CANWIFI Developer Guide

Table of Contents

1 Acronyms

TCP – Transport Control Protocol

CANBUS -

CBUS- Merg CANBUS implementation

IP – Internet Protocol

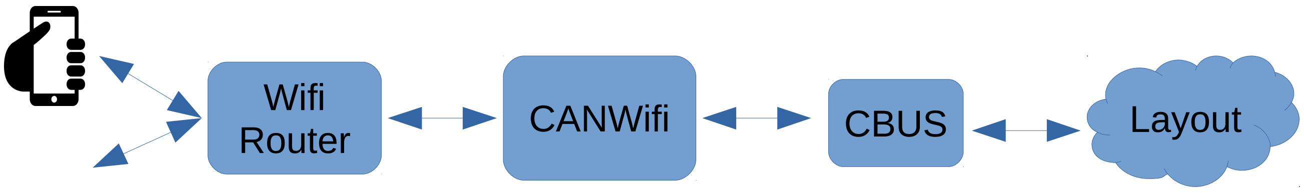
DHCP – Dynamic Host Configuration Protocol

2 Introduction

The canwifi module is a linux based module built to run on raspberry pi, but it can be expanded to any linux system. Its main puRaspberry Piose is to translate the messages from Engine Driver (ED) to CBUS module without using JMRI. It communicates with ED using TCP and with CBUS using CANSocket.

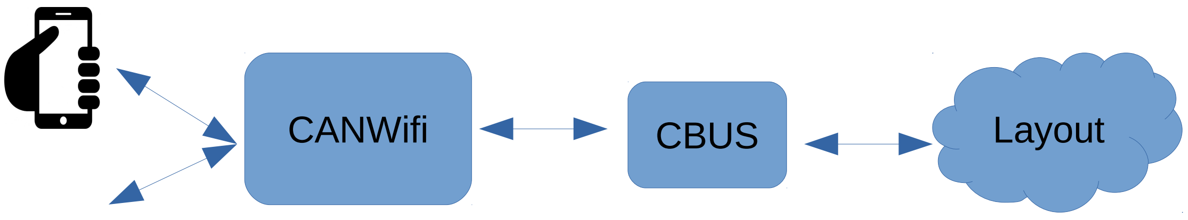
The CANWifi supports to modes: wifi mode and AP (Access Point) mode.

Below is an architecture overview on Wifi mode.



In this configuration the CANWifi behaves as a client of an existing IP network. The smaRaspberry Pihone connects to the wifi router and receives an IP address (ip1). The CANWifi also connects to the wifi router and receives an IP address (ip2). The CANWifi then starts the canwifi service and announces it via bonjour protocol. When the user starts the ED it should see the available connection and it will include the CANWifi. The user can connect to the CANWifi and drive the locos normally.

When the CANWifi is in AP mode the architecture is the following:



In this configuration the CANWifi will work as an Access Point, it means that it will allow any wifi device to connect to it. It will assign and IP to the connected device via dhcp. The CANWifi itself will have a fixed ip address (192.168.45.1). In order to ED control the layout the smartphone has to connect to the CANWifi, receive an IP and connect to the service announced by the bonjour protocol.

3 Knowledge requirements

Some TCP IP knowledge, linux commands and Merg CBUS are desirable to understand some terms.

The following internet pages can provide more information:

