Lafayette College | Electrical and Computer Engineering

VSCADA Maintenance Manual

ECE 492 Spring 2017

Assembled by Craig Lombardo and Austin Wiles

Table Of Contents:

Software

Java

Java was both a good choice and terrible one. It was great to use Java 90 percent of the time; however, that other 10 percent was rough. That 10% was working with CAN (yes CAN caused us a lot of problems). Since Java does not easily allow for access to things like GPIO, it might not be the best idea to use for a project which would ideally like to leverage such capabilities. If there are any additional problems or questions with the Java code that needs clarification feel free to reach out to Craig Lombardo (201-663-1127) with any questions.

SQLite Database

The SQLite Database is good, reliable, and fast. The only problem is that we got wrapped up in everything else that we were not able to provide means for trimming data. Yes that means this database will grow until the SD card has no more space, or you go and clean things up. The limit will not be SQLite and we can almost guarantee it as there is "maximum SQLite database size of about 140 terabytes." (Tough to believe but: https://www.sqlite.org/limits.html) SQLite also does a great job of preventing from data loss through abrupt shutdowns. Things are not actually written to the disk until the entire write is complete meaning data can be written; however, until the entire set of data is inserted the data is not 'declared' as written to the disk. It is for this reason that we decided to use SQLite rather than mySQL or a SQLServer.

Graphics

All of the views function and use Java Swing libraries. We initially wanted to move to JavaFX; however, ARM processors (which the PI is) do not come with JavaFX supports, nor will future versions. Additionally you cannot download support. Yet another potential pitfall to using this combination of Java/PI. Things worked in the end, it just took a little longer to ensure compatibility with everything and reliability. Graphics admittedly don't look flashy but it was never our intention to make the most beautiful display ever seen in the auto industry, but rather one that was clean, simple and robust. If you decide to move to a new hardware device (i.e. not ARM based) it is our recommendation that you move to JavaFX, it's where everything is moving.

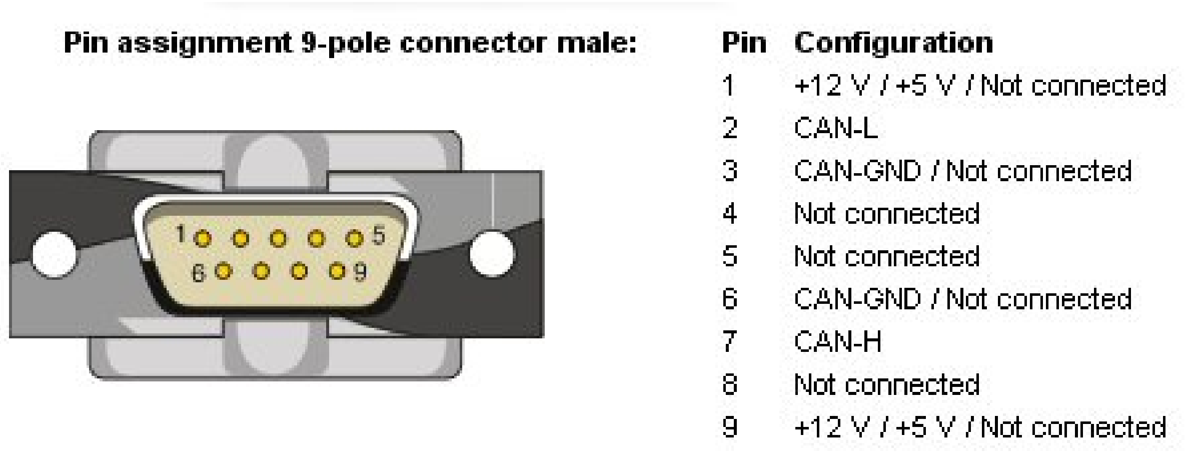
Hardware

Raspberry PI 3B

The device itself is great and we were able to greatly reduce the CPU usage in comparison to last year’s team, proving that the PI has the power; however, we ran into issues with graphics, CAN and the hardware onboard. Beware that if the PI does not have a high enough power supply then it will run in low power mode. The PI itself will run properly and should work without any problems, but the problem rests in the devices it is connected to. When the PI runs in low power mode it cannot push as much to all of its peripherals. We experienced an issue with devices connected to the PI working and then not and the issue happened to be because it was not and then was running in low power mode and the connected devices were not functioning properly. Don’t be so quick to blame the PI for all of your hardware problems, but don’t rule this possibility out.

