Installing CCGX functionality on a raspberry pi

(in progress, everybody reading this page, feel free too add and improve)

This page describes my (Matthijs Vader) experience while trying to make the CCGX code work on a raspberry pi.

Just for fun, originally more a Windows man, and with initially as little help from our own software engineers as possible. :).

Below steps have been done on a raspberry pi 2. Much of the source code is not yet available publicly. So to start, I am putting instructions down for all the projects that are available. And for projects that contain private Victron source code I'll perhaps see if I can make them compile, and put the binaries here for download. A explanation for adding binary packages to Raspbian wheeze is here.

Cross compile vs compiling locally

There are many different ways to make the ccgx functionality run on a rpi. For example do something with Poky/Yocto/Open embedded, our build system. But to keep it simple and also more fun, I will compile all of it locally on the rpi.

Optimizing make and gcc (not required!)

Below steps have all been done on my new raspberry pi 2, running this version:

```
pi@raspberrypi ~ $ cat /proc/version
Linux version 3.18.7-v7+ (dc4@dc4-XPS13-9333) (gcc version 4.8.3 20140303
(prerelease) (crosstool-NG linaro-1.13.1+bzr2650 - Linaro GCC 2014.03) )
#755 SMP PREEMPT Thu Feb 12 17:20:48 GMT 2015
pi@raspberrypi ~ $ cat /etc/issue
Raspbian GNU/Linux 7 \n \l
pi@raspberrypi ~ $ gcc --version
gcc (Debian 4.6.3-14+rpi1) 4.6.3
```

This post on the raspberry forums prompted me to try and install a newer gcc version (4.8 instead of 4.6) as well as running make with -j4, to use all 4 cores instead of only 1.

To see if there really was an improvement in compile time, I have timed compiling the git sources. Running make clean before retrying.

With the original gcc version (4.6), it took 12m26s With the original gcc version, and -j4, it took only 4m12 !

Then switched to the new compiler:

```
export CC=gcc-4.8
```

And ran make clean again, and ./configure –prefix=/usr. The output of ./configure shows that it will now use gcc-4.8.

Install screen (not required!)

screen is a handy little to that jeeps your session running after disconnect led from Ccgx. Most of my raspberry Pi work I am doing in left over time using my phone, therefore often just a few minutes trying out the next command: I love the screen function!

To install:

```
sudo apt-get install screen
```

Then to start a session:

```
screen
```

you'll get a fresh shell. You can even add second shell using Ctrl-c, and cycle through them with Ctrl-n.

Next time when you login, reconnect to the still running session with:

```
screen -r
```

Steps

1. Install daemontools

Daemontools is a set of tools to run services: it runs an executable, and when it fails it will start it again. To see some commonly used commands, like starting and stopping a service, or seeing status of all services, check the CCGX command-line introduction.

To install daemontools follows the instructions on its install page, with one addition: right before running /package install, there is one change that is necessary to make it compile, without it, you'll get a message “/usr/bin/ld: errno: TLS definition in …”. To fix this, open conf-cc in ./src, and append the following parameter to the gcc call:

```
-include /usr/include/errno.h
```

Daemontools will now be running, to make sure check the processlist:

```
pi@raspberrypi ~ $ ps ax | grep svscan
 2171 ?        Ss     0:00 /bin/sh /command/svscanboot
```

```
 2173 ?        S      0:00 svscan /service
 2260 pts/0    S+     0:00 grep --color=auto svscan
```

But is not doing anything yet, as there are no services setup in /service yet.

2. Install the dbus-cli

dbus-cli is a command line interface to dbus. Very useful to play with and debug all the different CCGX services. To find what D-Bus is, read the CCGX introduction and look on Google. See CCGX commandline intro for basic usage.

