Supervisory Control and Data Acquisition (SCADA) Software and Firmware Maintainability Document

Team: Irwin Frimpong, Adam Tunnell, Mithil Shah, Harrison Walker, Lia Chrysanthopoulos

December, 7th 2020

SCOPE:

The purpose of this document is to outline the software design decisions and the maintainability plans of the SCADA team

Table of Contents

INTRODUCTION

The overarching goal for the Supervisory Control and Data Acquisition (SCADA) system is to have both a maintainable and configurable system that is able to interface with devices & sensors on the car using different communication protocols, currently I2C and CANOPEN . In terms of functionality, the SCADA system is capable of reading from the respective communication protocols (I2C, CANOPEN) and displaying real time data from every connected sensor and present post-processed data for later review. In addition to passive data acquisition, an active system control is being implemented in SCADA to send out alerts based on sensor thresholds defined in the configuration file. This document details how the SCADA team plans to address the maintainability requirements as it pertains to Software.

SCADA INSTALLATION

The SCADA system runs on Raspberry Pi and or any device running a Debian based linux distribution, which the Raspbian OS on the Raspberry Pi is; Further there must be a viable way of communicating with the sensor protocols (CAN & I2C). To mitigate the hassle of installing the software dependencies (Python, SMBUS, CAN,etc) as well as the SCADA modules and service files, a bash installation shell script was created to automate the installation on a new Raspberry PI device .

Clone Git Repository and Run Bash Install Script:

To initiate the installation, open up the terminal and write the following shell commands:

```
$ git clone https://github.com/irwinfrimpong/scadafsae.git
$ cd scadafsae
$ sudo bash install
```

The Bash Install Script Explained

```
apt-get install python3
apt-get install python3-pip
apt-get install redis-server
apt-get install can-utils
apt-get install i2c-tool
apt-get install python-smbus
apt install postgresql libpq-dev postgresql-client postgresql-client-common -y
```

Python 3.8.6 Documentation: https://docs.python.org/release/3.8.6/
Python PIP 20.2.4 (Python Package Installer) Documentation: https://pip.pypa.io/en/stable/
Redis-Server Documentation: https://redis.io/topics/quickstart
Can Utils Documentation: https://sgframework.readthedocs.io/en/latest/cantutorial.html
I2C / SMBUS Documentation: https://smbus2.readthedocs.io/en/latest/

```
pip3 install python-can
pip3 install redis
pip3 install blessed
pip3 install psycopg2-binary
pip3 install pyyaml
pip  install pyyaml
pip3 install canopen
```

Python Can Documentation: https://python-can.readthedocs.io/en/master/
Python Redis Documentation: https://redis-py.readthedocs.io/en/stable/
Python Psycopg PostgreSQL Documentation: https://www.psycopg.org/docs/
Python PyYAML Documentation: https://pyyaml.org/wiki/PyYAMLDocumentation
Python Canopen Documentation: https://canopen.readthedocs.io/en/latest/

Start-up on Boot Automated Setup for Graphical User Interface (GUI)

```
sudo chmod +x GUI/New_GUI_Test.py
cd /usr/bin
sudo chmod +x scada_gui.py
cp -r /etc/xdg/lxsession /home/pi/.config
cd /home/pi/.config/lxsession/LXDE-pi
## Add following line to end of file
echo '@/usr/bin/scada_gui.py' >> autostart

echo 'INSTALL COMPLETE'
bash make
```

Run The Bash Make Shell Script

To initiate the Make Shell Script , run the following shell command in command line,making sure that you are already in the scadafsae repository:

```
$sudo bash make
```

The Bash Make File Explained

Setting Up The CanBus Connection for Testing

```
modprobe can
ip link set can0 down
ip link set can0 up type can bitrate 125000
```

Making Binary Files Executable

```
chmod +x install
chmod +x make
chmod +x scada
chmod +x sorter/sorter_new.py
chmod +x calibrator/calibrator_new.py
chmod +x logger/logger_new.py
chmod +x GUI/New_GUI_Test.py
```

Copying SCADA modules to /usr/bin

```
cp scada /usr/bin/scada
cp sorter/sorter_new.py /usr/bin/scada_sorter.py
cp calibrator/calibrator_new.py /usr/bin/scada_calibrator.py
cp logger/logger_new.py /usr/bin/scada_logger.py
cp GUI/New_GUI_Test.py /usr/bin/scada_gui.py
```

