LPRDS-BMS-2010

Lafayette Photovoltaic Research and Development System
Battery Management System
ECE492 – Spring 2010

Lafayette College
Electrical and Computer Engineering Department

User Manual

Final Draft

Contents

1 System Overview 4
2 System Components 5
2.1 DC Transfer Switch 5
2.2 Raw Power Interface (RPI) 5
2.2.1 Indicator LEDs 5
2.2.2 Isolation Contactor/Ground Fault Detector 5
2.3 Switch Controller (SC) 5
2.4 Filter & Inverter Box (FIB) 6
2.5 Energy Storage System (ESS) 6
2.5.1 Batteries 6
2.6 SCADA Interface Box (SIB) 6
2.7 Transformer 6
3 Getting Started 7
3.1 Normal Operation 7
3.1.1 Starting Up 7
3.1.2 Powering a Load 7
3.1.3 Shutting Down 7
3.2 Emergency Shutdown 7
3.3 Clearing Faults 8
4 FAQ 10
4.1 Calibration 11
5 Maintenance 11
5.1 Battery Maintenance 11
6 Schematics/Drawings 12
Appendix A – Scada ReadMe 14
1 SCADA Software 14
1.1 Goal 14
1.2 Basic Ubuntu Commands 15
1.2.1 ssh –Y *address* 15
1.2.2 ls 15
1.2.3 ps aux | grep *searchterm* 15

1.2.4 mv *fromLoc toLoc* 16
1.2.5 sudo *command* 16
1.2.6 tail *filename* 16
1.2.7 apt-get install *packagename* 16
2 Starting, Stopping, Compiling the Software 17
2.1 Starting and Stopping Kernel and the Operational State Manager 17
2.2 Starting and Stopping Applications 17
2.3 Compiling the Software 17
2.4 Troubleshooting 18
3 Using the Software 19
3.1 Software Block Diagram 19
3.2 Pipes 20
3.3 Database 20
3.4 SQL Commands 21
3.5 Installing MySQL and MySQL++ 22
3.6 Sunny Boy 23
3.6.1 Using the yasdi library: 23
3.7 Pico-LCD 23
3.7.1 Viewing the configuration file and the contents: 23
3.7.2 Starting and Stopping the configuration file: 24
4 Package List 25

1 System Overview

The Lafayette Photovoltaic Research and Development System - Battery Management System (LPRDS-BMS) is an experimental system that builds off of the work completed in 2009. It uses a photovoltaic array mounted on the roof of the Acopian Engineering Center to provide DC power for storage in a battery array and conversion to AC power by an inverter, all housed in AEC 410. A row of outlets in room AEC 400 provides power from the inverter using battery storage or a combination of PV power and batteries depending on the amount of energy required by the load. The system is monitored and controlled by the SCADA Interface Box (SIB). A safety system opens relays in the Raw Power Interface (RPI) and Switch Controller (SC) in the event of a system fault. A system shutdown can be scheduled through the SCADA system. DC power is converted into 120V 60Hz AC power through the Filter & Inverter Box (FIB).

2 System Components

This section describes all of the main components of the LPRDS-BMS.

2.1 DC Transfer Switch

The DC Transfer Switch can supply power from the PV array to the Sunny Boy inverter, the LPRDS-ETS system, or disconnect it completely. When the switch is all the way up, it provides power to the Sunny Boy, when it is all the way down it provides power to the LPRDS-ETS, and when it is in the middle both are disconnected. The switch should remain locked in position at all times, and should follow standard Lock-Out Tag-Out (LOTO) procedures.

2.2 Raw Power Interface (RPI)

The RPI provides power from the PV array to Switch Controller. It provides an isolation relay to disconnect the system from the array in the event of a fault or scheduled shutdown. LEDs on the box inform the user of the power and fault status of the RPI.

2.2.1 Indicator LEDs

The RPI system has five LEDs visible from the outside. A yellow high voltage indicator light (RPI11) signifies that there are 30V or more present within the RPI. The green system voltage indicator (RPI10) is lit whenever the RPI is providing power to the rest of the system. The red light (RPI12) is a fault indicator and is lit whenever a fault has occurred and the system is in a fault state. The small yellow light (L1) is lit when 5V power is supplied to the PCB and the small blinking yellow light (L2) blinks when the PIC is programmed and running.

2.2.2 Isolation Contactor/Ground Fault Detector

The ground fault detector (RPI04) monitors the system for the presence of a fault. It has two alarms. Alarm 2 opens the isolation contactor (RPI02) and trips the safety circuit whenever a ground fault occurs. Alarm 1 is a warning that comes on whenever the fault current reaches a percentage of the fault current level (default 50%).

