteile:

- Die HTML-Dateien laufen i.d.R. auf jedem PC, Smartphone und Tablet.
- Der nötige Code ist in der HTML-Datei und kann ohne Kompilieren angepasst werden.
- Er bietet mit einer Befehlszeile die Möglichkeit, allen anderen Clients (auch dem QT-Client) eine Nachricht zu senden
- Es macht deutlich, dass man über das Netzwerk / WLAN an alle CAN-Bus Daten kommt, und nur der Client für die Auswertung der Bytes verantwortlich ist.
- Er passt sich an die Breite des Browserfensters an.
- Mit *!dummy* als "Befehl" lässt sich der WebSocket Server in einen Demo-Modus schalten und mit dem selben Befehl auch wieder in den normalen Modus.

Leider ist der Raspberry Pi und der Web-Browser Midori aktuell nicht so leistungsstark, dass der HTML-Client brauchbar wäre.

/var/opt/Display/canWSClient/clientGui.html : ist eine Graphische Version mit einer SVG-Datei (integriert), an der mit Javascript Änderungen vorgenommen werden.

/var/opt/Display/canWSClient/clientRaw.html : ist eine abgespeckte Version, die einfach nur die Daten im *JSON* Format, so wie sie vom Server kommen, ausspuckt.

## Hinweise

Die Daten des Servers sind (Beispiel) so aufgebaut:

```
{"cobid": 385, "dlc": 4, "msg": "bitte langsamer fahren", "data": [48, 140, 216, 0] }
```

Bei den Zahlen handelt is sich NICHT um Hex-Werte! Um auch für andere CAN-Bus Systeme (trijekt ?) einen Client zu entwickeln, ist die REG-ID nicht explizit angegeben und entspricht dem ersten (0) Daten-Byte. Leider fehlt RTR des Kopfes. Bei Bedarf muss das jemand noch einbauen.

Die Befehlszeile im HTML-Client landet NICHT im CAN-Bus! Ein Senden in den CAN-Bus ist (bisher) nicht programmiert. Der Demo-Modus ist in der Datei *canThread.cpp* zu finden, falls man COB-ID und REG-ID oder die zufälligen Daten, die kommen sollen, anpassen will.

Sollte man am canWSServer arbeiten, so ist es sinnvoll als root den Dienst/Service zu stoppen:

```
systemctl stop canWSSd
```

Als *automat* User kann man den Daemon mit

```
./canWSSd start can0 61000
```

und

```
./canWSSd stop
```

dann “zu Fuß” starten und stoppen.

Aktuell sind von mir folgende REG-ID (und COB-IDs) in der Config des HTML-Client (Qt-Client) hinterlegt:

| COB-ID | REG-ID | Wert |
|---|---|---|
| 0x181 | 0x30 | IST-Wert Umdrehungen Links |
| 0x180 | 0x30 | IST-Wert Umdrehungen Rechts |
| 0x3FF | 0x81 | Geschwindigkeit |
| 0x3FF | 0x82 | Temperatur |
| 0x3FF | 0x83 | Spannung |

Die COBID 0x3FF ist von mir FREI erfunden! Aktuell weiss ich nicht, welches Gerät am Ende welchen Wert aussenden wird!

Es werden jeweils 2 Byte erwartet, die einen Wert von +/- 32767 annehemen können. Will man 480.3 V darstellen, sollten man dies mit dem Wert 4803 machen. Eine Anpassung im Code von tacho.cpp ist dafür notwendig!

Wir haben festgestellt, dass die REG-IDs 0x7a bis 0x8e von den Steuergeräten noch nicht benutzt sind und planen diese für unsere eigenen Parameter-Werte auf dem CAN-Bus ein.

# Der WebSocket Server Daemon

Ordner **/var/opt/Display/canWSServer/**

## canWSS.pro

```
QT += core websockets

TARGET = canWSSd

CONFIG  += console
CONFIG  -= app_bundle

TEMPLATE = app

SOURCES += \
    main.cpp \
    canWSS.cpp \
    canThread.cpp

HEADERS += \
    canWSS.h \
```

```
    canThread.h

MAKEFLAGS = -k

CFLAGS = -O2 -Wall
LDLIBS = -lpthread

CPPFLAGS += -Iinclude \
        -D_FILE_OFFSET_BITS=64 \
        -DSO_RXQ_OVFL=40 \
        -DPF_CAN=29 \
        -DAF_CAN=PF_CAN
```