Creating Workspace and Copying SCADA binary files into it

```
mkdir -p /usr/etc/scada
rm -rf /usr/etc/scada/utils
cp -r utils /usr/etc/scada/utils
rm -rf /usr/etc/scada/config
cp -r config /usr/etc/scada/config
cp ./install /usr/etc/scada
```

```
cp ./make /usr/etc/scada
rm -rf /usr/etc/scada/GUI
cp -r GUI /usr/etc/scada/GUI
rm -rf /usr/etc/scada/drivers
cp -r drivers /usr/etc/scada/drivers
```

Copy Service Files Down to SystemD for Startup on Boot

```
cp sorter/sorter.service /etc/systemd/system
cp calibrator/calibrator.service /etc/systemd/system
cp logger/logger.service /etc/systemd/system
cp GUI/gui.service /etc/systemd/system
```

Rerun all generators and reload all unit files

```
systemctl daemon-reload
echo 'MAKE COMPLETE'
```

PostgreSQL Database Setup

To configure the database, run the following commands in the command line:

This switches to the local Postgres user to configure the database .

```
$sudo su postgres
```

Creates a database user and when prompted with a password type "scada'

```
$createuser pi -P --interactive
```

Answer the following questions accordingly when prompted

Shall the new role be useruser ? (y/n) - Select n
Shall the new role be allowed to create databases? (y/n) Select y
Shall the new role be allowed to create more new roles? (y/n) Select y

Connect to Postgres using the shell and create "test" database

```
$ psql
> create database test;
```

To return to main terminal, press Ctrl + D twice

Enable I2C On The Raspberry PI

Inter-Integrated Circuit (I2C) bus is a two wire serial interface using a serial data line (SDL) , serial clock line (SCL) and a common ground for all the communication over the bus. I2C follows a master/slave approach where the master clocks the bus, uniquely addresses the slaves and is able to read and write from the respective registers of the slaves. The Raspberry PI supports the I2C protocol with its GPIO pins where sensors and other devices that communicate with I2C can be read. This functionality is disabled by default and to activate it , one must perform the following actions:

In command line type the following command:

```
$ sudo raspi-config
```

1. This should launch the raspi-config utility where you would want to select (5)- Interfacing Options
2. Navigate to and select the “P5 I2C” option to activate the Raspberry PI I2C protocol
3. Reboot the Raspberry PI

System Configuration

In order to support different sets of sensors beyond a one-time, hard-coded vehicle configuration, SCADA implements an editable human-readable configuration file. This file, which uses the YAML format, defines configuration options for:

- the data buses the system will read from
- the sensors transmitting to/from these data buses
- the layout of the real time data display GUI
- "Watcher" active system control conditions/actions

Because of the flexibility offered by the configuration file, SCADA can be used for any vehicle data collection, monitoring any safety conditions, with any set of sensors, as long as they can transmit to/from supported protocols. At the time of writing, supported protocols are CANopen and I2C.

The first section of configuration defines configuration for the CAN bus.

```
bus_info:
  CAN:
    bus_type: socketcan   #socketcan for actual bus, vcan for virtual bus
    channel: can0         #can0 for actual bus, vcan0 for virtual bus
    bitrate: 125000       #should always be 125000

# CAN node ids
can_nodes:
  motor: 1
  tsi: 3
  pack1: 4
  pack2: 5
```

The second section defines configuration for the SCADA software itself. Specifically, it includes sections for the logger and GUI display. The logger configuration at the time of writing was only a list of database keys not to be logged, and may not be necessary in future updates.

```
# a list of keys to be ignored by the logger
dont_log: ['*-raw','emulator-*']
```

For the GUI display, we define both groups of sensors to display together, and the layout of the GUI pages these sensors may be found on. As an example, a group called `Motor_Group2` is defined below.

```
# max 22 sensor per 1 group
Motor_Group2:
  - rtc_time
  - motor_throttle_voltage
  - motor_status
  - motor_controller_temp_fahrenheit
```

As an example, the layout of page `1` is defined below. Each group in the page `1` list appears as a column of sensors on that page of the GUI.

```
#max of 3 groups per page
    1:
      - Motor_Group1
      - Motor_Group2
      - Motor_Group3
```