2.3 Switch Controller (SC)

The SC controls the flow of power to or from the RPI, the ESS, and the FIB. A basic block diagram of the SC is shown in Figure 1:

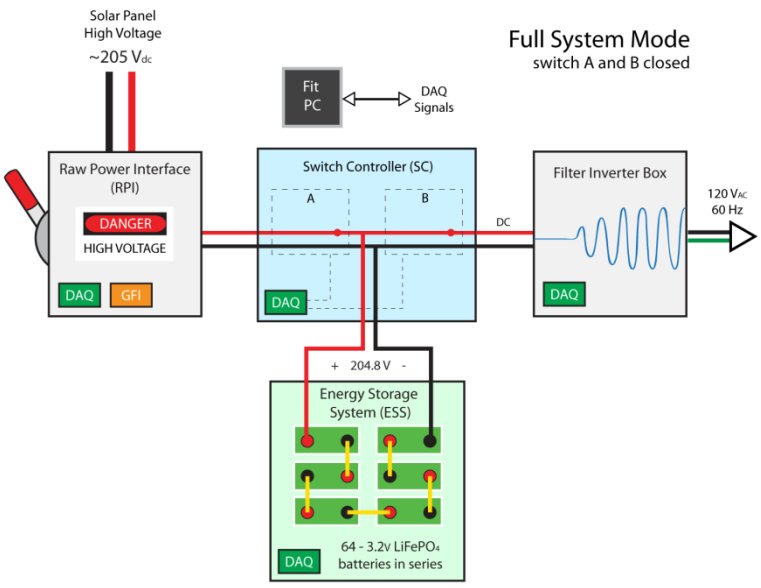


Figure 1: SC Block Diagram

2.4 Filter & Inverter Box (FIB)

The FIB converts DC power from the SC into 60Hz 120VAC using an H-Bridge and a filter.

2.5 Energy Storage System (ESS)

The ESS stores power in batteries to use in powering the system and/or the load. It can supply a total of 2KW of power to the system and load, with a maximum of 75W available for system power.

2.5.1 Batteries

The ESS uses 64 lithium iron phosphate (LiFePO4) battery cells to store power. The batteries have a nominal voltage of 205V. They are 235V when fully charged and can be safely discharged as low as 160V.

2.6 SCADA

SCADA monitors voltages, currents, and temperatures at various locations in the system. This information is monitored for any potential faults and shuts the system down if necessary. Information is stored for later viewing to monitor the performance of the system.

2.7 Transformer

The one-to-one transformer (T1) provides 120VAC to the load and isolates it from the rest of the system. It is connected to the outlet strip on the HV cabinet.

3 Getting Started

3.1 Normal Operation

3.1.1 Starting Up

Before powering the LPRDS-BMS, the system must be cleared of all faults and inspected. As stated in the safety manual (sections 2.7.4 and 3.1), if any high voltage seals are broken, the ground fault system in the RPI must be retested before resuming operation. If any seals are broken, have a qualified person retest the ground fault monitor as outlined in the maintenance manual.

The DC transfer switch should be open or providing power to the Sunny Boy and no power should be provided to the system. Make sure the emergency stop button is pulled into the out position and press the reset (R) button on the ground fault monitor to ensure it is reset.

Close the DC transfer switch by moving the switch to the lowest position to connect power to the RPI. Only the person who has locked/tagged out the system may power the system. The 30V indicator (RPI11) will light if there is 30V or more provided by the PV array, no other LEDs will change. With no load attached, there may be enough residual voltage across the relay to turn on RPI10.

To start up the SCADA software, please see the SCADA ReadMe document (Appendix A).

3.1.2 Powering a Load

Designated outlets in AEC400 and the outlet in the atrium of AEC attached to a steel I beam are powered by the LPRDS-BMS. If there is not enough energy in the system to power a load, the system will disconnect itself from the load in order to recharge the batteries to a higher level. The system will automatically reconnect to power a load when there is enough power in the batteries.

3.1.3 Shutting Down

To stop or shut down the SCADA software, please see the SCADA ReadMe document (Appendix A).

3.2 Emergency Shutdown

The emergency shutdown button, located on the side of the high voltage cabinet, immediately trips the safety circuit and shuts down the system into a safe state when pressed.

Resetting after an emergency shutdown consists of pressing the alarm acknowledge button and making sure the system is safe, before pressing the Clear Fault button on the RPI.