### main.cpp

```
#include <iostream>
#include <cstring>
#include <QString>
#include <QtCore/QCoreApplication>
#include "canWSS.h"

#include <syslog.h>
#include <sys/types.h>
#include <sys/stat.h>

void process(int argc, char *argv[]) {
    QCoreApplication a(argc, argv);
    quint16 port = atoi(argv[3]);

    try {
        QString info = "Try to bind CANbus to WebSocket Server on port ";
        info += QString::number(port);
        syslog(LOG_NOTICE, info.toUtf8().constData());

        // create file runFile
        FILE *log = fopen(CanWSS_runFilename, "a+");
        if (!log) {
            syslog(LOG_ERR, "error to create .run File!");
            exit(EXIT_FAILURE);
        } else {
            fclose(log);
        }

        CanWSS *server = new CanWSS(port, argv[2]);
        QObject::connect(server, &CanWSS::closed, &a, &QCoreApplication::quit);
```

```
        a.exec();
        exit(EXIT_SUCCESS);
    } catch(const std::string e) {
        // error on starting or "runFilename" is missing -> stop daemon!
        syslog(LOG_ERR, e.c_str());
        exit(EXIT_FAILURE);
    }
}

int main(int argc, char *argv[]) {
    if(argc<2) {
        std::cout<<"usage: "<<argv[0]<<" stop "<<std::endl;
        std::cout<<"       "<<argv[0]<<" start "
        "<can interface|dummy> <listen port>"<<std::endl;
        exit(EXIT_FAILURE);
    }

    std::string action = argv[1];
    if(action == "start" && argc<4) {
        std::cout<<"usage: "<<argv[0]<<" stop "<<std::endl;
        std::cout<<"       "<<argv[0]<<" start "
        "<can interface|dummy> <listen port>"<<std::endl;
        exit(EXIT_FAILURE);
    }


    setlogmask(LOG_UPTO(LOG_INFO));
    openlog("canWSSd", LOG_CONS|LOG_NDELAY|LOG_PERROR|LOG_PID,LOG_USER);

    if(action == "stop") {
        QString info = "deleting file ";
        info += CanWSS_runFilename;
        info += " to stop daemon ...";
        syslog(LOG_INFO, info.toUtf8().constData());
        remove(CanWSS_runFilename);
        closelog();
        exit(EXIT_SUCCESS);
    }

    pid_t pid, sid;
    pid = fork();

    if (pid < 0) exit(EXIT_FAILURE);
    if (pid > 0) exit(EXIT_SUCCESS);

    umask(0);
```

```
    sid = setsid();
    if (sid < 0) exit(EXIT_FAILURE);

    close(STDIN_FILENO);
    close(STDOUT_FILENO);
    close(STDERR_FILENO);

    process(argc, argv);
    closelog();
    return EXIT_FAILURE;
}
```

### canThread.h

```
#ifndef CANTHREAD_H
#define CANTHREAD_H

#include <QThread>
#include <QObject>

#include <unistd.h>
#include <linux/can.h>
#include <linux/can/raw.h>

class CanThread : public QThread
{
    Q_OBJECT

public:
    bool isDummy;
    int cansock;
    struct can_frame frame;
    void run(void);

signals:
    void getMsg(void);
};
#endif
```

### canThread.cpp

```
#include <iostream>
#include <QTime>
#include "canThread.h"
```

```
using namespace std;

#define DUMMYDELAY_MS 50

void CanThread::run(void) {
    if(isDummy == true) cout<<"DUMMY-";
    cout << "can Thread started" << endl;

    QTime time = QTime::currentTime();
    qsrand((uint)time.msec());

    int16_t speed     =0;
    int16_t temperatur=0;
    int16_t rpm_l     =0;
    int16_t rpm_r     =0;
    int16_t voltage   =0;

    while(1) {
        if(isDummy == true) {
            msleep(DUMMYDELAY_MS);

            switch(qrand() % 5) {

            case 0:
                // speed
                frame.can_id = 0x3FF;
                frame.can_dlc = 3;
                frame.data[0] = 0x81;
                frame.data[1] = speed; // 32767=7FFF, -5=FFFB, -32768=8000
                frame.data[2] = 0x00;
                speed++; if(speed >130) speed =0;
                break;

            case 1:
                // temperatur
                frame.can_id = 0x3FF;
                frame.can_dlc = 3;
                frame.data[0] = 0x82;
                frame.data[1] = temperatur;
                frame.data[2] = 0x00;
                temperatur++; if(temperatur>100) temperatur=0;
                break;

            case 2:
                // batterie
                frame.can_id = 0x3FF;
```

```
            frame.can_dlc = 3;
            frame.data[0] = 0x83;
            frame.data[1] = voltage & 0x00FF;
            frame.data[2] = (voltage>>8) & 0x00FF;
            voltage+=5; if(voltage>560) voltage =0;
            break;

        case 3:
            // left RPM -> -100% = -32768
            //             +100% = +32767
            frame.can_id = 0x181;
            frame.can_dlc = 4;
            frame.data[0] = 0x30;
            frame.data[1] = rpm_l & 0x00FF;
            frame.data[2] = (rpm_l>>8) & 0x00FF;
            frame.data[3] = 0; // dummy byte
            rpm_l-=100; if(rpm_l < -32600) rpm_l = 0;
            break;

        case 4:
            // right RPM -> -100% = -32768
            //              +100% = +32767
            frame.can_id = 0x180;
            frame.can_dlc = 4;
            frame.data[0] = 0x30;
            frame.data[1] = rpm_r & 0x00FF;
            frame.data[2] = (rpm_r>>8) & 0x00FF;
            frame.data[3] = 0; // dummy byte
            rpm_r+=100; if(rpm_r > 32600) rpm_r = 0;
            break;

        default:
            // RANDOM other
            frame.can_id = qrand() % 2048; // 11bit
            unsigned leng = (qrand() % 5)+1;
            frame.can_dlc = leng;
            for(unsigned j = 0; j < leng; ++j) {
                frame.data[j] = qrand() % 256;
            }
            break;
        }
    } else {
        read(cansock, &frame, sizeof(frame));
    }
    emit getMsg();
}
```

```
}
```