First, get the code from above, and put it somewhere in the path, like so:

```
cd /usr/local/bin
sudo wget https://dbus-tools.googlecode.com/svn/trunk/dbus
sudo chmod a+x ./dbus
```

Then, get some dependencies, first install pip, the python installer:

```
sudo apt-get install python-pip
```

Then install lxml, which is needed by dbus-cli:

```
sudo apt-get -y install libxml2-dev libxslt1-dev python-dev python-lxml
```

Done! If all works, you'll be able to run dbus -y from the commandline:

```
pi@raspberrypi ~ $ dbus -y
org.freedesktop.DBus
```

3. Install dbus-modbustcp

dbus-modbustcp is a modbustcp server, that allows for example PLCs to get data (battery voltage, etc.) from the D-Bus.

First get the sources:

```
cd
mkdir dev
cd dev
git clone https://github.com/victronenergy/dbus_modbustcp.git
```

Install the qt libraries:

```
sudo apt-get install libqt4-dev
```

Run qmake to create the makefile (set MACHINE to ccgx to make dbus_modbustcp use the system D-Bus):

```
export MACHINE=ccgx && qmake
```

Run make to compile the sources:

```
make
```

Done! Run the binary to see that you were successful:

```
~/dev/dbus_modbustcp $ ./dbus_modbustcp
INFO 2015-02-27T21:04:03.752 dbus_modbustcp v0.6.3 started (v0.6.3)
INFO 2015-02-27T21:04:03.752 Built with Qt 4.8.2 running on 4.8.2
INFO 2015-02-27T21:04:03.753 Built on Feb 27 2015 at 21:03:50
ERROR 2015-02-27T21:04:03.784 "[Server] QTcpServer error: The address is
protected"
INFO 2015-02-27T21:04:03.787 [Mappings] Add ("com.victronenergy.vebus",
"/Ac/ActiveIn/L1/V", "d", "V AC", "3", "uint16", "10", "R")
INFO 2015-02-27T21:04:03.788 [Mappings] Add ("com.victronenergy.vebus",
"/Ac/ActiveIn/L2/V", "d", "V AC", "4", "uint16", "10", "R")
```

Note that I have not yet looked into that error!

sudo ln -s /opt/color-control/dbus-fronius/service/ /service/dbus-fronius

4. Install dbus_qwacs

dbus_qwacs reads data from the Wireless AC Sensors and makes it available on D-Bus, for all the other processes.

To compile, follow the same steps as dbus_modbustcp.

When done, output will look like this:

```
~/dev/dbus_qwacs $ ./dbus_qwacs
INFO 2015-02-27T21:19:24.723 dbus_qwacs v1.0.1 started (v1.0.1)
INFO 2015-02-27T21:19:24.724 Built with Qt 4.8.2 running on 4.8.2
INFO 2015-02-27T21:19:24.724 Built on Feb 27 2015 at 21:15:59
INFO 2015-02-27T21:19:24.728 Wait for local setting on DBus...
INFO 2015-02-27T21:19:26.733 Wait...
INFO 2015-02-27T21:19:28.735 Wait...
```

And that goes on forever, since the local settings process is not yet installed, lets do that first!

5. Local settings

localsettings is a D-Bus settings manager that interfaces between xml file on disk(/data/conf/settings.xml) and D-Bus.

Steps to install (probably not as it should be done with regards to rights etc, welcome to correct this! Also it is perhaps better to make a install script for localsettings. Instead of just cloning the whole thing and adding some files, same thing: if you can improve, please do):

```
cd /opt
sudo mkdir color-control
cd ./color-control
sudo chmod a+w .
git clone --recursive https://github.com/victronenergy/localsettings.git
cd localsettings

sudo apt-get install python-gobject
```

Create the folder where localsettings will store settings.xml

```
sudo mkdir /conf
sudo chmod a+w /conf
```