3.3 Clearing Faults

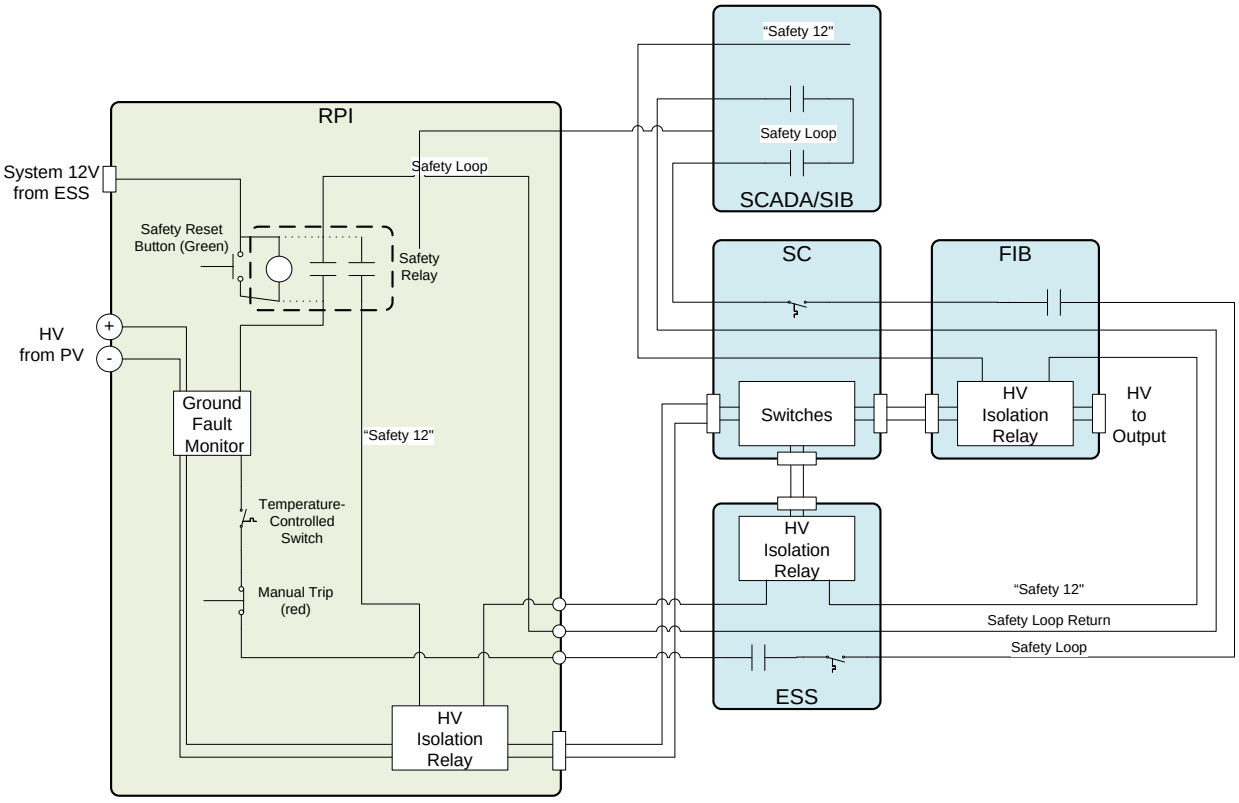


Figure 1: Safety Loop Block Diagram

As shown in Figure 42, the main logic for the safety interface exists in the RPI, but each subsystem is part of the safety loop. The ground fault monitor in the RPI operates by monitoring the current on the high voltage lines and detecting a ground fault if the current entering and leaving do not match. It opens the safety loop if a ground fault is detected.

Temperature sensors in each subsystem open automatically when overheated, breaking the safety loop. Also in each subsystem are controllable relays that can be opened via SW commands, allowing SCADA to break the safety loop if it detects unfavorable conditions internal to a subsystem.

If there are no faults or failures and the safety loop is closed, then the "Safety 12V" created in the RPI is provided to all subsystems. Isolation relays are used wherever high voltage is present (except in the SC, which contains two switches that can also be used to isolate high voltage). These high-voltage isolation relays are normally open, and require the 12V to stay closed. Therefore, if a fault occurs, the safety loop opens, stopping delivery of 12V to the high-voltage isolation relays and breaking all high voltage connections.

On the RPI are two buttons, one green and one red. The red button is a shutdown button that provides a way to open the safety loop manually in case a fault is suspected, or in case the system fails to automatically detect a fault. The green button is a safety reset button, which clears any faults and allows the safety logic to operate. If a fault is detected, the safety loop will be open and "Safety 12V" will not be

delivered, but the safety loop logic is still operating—i.e., once the fault is cleared the system will continue to operate. If the safety loop is broken manually with the red shutdown button, however, it must also be cleared manually with the green safety reset button for the system to continue operating.