### canWSS.h

```
#ifndef CANWSS_H
#define CANWSS_H

#include <QtCore/QObject>
#include <QtCore/QList>
#include <QtCore/QByteArray>

#include <cstdio>
#include <cstdlib>
#include <unistd.h>
#include <string>

#include <net/if.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/ioctl.h>

#include <linux/can.h>
#include <linux/can/raw.h>

#include "canThread.h"

QT_FORWARD_DECLARE_CLASS(QWebSocketServer)
QT_FORWARD_DECLARE_CLASS(QWebSocket)

#define CanWSS_runFilename "/tmp/canWSSd.run"

class CanWSS : public QObject {
    Q_OBJECT

public:
    explicit CanWSS(quint16 port,std::string canIf,QObject *parent=Q_NULLPTR);
    ~CanWSS();

Q_SIGNALS:
    void closed();

private Q_SLOTS:
    void onNewConnection();
    void socketDisconnected();
    void canMessage();
```

```
    void stopMeOrNot();
    void clientMessage(QString);

private:
    std::string _canIf;
    QString _msg;
    CanThread * _canThr;
    QWebSocketServer *m_pWebSocketServer;
    QList<QWebSocket *> m_clients;
};
#endif
```

### canWSS.cpp

```
#include "canWSS.h"
#include "QtWebSockets/qwebsocketserver.h"
#include "QtWebSockets/qwebsocket.h"
#include <QTimer>
#include <sstream>
#include <iostream>
#include <fstream>

#include "canThread.h"

#include <syslog.h>
#include <sys/types.h>
#include <sys/stat.h>

using namespace std;

QT_USE_NAMESPACE

CanWSS::CanWSS(quint16 port, string canIf, QObject *parent) :
        QObject(parent),
        m_pWebSocketServer(
            new QWebSocketServer(
            QStringLiteral("CAN WebSocket Server"),
            QWebSocketServer::NonSecureMode, this)
        ),
        m_clients()
{
    _canIf = canIf;
    _msg = "";

    if (m_pWebSocketServer->listen(QHostAddress::Any, port)) {
```

```
struct sockaddr_can addr;
struct ifreq ifr;

_canThr = new CanThread();
_canThr->cansock = -1;
_canThr->isDummy = false;

// websocket bindung
syslog(LOG_NOTICE, "Server is listening ...");
connect(
    m_pWebSocketServer,
    &QWebSocketServer::newConnection,
    this,
    &CanWSS::onNewConnection
);
connect(
    m_pWebSocketServer,
    &QWebSocketServer::closed,
    this,
    &CanWSS::closed
);

// Canbus bindung

if(_canIf == "dummy") {
    syslog(LOG_INFO, "using dummy canbus");
    _canThr->isDummy = true;
} else {

    if((_canThr->cansock = socket(PF_CAN, SOCK_RAW, CAN_RAW))<0){
        syslog(LOG_ERR, "Error while opening canbus socket!");
        exit(1);
    }

    snprintf(ifr.ifr_name, sizeof(ifr.ifr_name), canIf.c_str());
    ioctl(_canThr->cansock, SIOCGIFINDEX, &ifr);
    addr.can_family  = AF_CAN;
    addr.can_ifindex = ifr.ifr_ifindex;

    QString info = "Canbus ";
    info += canIf.c_str();
    info += " at index ";
    info += QString::number(ifr.ifr_ifindex);
    syslog(LOG_INFO, info.toUtf8().constData());
```

```
            if(bind(
                _canThr->cansock,
                (struct sockaddr *)&addr,
                sizeof(addr)
            ) < 0) {
                syslog(LOG_ERR, "Error in canbus socket bind");
                exit(2);
            }
        }

        connect(_canThr, SIGNAL(getMsg()), this, SLOT(canMessage()));
        _canThr->start();

        // check for stopping the daemon
        QTimer *timer = new QTimer(this);
        connect(timer, SIGNAL(timeout()), this, SLOT(stopMeOrNot()));
        timer->start(500);

    }else{
        throw (string) "portbind error - still in use?";
    }
}

/// sieht aus, als wuerde dies nie aufgerufen werden :-(
CanWSS::~CanWSS() {
    syslog(LOG_NOTICE, "Closing client connections ...");
    qDeleteAll(m_clients.begin(), m_clients.end());
    syslog(LOG_NOTICE, "Closing WebSocket Server ...");
    m_pWebSocketServer->close();
    syslog(LOG_NOTICE, "WebSocket Server closed");
}

void CanWSS::onNewConnection() {
    QWebSocket *pSocket = m_pWebSocketServer->nextPendingConnection();
    connect(
        pSocket,
        &QWebSocket::textMessageReceived,
        this,
        &CanWSS::clientMessage
    );
    connect(
        pSocket,
        &QWebSocket::disconnected,
        this,
        &CanWSS::socketDisconnected
    );
```

```
    m_clients << pSocket;
}

// toggle dummy mode by sending "dummy" from client!
void CanWSS::clientMessage(QString message) {
    _msg = message.toHtmlEscaped();
    QString temp = "Client sends: ";
    temp += _msg;
    syslog(LOG_NOTICE, temp.toUtf8().constData());

    if(message == "!dummy" && _canIf != "dummy") {
        if(_canThr->isDummy == true) {
            syslog(LOG_NOTICE, "toggle to canbus mode");
            _canThr->isDummy = false;
        } else {
            syslog(LOG_NOTICE, "toggle to DUMMY mode");
            _canThr->isDummy = true;
            // blockierendes lesen abbrechen im QThread!
            _canThr->terminate();
            _canThr->start();
        }
    }
}

void CanWSS::canMessage(void) {
    QString buffer = "";

    buffer += "{\"cobid\": "+QString::number(_canThr->frame.can_id);
    buffer += ", \"dlc\": "+QString::number(_canThr->frame.can_dlc);
    buffer += ", \"msg\": \""+ _msg +"\"";
    buffer += ", \"data\": [";
    unsigned i,leng;
    leng = _canThr->frame.can_dlc -1;
    for(i=0; i < leng; i++) {
        buffer += QString::number(_canThr->frame.data[i])+", ";
    }
    buffer += QString::number(_canThr->frame.data[i])+"] }";

    Q_FOREACH (QWebSocket *pClient, m_clients) {
        pClient->sendTextMessage(buffer);
    }
}

void CanWSS::stopMeOrNot(void) {
    ifstream ifile(CanWSS_runFilename);
    if(!ifile) {
```

```
        QString info = "Daemon stops because ";
        info += CanWSS_runFilename;
        info += " is missing";
        throw (string) info.toUtf8().constData();
    }
}

void CanWSS::socketDisconnected() {
    QWebSocket *pClient = qobject_cast<QWebSocket *>(sender());
    if (pClient) {
        m_clients.removeAll(pClient);
        pClient->deleteLater();
    }
}
```