All done (except for setting this up with daemontools). Run it to see what happens:

```
$ ./localsettings
localsettings v1.01 starting up
2015-03-01 14:09:00,093 INFO Localsettings version is: 0x0101
2015-03-01 14:09:00,094 INFO Localsettings PID is: 18457
2015-03-01 14:09:00,097 INFO Settings file /conf/settings.xml validated
Traceback (most recent call last):
  File "./localsettings.py", line 697, in <module>
    main(sys.argv[1:])
  File "./localsettings.py", line 695, in main
    run()
  File "./localsettings.py", line 643, in run
    busName = dbus.service.BusName(dbusName, bus)
  File "/usr/lib/python2.7/dist-packages/dbus/service.py", line 131, in
__new__
    retval = bus.request_name(name, name_flags)
  File "/usr/lib/python2.7/dist-packages/dbus/bus.py", line 303, in
request_name
    'su', (name, flags))
  File "/usr/lib/python2.7/dist-packages/dbus/connection.py", line 651, in
call_blocking
    message, timeout)
dbus.exceptions.DBusException: org.freedesktop.DBus.Error.AccessDenied:
Connection ":1.7" is not allowed to own the service
"com.victronenergy.settings" due to security policies in the configuration
file
```

So.. that doesnt work yet. The problem is that there are two D-Busses: system and session. On the CCGX we use the system D-Bus: all services, such as localsettings will connect tot the system D-Bus instead of the session one. But on Linux by default the system D-Bus is locked down: only user such and such is allowed to create service such and such, etc. I have looked a few seconds into what the session D-Bus is, and what it is normally used for, and it seems that there is one per user. And that

doesn't seem like a good idea. So I decided to open up the system D-Bus instead, to make it match the CCGX configuration.

Open the config file with:

```
sudo nano /etc/dbus-1/system.conf
```

And then look for the <policy>. It is quite a long one. What I did is just replace it with this part that I found on a CCGX:

```
 <policy context="default">
    <!-- All users can connect to system bus -->
    <allow user="*"/>

    <!-- Signals and reply messages (method returns, errors) are allowed
         by default -->
    <allow send_type="signal"/>
    <allow send_requested_reply="true" send_type="method_return"/>
    <allow send_requested_reply="true" send_type="error"/>
    <allow send_interface="*"/>
    <allow receive_interface="*"/>
    <allow receive_sender="*"/>

    <!-- All messages may be received by default -->
    <allow receive_type="method_call"/>
    <allow receive_type="method_return"/>
    <allow receive_type="error"/>
    <allow receive_type="signal"/>

    <!-- Allow everything to be sent -->
    <allow send_destination="*" eavesdrop="true"/>
    <!-- Allow everything to be received -->
    <allow eavesdrop="true"/>
    <!-- Allow anyone to own anything -->
    <allow own="*"/>

    <!-- Allow anyone to talk to the message bus -->
    <allow send_destination="org.freedesktop.DBus"/>
  </policy>

  <limit name="max_match_rules_per_connection">1024</limit>
```

Alternatively it could also be a good idea to look even better at how our smart guys have configured this on the CCGX, but for now the goal of this adventure is just to make it work. Not to do it by the book. :)

So, after making above change, reboot (sudo reboot), and retry:

```
$ ./localsettings
localsettings v1.01 starting up
```

```
2015-03-01 14:30:30,827 INFO Localsettings version is: 0x0101
2015-03-01 14:30:30,827 INFO Localsettings PID is: 2266
2015-03-01 14:30:30,840 INFO Settings file /conf/settings.xml validated
```

Yes! Now open a second terminal, and see if you can talk to the settings:

```
pi@raspberrypi ~ $ dbus -y
org.freedesktop.DBus
com.victronenergy.settings
pi@raspberrypi ~ $ dbus -y com.victronenergy.settings
/
/Settings
```

Yes! From that same terminal, we can try creating a new setting:

```
$ dbus -y com.victronenergy.settings /Settings AddSetting Matthijs Setting1
50 i 0 100
```

It will return 0, which means succesfull, and in the other terminal you'll this:

```
2015-03-01 14:33:02,167 INFO Added new setting /Settings/Matthijs/Settings1.
default:50, type:i, min:0, max: 100
```

To see more instructions, see the readme in localsettings.

So still left on the todo list is making localsettings run as a service under daemontools. More later.

DISQUS

~~DISQUS~~