TCP IP

http://www.thegeekstuff.com/2011/11/tcp-ip-fundamentals/

http://basicnetworkingconcepts.blogspot.co.nz/2011/02/what-is-tcpip.html

Linux concepts and commands

http://linuxcommand.org/learning_the_shell.php

http://www.tldp.org/LDP/gs/node5.html

Merg CBUS

http://www.merg.org.uk/merg_resources/cbus.php

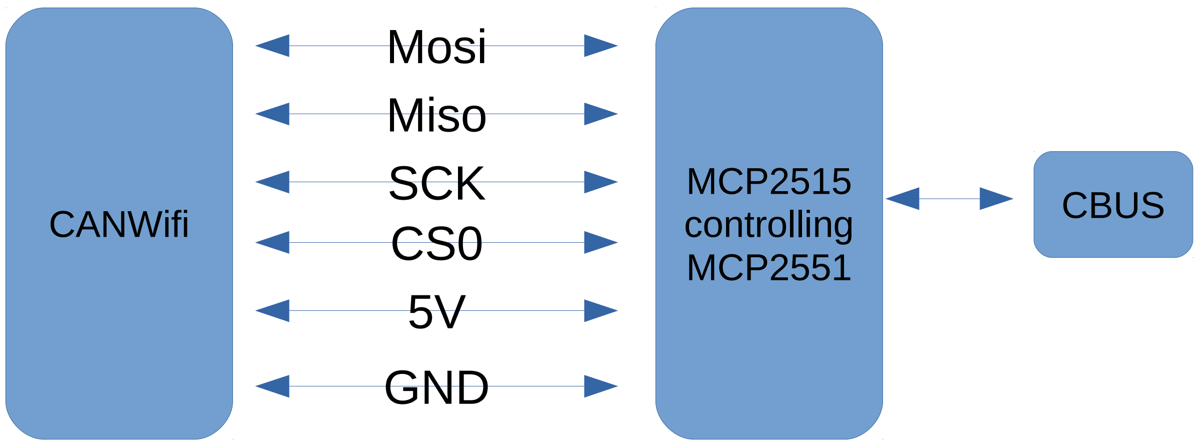
http://www.merg.org.uk/merg_resources/cbus-dcc.php

4 Physical components

- Raspberry Pi
- Can controller and transceiver
- Wifi dongle

The Raspberry Pi runs jessie-lite, a debian based linux distribution officially supported by the Raspberry Pi foundation. It comes with support for CAN Socket and has the drivers for the CAN Controller MCP2515. The communication with MCP2515 is done by SPI using the SPI pins from Raspberry Pi.

Below is a diagram showing the logical connections:



5 Initial Setup

For a new developer or tester some steps are required to have the CANWifi working:

First you need to create a Raspberry Pi ssid card to boot the Raspberry Pi. You can follow the oficial Raspberry Pi documentation in https://www.raspberrypi.org/ or follow one of the several tutorials in internet. It also necessary to have the access to internet to have the project source code and cofiguration scripts.

Second we need to setup the basic components. There is 2 options here: automatic or manual setup.

5.1 Automatic setup

For automatic setup, you need to create a linux bash script for that or download the existing one from https://github.com/amaurial/canpi/blob/master/initial-setup.sh.

To create manually, open a terminal or use a text editor, create a file called **initial-setup.sh** inside the Raspberry Pi and copy the commands in the Appendix A the file.

Make the file executable by typing this command the linux terminal.

chmod +x initial-setup.sh

To run the scrip run

sudo ./initial-setup.sh

This will install all the software components, compile the project and setup the CANWifi in wifi mode. The script will ask the credential to connect to your wifi router.

5.2 Manual setup

Install git, isc-dhcp-server, hostapd and avahi by typing in a command line:

sudo apt-get update

sudo apt-get install git

sudo apt-get install isc-dhcp-server

sudo apt-get install hostapd

sudo apt-get install   avahi

git clone https://github.com/amaurial/canpi.git

cd canpi

sudo cp start_canpi.sh /etc/init.d

sudo chmod +x /etc/init.d/start_canpi.sh

Edit the file canpi.cfg and make sure you set these properties properly:

```
ap_mode="False"

router_ssid="your ssid"

router_password="your ssid password"
```

Now execute

make clean

make all

sudo ./install_webpy.sh

sudo /etc/init.d/start_canpi.sh/configure

The last command will read the contents of the canpi.cfg and do the proper setup of the CANWifi. See canpi daemon.

6 Sofware components

The solution consists of several linux components and a software developed to handle the ED clients and translate the messages to CBUS.

The components are:

1. linux tcp ip stack (including the CAN Socket)
2. avahi daemon (service announcement – bonjour protocol)
3. wpa_supplicant (wifi handler)
4. isc-dhcp-server (dhcp server used when in AP mode)
5. hostapd daemon
6. canpi daemon (executable software with configuration scripts)

7. webpy (simple web browser)

6.1 Linux TCP IP stack

The linux tcp stack is a robust and well tested stack that runs on millions of devices. The major configuration is done by editing the file **/etc/network/interfaces**.