### erstellen

```
qmake
make
```

## Der WebSocket Client

Ordner **/var/opt/Display/canWSClient/**

### config/Default.ini

```
VALUE                   str     DESCRIPTION/ID
1.0                     float   Refresh Distance
./config/Default.png    str     Image 1024x600
0xff0000dd              color   Speed
0xffffffdd              color   Motor Left
0xffffffdd              color   Motor Right
0xffaaffdd              color   Temperature
0x00ff00dd              color   Voltage
0xffffffff              color   Message
ws://localhost:61000    str     Websocket Server URL
0x3FF                   hex     cobid Speed
0x81                    hex     regid Speed
120                     int     max Speed
0x3FF                   hex     cobid Temperature
0x82                    hex     regid Temperature
90                      int     max Temperature
```

```
0x181                   hex     cobid Left RPM
0x30                    hex     regid Left RPM
-32767                  int     max Left RPM
0x180                   hex     cobid Right RPM
0x30                    hex     regid Right RPM
32767                   int     max Right RPM
0x3FF                   hex     cobid Voltage
0x83                    hex     regid Voltage
552                     int     max Voltage
404                     int     min Voltage
```

### clientGui.html

```
<!DOCTYPE html>
<html>
<head>
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <title>Hochschule Niederrhein Racing - HSNR</title>
<style>

body {background: #000000;}

* {
    margin: 0;
    padding: 0;
    color: #ffffff;
    font-family: Helvetica,Arial,sans-serif;
}

input[type=text] {
    color: #cccccc;
    background: #555555;
    margin: 5px;
    width: 60%;
    border: solid 1px #999999;
    padding: 2px;
    border-radius: 4px;
    height: 20px;
    line-height: 20px;
}

button {
    min-width: 90px;
    background-color: #f5f5f5;
```

```
    background-image: linear-gradient(to bottom, #ffffff, #e6e6e6);
    background-repeat: repeat-x;
    width: 15%;
    border-color: rgba(0, 0, 0, 0.1) rgba(0, 0, 0, 0.1) rgba(0, 0, 0, 0.25);
    color: #000000;
    text-shadow: 0 -1px 0 rgba(0, 0, 0, 0.25);
    -moz-border-bottom-colors: none;
    -moz-border-left-colors: none;
    -moz-border-right-colors: none;
    -moz-border-top-colors: none;
    vertical-align: middle;
    cursor: pointer;
    display: inline-block;
    font-size: 10px;
    line-height: 14px;
    margin-bottom: 0;
    padding: 4px 12px;
    text-align: center;
    border-radius: 4px;
    border-style: solid 1px;
    box-shadow: 0 1px 0 rgba(255,255,255,0.2) inset, 0 1px 2px rgba(0,0,0,0.05);
}

.prim {
    color: #ffffff;
    background-color: #006dcc;
    background-image: linear-gradient(to bottom, #0088cc, #0044cc);
}

.fullw {
    width: 90%;
    margin: auto;
}

.label {
    min-width: 75px;
    width: 15%;
    display: inline-block;
}

pre {
    font-family: fixed;
}

</style>
<script type="text/javascript">
```

```
var wsUri = "";

var maxSpeed = 120;
var maxTermo = 90;
var maxVoltage = 552;
var minVoltage = 404;
var maxRpmLeft  = -32767;
var maxRpmRight = +32767;

function elem(id) {
    return document.getElementById(id);
}

function toggleWSServer(obj) {
    if(obj.innerHTML == "disconnect") {
        websocket.close();
    } else {
        wsUri = elem("wsuri").value;
        websocket = new WebSocket(wsUri);
    }

    websocket.onopen = function (evt) {
        elem("connectBtn").innerHTML = "disconnect";
    };

    websocket.onclose = function (evt) {