USB2CAN

The USB2CAN device has caused us many more problems than it has solved. It is our recommendation that the USB2CAN device is not used further. We have experienced problems with connection, reliable data acquisition and occasionally it would disconnect and give us the “PC USB driver isn’t installed properly” error (red and green flashing) during normal use. When it worked it was great; however, we spent more time trying to fix it for our specific needs than actually using it. To further our recommendation, we feel that the Raspberry PI should be ditched to make way for a board with native CAN so we do not need to deal with such devices.



Pin assignment 9-pole connector male:

Pin	Configuration
1	+12 V / +5 V / Not connected
2	CAN-L
3	CAN-GND / Not connected
4	Not connected
5	Not connected
6	CAN-GND / Not connected
7	CAN-H
8	Not connected
9	+12 V / +5 V / Not connected

120 Ohm termination between CAN high and CAN low

LED indicator values

RED and GREEN steady	bootloader mode
RED/GREEN toggling	PC USB driver isn't installed properly
RED steady	converter is closed
GREEN steady	converter is open
GREEN blinking slow	CAN bus warning
GREEN blinking fast	CAN bus error passive
RED blinking slow	CAN bus off

Driver GitHub repository: https://github.com/krumboeck/usb2can

WiFi Extender

Initially we thought the device was either broken or the wrong device, turns out it's exactly what you want. Reference the User Manual for assistance on set up, a little finicky but we've worked out all of the issues (we think). We have followed the instructions verbatim 5 times on a clean SD card and had 5 successful wireless access points created with the PI. The problem mentioned in the Raspberry PI section with regards to hardware plagued with WiFi extender. If it does not receive enough power then it will not function properly, the issue we saw most often was the SSID would appear on the list of WAPs; however, you could not actually connect to it. Powering the PI a little more fixed this problem. If that does not fix the problem then refer back to the User Manual, specifically the server setup section and ensure that all files mentioned match the instructions, the slightest typo can wreck the entire WAP.

Adding New Sensors

1) Allocate ID for new sensor in ID Allocation Table (found in VSCADA's ICD)
2) Set offset and byte lengths for new sensors data over CAN
3) Connect new Sensor to CAN bus
4) VSCADA will then be able to collect data from new sensor

Interfaces

CAN

Can works with the code we have set up but as aforementioned, it isn't the prettiest considering we are using a device that does not natively have CAN support and a language which does not natively support GPIO access. Luckily there is not much that can go wrong with the CAN network once it is up and running. Make sure that there is proper termination at junctions otherwise CAN may fail and then the entire data acquisition part of VSCADA is kind of shot.

I2C

We did not accomplish this task as it was pushed out of our WBS. This is important moving forward though as it will allow for data retrieval from the GLV team and will eventually allow for direct control of the Dyno. The datasheets for the two parts that will need to be accessed via I2C are as follows:

http://www.analog.com/media/en/technical-documentation/data-sheets/AD5593R.pdf
http://www.ti.com/lit/ds/symlink/ina226.pdf

While we were not able to accomplish I2C we would recommend enlisting the help of the GLV team and/or Professor Watkins.

UART

We did not accomplish this task as it was pushed out of our WBS. This will be used to communicate with the TSI team. The TSI team got swamped with documentation and forms which restricted their ability to focus on the project as they were consistently pulled away from their work to complete these forms. When this is set up, the idea of using a communication protocol that no other systems are using is to ensure there is always maximum priority with TSI. Since we will be communicating to them information about cruise control response.

General Notes

If things go awry, don’t be afraid to start over again. The install instructions are straight-forward and are pretty reliable. The only potential problem that we could see with this are the interactions with the USB2CAN device. We aren’t entirely sure why but for some reason the USB2CAN device works only sometimes and depending on the kernel that you are using you may have a whole new set of problems so be careful when re-installing things. Generally the best place to refer to when having problems with the software is the user manual.