When setting up a CANWifi the developer should first do some configuration in order to get the wifi dongle and/or the ethernet interface working.

Here is a typical configuration:

```
# interfaces(5) file used by ifup(8) and ifdown(8)
# Please note that this file is written to be used with dhcpcd
# For static IP, consult /etc/dhcpcd.conf and 'man dhcpcd.conf'
# Include files from /etc/network/interfaces.d:
source-directory /etc/network/interfaces.d

auto lo
iface lo inet loopback

auto eth0
iface eth0 inet dhcp

allow-hotplug wlan0
iface wlan0 inet dhcp
    wpa-conf /etc/wpa_supplicant/wpa_supplicant.conf

auto can0
iface can0 can static
      bitrate 125000 restart-ms 1000
```

In this configuration there are 4 interfaces: loopback or lo, eth0, wlan0 and can0.

The lo, eth0 and wlan0 are auto configured and the IP address will assigned using the dhcp protocol.

The wlan0 uses the wpa_supplicant utility to configure and access the wifi network.

For more information see:

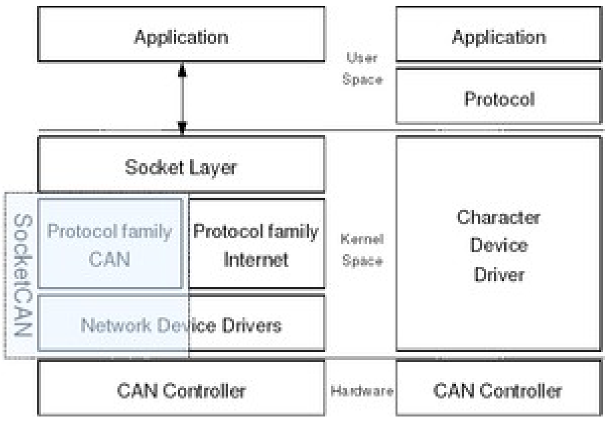
http://www.tldp.org/HOWTO/NET3-4-HOWTO-5.html

http://www.tldp.org/LDP/nag2/nag2.pdf

The can0 interface uses the new SocketCAN support added by Volkswages to linux kernel. Basically it allow the communication to a CANBUS by using linux sockets, like the usaged of tcp sockets.

Extracted from https://en.wikipedia.org/wiki/SocketCAN

“**SocketCAN** is a set of open source CAN drivers and a networking stack contributed by Volkswagen Research to the Linux kernel. Formerly known as Low Level CAN Framework (LLCF).



Typical CAN communication layers. With SocketCAN (left) or conventional (right).

Traditional CAN drivers for Linux are based on the model of character devices. Typically they only allow sending to and receiving from the CAN controller. Conventional implementations of this class of device driver only allow a single process to access the device, which means that all other processes are blocked in the meantime. In addition, these drivers typically all differ slightly in the interface presented to the application, stifling portability. The SocketCAN concept on the other hand uses the model of network devices, which allows multiple applications to access one CAN device simultaneously. Also, a single application is able to access multiple CAN networks in parallel.

The SocketCAN concept extends the Berkeley sockets API in Linux by introducing a new protocol family, PF_CAN, that coexists with other protocol families like PF_INET for the Internet Protocol. The communication with the CAN bus is therefore done analogously to the use of the Internet Protocol via sockets. Fundamental components of SocketCAN are the network device drivers for different CAN controllers and the implementation of the CAN protocol family. The protocol family, PF_CAN, provides the structures to enable different protocols on the bus: Raw sockets for direct CAN communication and transport protocols for point-to-point connections. Moreover the broadcast manager which is part of the CAN protocol family provides functions e.g. for sending CAN messages periodically or realize complex message filters.