        elem("connectBtn").innerHTML = "connect";
    };

    websocket.onmessage = function(evt) {
        onMessage(evt);
    };
}

function postMsg(obj) {
    if(websocket.OPEN) {
        websocket.send(elem("msg").value);
    }
}

function byte16ToInt(data) {
    var val = 256*data[2] + data[1];
    if(data[2] >= 0x80)
        val = -1.0 *(val ^ 0xFFFF) +1;
    return val;
```

```
}

var speed = 0;
var termo = 0;
var voltage = 0;
var rpmLeft = 0;
var rpmRight = 0;

function onMessage(evt) {
    serverData = JSON.parse(evt.data);

    if(serverData.data[0] == 0x81) { // regid
        var speedMaxAngle = 180;
        speed = byte16ToInt(serverData.data);
        var speedAngle = speed/maxSpeed * speedMaxAngle;
        var needle = elem("Needle").setAttribute(
            "transform",
            "rotate("+speedAngle+" 510 488)"
        );
        elem("valSpeed").innerHTML = speed;
    }

    if(serverData.data[0] == 0x82) { // regid
        var termoMaxHeight = 591;
        termo = byte16ToInt(serverData.data);
        var termoHeight = termo/maxTermo * termoMaxHeight;
        elem("Termo").setAttribute("height", termoHeight);
        elem("Termo").setAttribute("y", 7 + termoMaxHeight - termoHeight);
        elem("valTermo").innerHTML = termo;
    }

    if(serverData.data[0] == 0x83) { // regid
        var voltageMaxHeight = 591;
        voltage = byte16ToInt(serverData.data);
var voltageHeight = voltageMaxHeight * (voltage - minVoltage)/(maxVoltage - minVoltage);
        elem("Power").setAttribute("height", voltageHeight);
        elem("Power").setAttribute("y", 7 + voltageMaxHeight - voltageHeight);
        elem("valVoltage").innerHTML = voltage;
    }

    if(serverData.cobid == 0x181) {
        if(serverData.data[0] == 0x30) { // regid
            var leftMaxAngle = -60;
            rpmLeft = byte16ToInt(serverData.data);
            var leftAngle = rpmLeft/maxRpmLeft * leftMaxAngle;
            elem("LeftMot").setAttribute(
```

```
                "transform",
                "rotate("+leftAngle+" 190 488)"
            );
            elem("valLeft").innerHTML = rpmLeft;
        }
    }

    if(serverData.cobid == 0x180) {
        if(serverData.data[0] == 0x30) { // regid
            var rightMaxAngle = 60;
            rpmRight = byte16ToInt(serverData.data);
            var rightAngle = rpmRight/maxRpmRight * rightMaxAngle;
            elem("RightMot").setAttribute(
                "transform",
                "rotate("+rightAngle+" 827 488)"
            );
            elem("valRight").innerHTML = rpmRight;
        }
    }

    elem("textValue").innerHTML = serverData.msg;
}

</script>
</head>

<body>

<form class="fullw" action="#">
    <span class="label">Server:</span>
    <input type="text" id="wsuri" value="ws://localhost:61000/">
    <button type="submit" id="connectBtn"
     onClick="javascript:toggleWSServer(this);">connect</button>
    <br />
    <span class="label">Befehl:</span> <input type="text" id="msg">
    <button type="submit" id="sendBtn" class="prim"
     onClick="javascript:postMsg(this);">post</button>
</form>

<div class="fullw">
<svg ... >... SVG-BILD !!!! ...</svg>
</div>

<hr />

<pre>
```

```
Speed:    <span id="valSpeed"></span> km/h
Termo:    <span id="valTermo"></span> &deg; C (Baterie ? Kühlung ?)
Voltage:  <span id="valVoltage"></span> V
Ampere:   <span id="valAmpere"></span> A (todo ?)
Left:     <span id="valLeft"></span> RPM (todo ?)
Right:    <span id="valRight"></span> RPM (todo ?)
</pre>

</body>
</html>
```