Finally, we define the attributes needed to read from, calibrate, and display any sensor on the vehicle. An example sensor is defined below, as it is directly above the individual sensor portion of the configuration file.

```
subsytem_name_data_name: #this is the sensor name
  unit: V
  inputs:
    varName1: sensor1_name
    varName2: sensor2_name             ### Note: an actual sensor would only have one
cal_function
  cal_function: "varName1 * varName2" <-- number calibration (display_variable must be
boolean or  number)
  cal function:                       <-- state calibration (display_variable must be
state)
    1: OFF
    2: STANDBY
    3: DRIVE-READY
    4: DRIVE
  bus_type: I2C
  primary_address: 0x01
  secondary_address: [0x68,0x69]    #single hex value for CAN, list for I2C
  precision: 2                      #number of decimal places
  display_variable: number          #number/state/boolean
  sample_period: 1                  #in seconds
  bit_length: 16                    #probably don't need this
  description: "description of the data being stored"
```

When configuring individual sensors, you may define three different types of calibration functions: mathematical, state, or conditional.

As an example of mathematical calibration functions, the inputs and calibration for `motor_temp` is defined below. As you can see, this calibration returns the raw data unchanged. This same type of calibration is used for any sensor that is already calibrated when read from its respective bus.

```
inputs:
  Tin: motor_temp
cal_function: 'Tin'
```

As an example of state calibration functions, the inputs and calibration for `motor_status` is defined below. As you can see, this calibration takes in a numerical raw data value and returns a state value represented by a string.

```
inputs:
  status-enum: motor_status
cal_function:
  0: 'OFF'
  1: 'READY TO SWITCH ON'
  2: 'SWITCHED ON'
  4: 'OPERATION ENABLED'
  8: 'FAULT'
```

As an example of conditional calibration functions, the inputs and calibration for `motor_controller_temp_over_23` is defined below. As you can see, this calibration returns a different mathematical conversion depending on the value of the raw data input.

```
inputs:
  in: motor_controller_temp
cal_function:
  'in>=23': 'in-23'
  'True': 'in'
```

Conditions being monitored and acted upon by the Watcher portion of our software will also be configured in the configuration YAML file. This configuration will allow the user to choose what sensors to monitor, what their threshold values are, and whether this threshold must be met just once, met a predetermined number of times in a predetermined time period, or met for a certain period of time consecutively. This configuration will also allow the user to choose a LOG output(writing something to the SCADA log file), WARNING output (sending a warning to the EPAL driver display), or ACTION output (writing a value to a sensor, or “taking control of the vehicle” by interfacing with a sensor on the vehicle, which are essentially the same action). An example of this configuration is not provided at this time because theWatcher has not yet been finalized, nor has its configuration layout.

System Backup & Restore Procedure

All backup and restoration of the SCADA system to a previous version is handled through the Github repository. All code is regularly updated and documented in the Git repository found currently at https://github.com/irwinfrimpong/scadafsae. The repository is also accessible from the FSAE Motorsports website, under the SCADA page, under Source Repositories, as a hyperlink to the text "Source Code Repository 2021" This git repository can be cloned and installed through the bash install script on the Raspberry Pi or any device with Linux distribution following the SCADA INSTALLATION section by writing the following shell commands as follows:

```
$ git clone https://github.com/irwinfrimpong/scadafsae.git
$ cd scadafsae
$ sudo bash install
```

Each official release, snapshot, and version of the SCADA system will be documented on the Git repository as well as on the FSAE Motorsports website on the SCADA subpage. The details for the release, snapshot, and version can be found in the ReadMe file of each release, as well as in the Configuration YAML file with the date the system was last edited.

Tool Chain

VSCode is the IDE we used to code our SCADA system, however any IDE that can run Python3 would be suitable for compiling and executing our code. The GUI is built using a Python3 extension named Tkiner. Tkiner requires no extra installation and is written to be compatible with any version of Python3. Python is a low maintenance software, which the GUI is also dependent on. For this reason, the system has stable functionality. We also created an install file to build the program on any fresh system. By running the command in terminal, *bash install,* the SCADA system is easily installed in minutes on any new development system.

Third Party Software

Redis and Postgres database services will be the two third party softwares that are a part of the SCADA system. The Redis and Postgres packages will be installed and updated when the bash install script is run and with the dependencies already up to date, no actions will take place. The version number and the release of the Redis and Postgres service used in the SCADA system will be documented in the manual, in the readme file, as well as on the website, to be installed manually if necessary without the bash install script. The programming language Python is used for the SCADA system, in the version 3 iteration which is also installed with the bash install script. The installation of these two services can be seen through the SCADA INSTALLATION under the Bash Install Script.

Errors and Logging

All software errors are handled with in-code exceptions and are written to a SCADA error log file which the user(s) can access. This log file is automatically cleared on system reset so no manual trimming is required.