Patches about CAN were added in the 2.6.25 Linux kernel. Meanwhile some controller drivers were added and work is going on to add drivers for a variety of controllers.“

For more information about SocketCAN see https://en.wikipedia.org/wiki/SocketCAN

6.1.1 Useful commands and files

To start or stop the networking services do:

```
sudo services networking start
sudo services networking stop
```

To start, stop or configure an interface use:
sudo ip link set wlan0 down
sudo ip link set wlan0 up

For more information on the command **ip** and **ifconfig** type this on command line:
man ip
man ifconfig

6.2 Avahi daemon

From Wikipedia (https://en.wikipedia.org/wiki/Avahi_(software)):

"**Avahi** is a free zero-configuration networking (zeroconf) implementation, including a system for multicast DNS/DNS-SD service discovery. It is licensed under the GNU Lesser General Public License (LGPL).
Avahi is a system which enables programs to publish and discover services and hosts running on a local network. For example, a user can plug their computer into a network and have Avahi automatically advertise the network services running on the machine which could enable access to files and printers."

The CANWifi configures the avahi to announce the canpi service expected by the ED. When ED starts it looks for services identified by "**_withrottle._tcp**". See http://jmri.sourceforge.net/help/en/html/doc/Technical/Networking.shtml

6.2.1 Useful commands and files

The configuration files are:

/etc/avahi/services/multiple.services

To manually start or stop the service use the commands:

sudo services avahi start

sudo services avahi stop

6.3 wpa_supplicant

From Wikipedia (https://en.wikipedia.org/wiki/Wpa_supplicant):

"**wpa_supplicant** is a free software implementation of an IEEE 802.11i supplicant for Linux, FreeBSD, NetBSD, QNX, AROS, Microsoft Windows, Solaris, OS/2 (including eComStation) and Haiku. In addition to being a fully featured WPA2 supplicant, it also implements WPA and older wireless LAN security protocols. Features include:
- WPA and full IEEE 802.11i/RSN/WPA2
- WPA-PSK and WPA2-PSK ("WPA-Personal", pre-shared key)

•WPA with EAP ("WPA-EnteRaspberry Pirise", for example with RADIUS authentication server)
•key management for CCMP, TKIP, WEP (both 104/128- and 40/64-bit)
•RSN: PMKSA caching, pre-authentication
•IEEE 802.11r
•IEEE 802.11w
•Wi-Fi Protected Setup (WPS)
Included with the supplicant are a GUI and a command-line utility for interacting with the running supplicant. From either of these interfaces it is possible to review a list of currently visible networks, select one of them, provide any additional security information needed to authenticate with the network (for example, a passphrase, or username and password) and add it to the preference list to enable automatic reconnection in the future.[2]“

The CANWifi make basic configuration on wpa supplicant just to inform the wifi ssid and the password when in Wifi mode.

6.3.1 Useful commands and files

The configuration files are:

/etc/wpa_supplicant/wpa_supplicant.conf

6.4 isc-dhcp-server

The Dynamic Host Configuration Protocol (DHCP) is a network service that enables host computers to be automatically assigned settings from a server as opposed to manually configuring each network host. Computers configured to be DHCP clients have no control over the settings they receive from the DHCP server, and the configuration is transparent to the computer's user.

The CANWifi uses the dhcp server to assign IP addresses to clients connected when working on AP mode.

This page provides all information about installing and configuring the dhcp server