### tacho.pro

```
QT += widgets websockets

HEADERS     = tacho.h
SOURCES     = tacho.cpp \
              main.cpp

MAKEFLAGS = -k

CFLAGS = -O2 -Wall
LDLIBS = -lpthread

CPPFLAGS += -Iinclude \
          -D_FILE_OFFSET_BITS=64 \
          -DSO_RXQ_OVFL=40 \
          -DPF_CAN=29 \
          -DAF_CAN=PF_CAN
```

### main.cpp

```
#include <QApplication>
#include "tacho.h"
#define DEFAULTINI "./config/Default.ini"

using namespace std;

int main(int argc, char *argv[]) {
    QApplication app(argc, argv);

    Tacho *tach = new Tacho(DEFAULTINI, NULL);
    QObject::connect(tach, &Tacho::closed, &app, &QApplication::quit);
```

```
    tach->show();
    return app.exec();
}
```

### tacho.h

```
#ifndef TACHO_H
#define TACHO_H

#include <QWidget>
#include <QString>
#include <QJsonObject>
#include <QJsonArray>

#include <cstdio>
#include <cstdlib>
#include <unistd.h>
#include <string>
#include <map>

#include <QtCore/QObject>
#include <QtWebSockets/QWebSocket>

#define BUFSIZE 512

typedef struct{
    std::map<std::string, std::string> strMap;
    std::map<std::string, int> intMap;
    std::map<std::string, int> hexMap;
    std::map<std::string, QColor> colorMap;
    std::map<std::string, float> floatMap;
}tachoConf;

class Tacho : public QWidget {
    Q_OBJECT
    public:
        Tacho(const QString &filename, QWidget *parent);
        ~Tacho(void);

    Q_SIGNALS:
        void closed(void);

    private Q_SLOTS:
        void onConnected(void);
        void onTextMessageReceived(QString message);
```

```
    protected:
        void configReader(const QString & filename);
        int byte16ToInt(QJsonArray data);
        void paintEvent(QPaintEvent *event);
        void readValue(void);

        void showMessage(void);
        void showPower(void);
        void showTermo(void);
        void showLeft(void);
        void showRight(void);

        void showValue(void);

        QColor hexToColor(std::string hex);

    private:
        int _value;
        int _left;
        int _right;
        int _termo;
        int _power;

        float _newValue;
        float _newLeft;
        float _newRight;
        float _newTermo;
        float _newPower;

        QJsonObject _jdata;
        tachoConf _config;

        QWebSocket m_webSocket;
        QUrl m_url;

};
#endif
```