https://wiki.debian.org/DHCP_Server

6.4.1 Useful commands and files

To manually start and stop the dhcp service do

sudo services isc-dhcp-server start

sudo services isc-dhcp-server stop

The configuration files are

/etc/default/isc-dhcp-server

/etc/dhcp/dhcpd.conf

6.5 Hostapd daemon

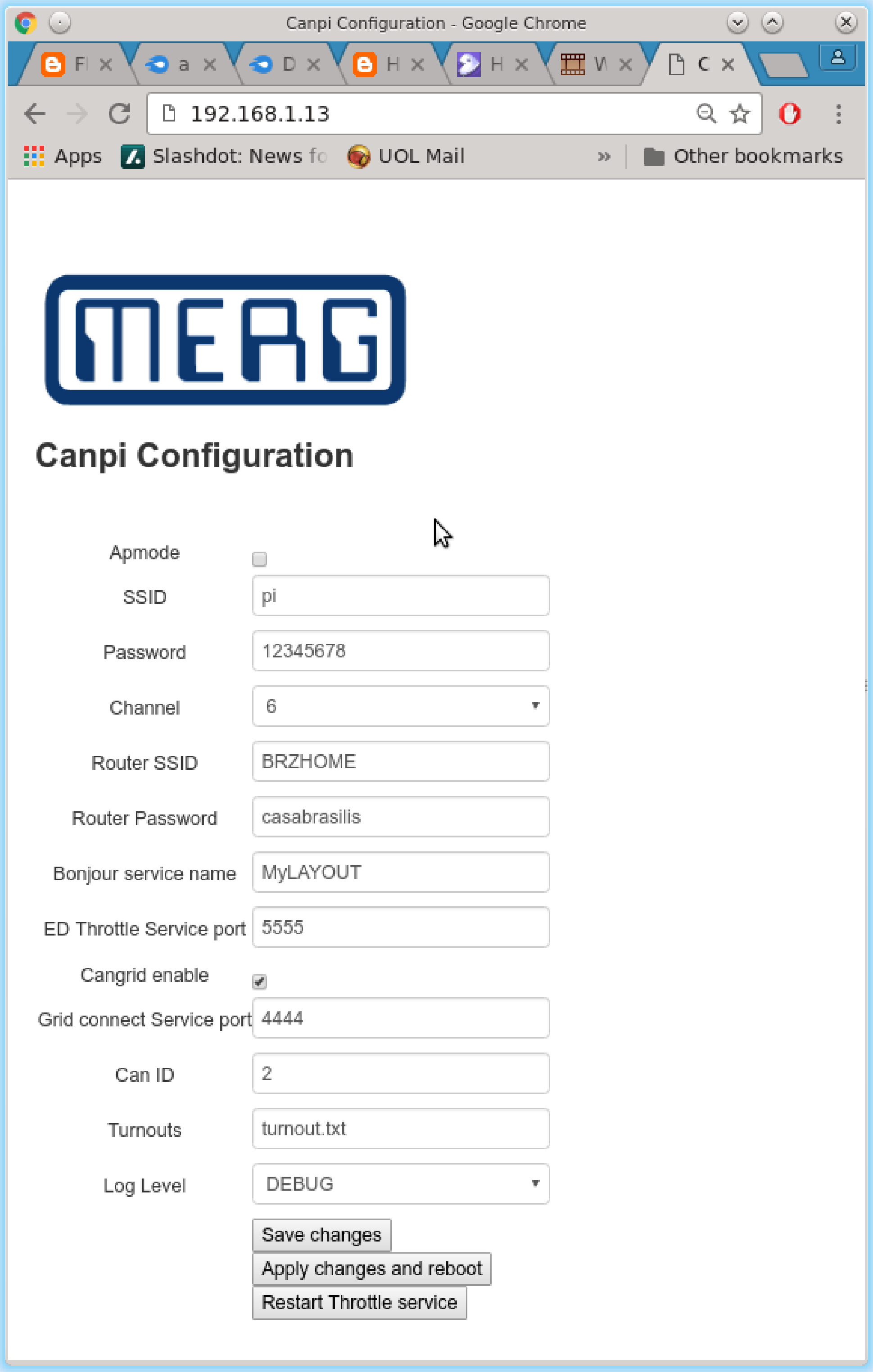
Hostapd (Host access point daemon) is a user space software access point capable of turning normal network interface cards into access points and authentication servers. The current version supports Linux (Host AP, madwifi, mac80211-based drivers) and FreeBSD (net80211)

The CW uses the hostapd to put the RP in AP mode. The user has the option to set the wifi channel, ssid name and ssid password. The AP mode requires the wlan0 to have a fixed ip address. The start_canpi.sh configure do the proper configuration by copying the template files present in **/home/pi/canpi** to the **/etc/default** and **/etc/hostapd/** and restart the RP.

For more information on hostapd see https://wiki.gentoo.org/wiki/Hostapd

6.6 Webpy

WebPy is light webserver in python. CW uses it to offer a configuration interface by the browser. Once the CW is turned on and active, the user can access the configuration interface using a browser. This the screen of the webpage:



Canpi Configuration

Apmode	☐
SSID	pi
Password	12345678
Channel	6
Router SSID	BRZHOME
Router Password	casabrasilis
Bonjour service name	MyLAYOUT
ED Throttle Service port	5555
Cangrid enable	✓
Grid connect Service port	4444
Can ID	2
Turnouts	turnout.txt
Log Level	DEBUG

This page is used to set CW configuration. The user can select any of the configuration above. Not all configuration possible are present, and we can add more by changing the program **canpiconfig.py** present in **/home/pi/canpi/webserver**.

The button "*Save changes*" write the configuration to the the canpi.cfg.

"*Apply changes and reboot*" reconfigures the pi and reboot while "*Restart Throttle service*" just restarts the canpi daemon.

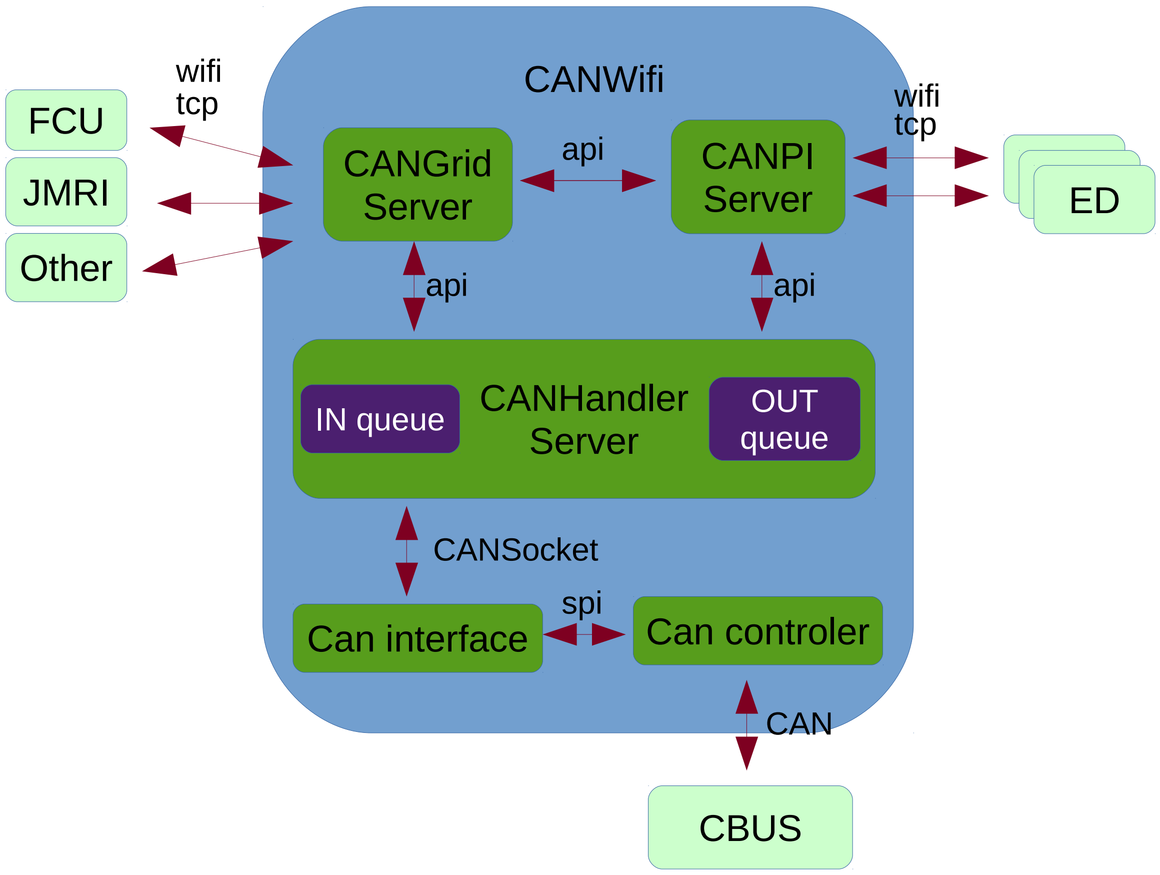
To start the page manually do

sudo python configcanpi.py 80

It will start the service for all ip interfaces on the port 80. This port is the default for the HTTP protocol.