### tacho.cpp

```
#include <QtGui>
#include <QStaticText>
#include <QFile>
#include <QTime>
```

```
#include <sstream>
#include <QJsonDocument>
#include <QJsonArray>
#include <iostream>
#include <QUrl>

#include "QtWebSockets/qwebsocketserver.h"
#include "QtWebSockets/qwebsocket.h"
#include <QtCore/QDebug>

#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#include <cstring>
#include <cstdlib>

#include "tacho.h"

QT_USE_NAMESPACE

QColor Tacho::hexToColor(std::string hex) {
    int red   = strtol(hex.substr(2, 2).c_str(), NULL, 16);
    int green = strtol(hex.substr(4, 2).c_str(), NULL, 16);
    int blue  = strtol(hex.substr(6, 2).c_str(), NULL, 16);
    int alpha = strtol(hex.substr(8, 2).c_str(), NULL, 16);
    return QColor(red, green, blue, alpha);
}

void Tacho::configReader(const QString & filename) {

    QFile file(filename);

    if (file.open(QFile::ReadOnly)) {
        char buff[BUFSIZE];
        std::string strBuff, cvalue, cid;
        qint64 lineLength;
        int pos1, pos2;
        do{
            lineLength = file.readLine(buff, sizeof(buff));
            if(lineLength < 4) break;

            strBuff = buff;

            pos1 = strBuff.find('\t');
            cvalue = strBuff.substr(0, pos1);
```

```
            pos2 = strBuff.find_last_of('\t');
            cid = strBuff.substr(pos2+1);
            pos1 = cid.find_last_not_of('\n');
            cid = cid.substr(0, pos1+1);

            if(strBuff.find("\tint\t") != std::string::npos) {
                _config.intMap[cid] = strtol(cvalue.c_str(), NULL, 10);
                std::cout << "int:   " << _config.intMap[cid];
                std::cout << " -> "<< cid << std::endl;
            } else if(strBuff.find("\tcolor\t") != std::string::npos) {
                _config.colorMap[cid] = hexToColor(cvalue);
                std::cout << "color: " << cvalue;
                std::cout << " -> "<< cid << std::endl;
            } else if(strBuff.find("\tfloat\t") != std::string::npos) {
                _config.floatMap[cid] = atof(cvalue.c_str());
                std::cout << "float: " << _config.floatMap[cid];
                std::cout << " -> "<< cid << std::endl;
            } else if(strBuff.find("\thex\t") != std::string::npos) {
                _config.hexMap[cid] = strtol(
                    cvalue.substr(2, 3).c_str(), NULL, 16
                );
                std::cout << "hex:   " << _config.hexMap[cid];
                std::cout << " -> "<< cid << std::endl;
            } else {
                _config.strMap[cid] = cvalue;
                std::cout << "str:   " << _config.strMap[cid];
                std::cout << " -> "<< cid << std::endl;
            }
        }while(lineLength > 3);
    }
    m_url = QUrl(_config.strMap["Websocket Server URL"].c_str());
}

Tacho::Tacho(const QString &filename, QWidget *parent) : QWidget(parent) {
    configReader(filename);

    connect(&m_webSocket, &QWebSocket::connected, this, &Tacho::onConnected);
    connect(&m_webSocket, &QWebSocket::disconnected, this, &Tacho::closed);
    m_webSocket.open(m_url);

    _value = 0;
    _left = 0;
    _right = 0;
    _termo = 0;
    _power = 0;
```

```
    _newValue = 0;
    _newLeft  = 0;
    _newRight = 0;
    _newTermo = 0;
    _newPower = 0;

    QPalette * palette = new QPalette();
    QPixmap image(_config.strMap["Image 1024x600"].c_str());
    palette->setBrush(QPalette::Background,*(new QBrush(image)));
    setPalette(*palette);

    setWindowTitle(tr("Hochschule Niederrhein Racing - HSNR"));
    setFixedSize(image.width(), image.height());
}

Tacho::~Tacho(void) {
    m_webSocket.close();
}

void Tacho::onConnected(void) {
    qDebug() << "WebSocket connected";
    connect(
        &m_webSocket,
        &QWebSocket::textMessageReceived,
        this,
        &Tacho::onTextMessageReceived
    );
    m_webSocket.sendTextMessage(QStringLiteral("Displ On"));
}

void Tacho::onTextMessageReceived(QString json) {
    QJsonDocument document = QJsonDocument::fromJson(json.toUtf8());
    _jdata = document.object();
    //qDebug() << json;
    update();
}

int Tacho::byte16ToInt(QJsonArray data) {
    int16_t val = 256 * data[2].toInt() + data[1].toInt();
    if(data[2].toInt() >= 0x80)
        val = -1.0 *(val ^ 0xFFFF) +1;
    return val;
}

void Tacho::readValue(void) {
```

```
// only read every X times vor a smooth zeiger movement
static float times = _config.floatMap["Refresh Distance"];
static float count = 0;
int cobid;
int regid;

// only read every X times
if(count>=times) {
    _value = _newValue;
    _left  = _newLeft;
    _right = _newRight;
    _termo = _newTermo;
    _power = _newPower;

    QJsonArray data = _jdata["data"].toArray();

    /// @todo: muss ein switch case oder so werden !!!!!

    cobid = _jdata["cobid"].toInt();
    regid = data[0].toInt();

    if(
        cobid == _config.hexMap["cobid Speed"] &&
        regid == _config.hexMap["regid Speed"]
    ){
        _newValue = byte16ToInt(data);
    }else if (
        cobid == _config.hexMap["cobid Temperature"] &&
        regid == _config.hexMap["regid Temperature"]
    ){
        _newTermo = byte16ToInt(data);
    }else if(
        cobid == _config.hexMap["cobid Left RPM"] &&
        regid == _config.hexMap["regid Left RPM"]
    ){
        _newLeft = byte16ToInt(data);
    }else if(
        cobid == _config.hexMap["cobid Right RPM"] &&
        regid == _config.hexMap["regid Right RPM"]
    ){
        _newRight = byte16ToInt(data);
    }else if(
        cobid == _config.hexMap["cobid Voltage"] &&
        regid == _config.hexMap["regid Voltage"]
    ){
        _newPower = byte16ToInt(data);
```

```
        }else{
            //qDebug() << "unknown cobid";
            ;
        }

        count = 0;
    }
    // for a sooth movement of the zeiger !!
    count += 1.0;

    _value = _value + count * (_newValue - _value)/times;
    _left  =  _left + count * (_newLeft  - _left )/times;
    _right = _right + count * (_newRight - _right)/times;
    _termo = _termo + count * (_newTermo - _termo)/times;
    _power = _power + count * (_newPower - _power)/times;
}

void Tacho::showMessage(void) {
    QPainter painter(this);
    painter.setRenderHint(QPainter::Antialiasing);

    // PIXEL
    painter.translate(130, height() - 45);

    painter.setPen(_config.colorMap["Message"]);
    QStaticText txt = QStaticText(_jdata["msg"].toString());

    painter.setFont(QFont("Sans Serif", 25));
    painter.drawStaticText(0, 0, txt);

    painter.save();
    painter.restore();
}

void Tacho::showPower(void) {
    QPainter painter(this);
    painter.setRenderHint(QPainter::Antialiasing);

    // PIXEL
    painter.translate(959, this->height());

    painter.setPen(Qt::NoPen);
    painter.setBrush(_config.colorMap["Voltage"]);

    // PIXEL
    int heightPower = (_power - _config.intMap["min Voltage"]);
```

```
    heightPower    *= ((float) this->height() - 18.0);
    heightPower    *= 1.0/(
        _config.intMap["max Voltage"] - _config.intMap["min Voltage"]
    );
    painter.drawRect(2, -3, 61, -1 * heightPower);

    painter.setPen(_config.colorMap["Voltage"]);
    QStaticText txt = QStaticText(QString::number(_power) + " Volt");
    painter.setFont(QFont("Sans Serif", 18));
    // PIXEL
    painter.drawStaticText(-150, -1 * this->height() +20, txt);

    painter.save();
    painter.restore();
}

void Tacho::showTermo(void) {
    QPainter painter(this);
    painter.setRenderHint(QPainter::Antialiasing);

    // PIXEL
    painter.translate(0, this->height());

    painter.setPen(Qt::NoPen);
    painter.setBrush(_config.colorMap["Temperature"]);

    // PIXEL
    int heightTermo = _termo * (
        (float) this->height()- 18.0
        )/_config.intMap["max Temperature"];
    painter.drawRect(2, -2, 59, -1 * heightTermo);

    painter.setPen(_config.colorMap["Temperature"]);
    QStaticText txt = QStaticText(QString::number(_termo) + " Celsius");
    painter.setFont(QFont("Sans Serif", 18));
    // PIXEL
    painter.drawStaticText(120, -1 * this->height() +20, txt);

    painter.save();
    painter.restore();
}

void Tacho::showLeft(void) {
    QPainter painter(this);
    painter.setRenderHint(QPainter::Antialiasing);
```

```
    // PIXEL
    painter.translate(191, 484);

    painter.setPen(_config.colorMap["Motor Left"]);
    QStaticText txt = QStaticText(QString::number(_left));
    painter.setFont(QFont("Sans Serif", 18));
    // PIXEL
    painter.drawStaticText(100, 8, txt);

    // der Zeiger ist ein 4eck
    static const QPoint zeigerHand[4] = {
        QPoint(0, 6),
        QPoint(314, 2),
        QPoint(314, -2),
        QPoint(0, -6)
    };

    painter.setPen(Qt::NoPen);
    painter.setBrush(_config.colorMap["Motor Left"]);
    painter.rotate(-60.0 * _left/_config.intMap["max Left RPM"]);
    painter.drawConvexPolygon(zeigerHand, 4);

    painter.save();
    painter.restore();
}

void Tacho::showRight(void) {
    QPainter painter(this);
    painter.setRenderHint(QPainter::Antialiasing);

    // PIXEL
    painter.translate(830, 484);

    painter.setPen(_config.colorMap["Motor Right"]);
    QStaticText txt = QStaticText(QString::number(_right));
    painter.setFont(QFont("Sans Serif", 18));
    // PIXEL
    painter.drawStaticText(-190, 8, txt);

    // der Zeiger ist ein 4eck
    static const QPoint zeigerHand[4] = {
        QPoint(0, 6),
        QPoint(-314, 2),
        QPoint(-314, -2),
        QPoint(0, -6)
    };
```

```
    painter.setPen(Qt::NoPen);
    painter.setBrush(_config.colorMap["Motor Right"]);
    painter.rotate(60.0 * _right/_config.intMap["max Right RPM"]);
    painter.drawConvexPolygon(zeigerHand, 4);

    painter.save();
    painter.restore();
}

void Tacho::showValue(void) {
    QPainter painter(this);
    painter.setRenderHint(QPainter::Antialiasing);

    // PIXEL
    painter.translate(width() / 2, height()-115);

    painter.setPen(_config.colorMap["Speed"]);
    QStaticText txt = QStaticText(QString::number(_value) + " km/h");
    painter.setFont(QFont("Sans Serif", 18));
    // PIXEL
    painter.drawStaticText(-50, -300, txt);

    painter.rotate(270.0);

    // der Zeiger ist ein Dreieck
    static const QPoint zeigerHand[3] = {
        QPoint(20, -321),
        QPoint(-20, -321),
        QPoint(0, -437)
    };

    painter.setPen(Qt::NoPen);
    painter.setBrush(_config.colorMap["Speed"]);
    painter.rotate(180.0 * _value/_config.intMap["max Speed"]);
    painter.drawConvexPolygon(zeigerHand, 3);

    painter.save();
    painter.restore();
}

void Tacho::paintEvent(QPaintEvent *) {
    readValue();
    showMessage();
    showLeft();
    showRight();
```

```
    showTermo();
    showPower();
    showValue();
}
```

### erstellen

```
qmake
make
```