6.7 canpi daemon and scripts

The canpi daemon is a multithread application that communicates with the CBUS and the ED clients connected. It also provides a service that allows other applications access the CBUS via grid connect protocol.
Here is the architecture of the application:



When the CW starts, the script **start_canpi.sh** in **/etc/init.d**, starts the canpi daemon and the configuration webpage. The canpi daemon starts 3 major threads:
- canHandler
- tcpServer
- tcpGridServer

6.7.1 canHandler

canHandler is a process that controls the data from and towards CBUS. This same process also deals with the configuration messages from FCU.

The CAN interface is controled by the linux kernel module, has the MCP2515 driver loaded, which communicates with the CAN Controler via SPI protocol.

The canHandler process communicates with the CAN interface using the CANSocket api. Internally it maintains 3 threads:

- one the to read the can interface and put the message on a incoming queue

- another to read the incoming queue and send the messages towards the tcpServer and tcpGridServer.

- another thread to read the outgoing queue and send it to the CBUS and to the tcpGridServer.

The canHandler guarantees that all messages from and towards the CBUS reaches all the EDs and all the clients of the tcpGridServer.

6.7.2 tcpServer

This process creates a tcp server on the port defined in the **canpi.cfg** under the property **tcpport**. If no port is configured it will use the port 4444. This is the component on which the EDs connect to.

When ever a client connects, the tcpServer creates a new thread to handle the communication with each ED. Each ED has access to the canHandler and can send and receive messages to CBUS. The messages to CBUS are included in the outgoing queue and the messages in the incoming queue are sent to all EDs.

The messages from on ED to the CBUS also reaches the others EDs, but because they are handling different sessions, it does not affect each other.

6.7.3 tcpGridServer

This process creates a tcp server on the port defined in the **canpi.cfg** under the property **cangrid_port**. If no port is configured it will use the port 5555. The FCU and JMRI connect to this server on this port. It is optional to not start this process by setting the property **can_grid** to “false”.
The tcpGripServer translates the CAN messages to the GridPrototol format and vice-versa.
The GridConnect protocol encodes messages as an ASCII string of up to 24 characters of the form:

:ShhhhNd0d1d2d3d4d5d6d7;

The S indicates a standard CAN frame

:XhhhhhhhhNd0d1d2d3d4d5d6d7;

The X indicates an extended CAN frame hhhh is the two byte header N or R indicates a normal

or remote frame, in position 6 or 10 d0 - d7 are the (up to) 8 data bytes.

The actual implementation just support the standard CAN frame.

7 Appendix A – Initial setup script

```
#!/bin/bash

PATH=/sbin:/bin:/usr/sbin:/usr/bin
. /lib/lsb/init-functions

dir="/home/pi"
canpidir="/home/pi/canpi"

append_to_file(){
   val=$1
   file=$2
   grep -q "$val" "$file" || echo "$val" >> $file
}

config_boot_config(){
   bootconf="/boot/config.txt"
   cp $bootconf "/boot/config.txt.$RANDOM"
   sed -i "s/dtparam=spi=off/dtparam=spi=on/" $bootconf
   #in case the entry is not there
   append_to_file "dtparam=spi=on" $bootconf
   ls /boot/overlay/mcp2515-can0-overlay*
   if [ $? == 0 ];then
       append_to_file "dtoverlay=mcp2515-can0-
overlay,oscillator=16000000,interrupt=25" $bootconf
   else
       append_to_file "dtoverlay=mcp2515-
can0,oscillator=16000000,interrupt=25" $bootconf
   fi
#    append_to_file "dtoverlay=spi-bcm2835-overlay" $bootconf
}

add_can_interface(){
  f="/etc/network/interfaces"
  append_to_file "auto can0" $f
  append_to_file "iface can0 can static" $f
  append_to_file "bitrate 125000 restart-ms 1000" $f
}

create_default_canpi_config(){
  conf="$canpidir/canpi.cfg"
  echo "#log config" > $conf
  echo "logfile=\"canpi.log\"" >> $conf
  echo "#INFO,WARN,DEBUG" >> $conf
  echo "loglevel=\"INFO\"" >> $conf
  echo "logappend=\"false\"" >> $conf
  echo "#ed service config" >> $conf
  echo "tcpport=5555" >> $conf
  echo "candevice=\"can0\"" >> $conf
  echo "#cangrid config" >> $conf
  echo "can_grid=\"true\"" >> $conf
  echo "cangrid_port=4444" >> $conf
  echo "#Bonjour config" >> $conf
  echo "service_name=\"MYLAYOUT\"" >> $conf
  echo "#network config" >> $conf
  echo "ap_mode=\"False\"" >> $conf
  echo "ap_ssid=\"pi\"" >> $conf
  echo "ap_password=\"12345678\"" >> $conf
  echo "ap_channel=6" >> $conf
  echo "#if not running in ap mode use this" >> $conf
  echo "router_ssid=\"SSID\"" >> $conf
  echo "router_password=\"PASSWORD\"" >> $conf
  echo "node_number=7890" >> $conf
  echo "turnout_file=\"turnout.txt\"" >> $conf
  echo "canid=110" >> $conf
}

echo "Type the Wifi SSID followed by [ENTER]:"
read wssid

echo "Type the Wifi password followed by [ENTER]:"
read wpassword

echo "########### APT UPDATE ###############"
apt-get update
echo "########### GIT ###############"
apt-get install git
echo "########### HOSTAPD ###############"
apt-get install hostapd
echo "########### DHCP ###############"
apt-get install isc-dhcp-server
echo "########### CAN UTILS ###############"
apt-get install can-utils

echo "########### BOOT CONFIG ###############"
config_boot_config
echo "########### CAN INTERFACE ###############"
add_can_interface

echo "########### GET THE CANPI CODE ###############"
#get the code
cd $dir
git clone https://github.com/amaurial/canpi.git

echo "########### COMPILE CANPI ###############"
#compile the code
cd canpi
make clean
make all
echo "########### CREATE CONFIG ###############"
create_default_canpi_config
```

```
#change the router ssid and password
sed -i 's/SSID/'"$wssid"'/' "$canpidir/canpi.cfg"
sed -i 's/PASSWORD/'"$wpassword"'/' "$canpidir/canpi.cfg"

echo "########### WEBSERVER ###############"
#install the webpy
tar xvf webpy.tar.gz
mv webpy-webpy-770baf8 webpy
cd webpy
python setup.py install

echo "########### CHANGE DIR OWNER ###############"
cd $dir
chown -R pi.pi canpi

echo "########### MOVE CONFIG FILES ###############"
#backup and move some basic files
FILE="/etc/hostapd/hostapd.conf"
FILEBAK="${FILE}.bak"
if [ -f $FILE ];
then
    mv $FILE $FILEBAK
fi
cp "$canpidir/hostapd.conf" /etc/hostapd/


FILE="/etc/default/hostapd"
FILEBAK="${FILE}.bak"
if [ -f $FILE ];
then
    mv $FILE $FILEBAK
fi

cp "$canpidir/hostapd" /etc/default/

FILE="/etc/dhcp/dhcpd.conf"
FILEBAK="${FILE}.bak"
if [ -f $FILE ];
then
    mv $FILE $FILEBAK
fi

cp "$canpidir/dhcpd.conf" /etc/dhcp/

mv /etc/default/isc-dhcp-server /etc/default/isc-dhcp-server.old
cp "$canpidir/isc-dhcp-server" /etc/default

echo "########### CONFIG SCRIPT FILES ###############"
#copy the configure script
cp "$canpidir/start_canpi.sh" /etc/init.d/
chmod +x /etc/init.d/start_canpi.sh
update-rc.d start_canpi.sh defaults
```

```
echo "########### RUN CONFIGURE ###############"
#run configure
/etc/init.d/start_canpi.sh configure
```