On a fault, the SCADA software should have entered the HV Unsafe state. To leave this state, the user needs to run the Clear HV Unsafe application. For more information on starting and stopping this application, please see the SCADA ReadMe document (Appendix A).

4 FAQ

What solar panels are used and how are they installed?

The PV Array consists of 10 GEPVp-200-MS modules installed on the roof of Acopian Engineering Center. These modules are connected in a series configuration and placed at a 5 degree angle facing south. The IV curve for these panels is provided in Figure 1:

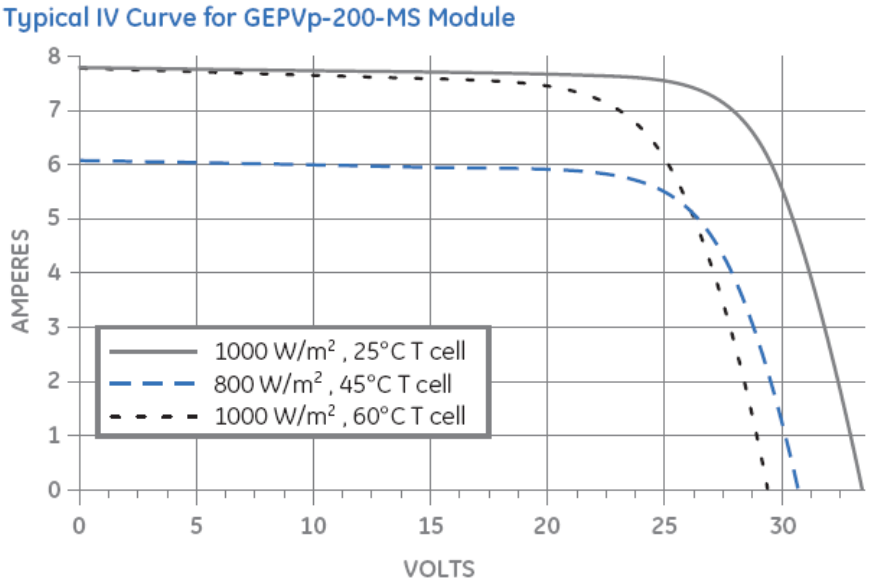


Figure 1: IV Curves for GEPVp-200-MS Solar Panels

What are the voltages of the various components of the system, such as the PV Array and the batteries?

Each PV module is capable of providing over 30V in optimal conditions; however, in nominal conditions, each panel will provide approximately 25V for a total series voltage of 250V.

64 LiFePO4 are connected in series, each having a charged voltage of 3.2V. The batteries are connected in series in the ESS rack such that when maintenance is performed on them, only four are connected for a maximum voltage of 12.8V per cell. When operational and fully charged, the entire ESS rack provides about 205V @ 10Ah, or 2kW of power.

What is the power budget available to both the system and the load?

The LPRDS-BMS provides approximately 75W for system power (SC, SIB, etc.), and about 2kW for a load.

4.1 Calibration

The various voltage and current sensors used in the LPRDS-BMS have been characterized and compensated for offsets and non-linearities. No additional calibration is required for the system.

These compensations can be found in the Hardware XML file which is used by SCADA. For each sensor, a scale and an offset is specified, so that the actual voltage or current reading can be found from the equation Value = Scale * (Sensor Reading) + Offset.

5 Maintenance

Refer to the Maintenance Manual for proper maintenance procedures not discussed in the User Manual.

5.1 Battery Maintenance

The batteries used in the LPRDS-BMS are Lithium Iron Phosphate Batteries (LiFePO4). These batteries do not need periodic maintenance like Nickel Metal Hydride or Lead acid. Their chemistry is not explosive and stable even under under/over charge situations. A typical service time for LiFePO4 is about 6-7 years or 3000 cycles, depending which comes first. If a battery is cycled once per day, it would last for over 8 years not accounting for capacity loss due to just aging (>3000 cycles).

Even if there are no specific maintenance instructions for LiFePO4 batteries, there are a few good rules of thumb when it comes to battery conditioning and maintenance:

- Cycle the battery a few times before using for the first time
- Store in a cool dry place
- If the memory effect is a problem, cycle the battery a few times every few weeks to undo the effect
- If the batteries need to be stored for a long time unused, then store them in a cool dry place and charge the batteries before their next use to account for their self discharge.

6 Schematics/Drawings

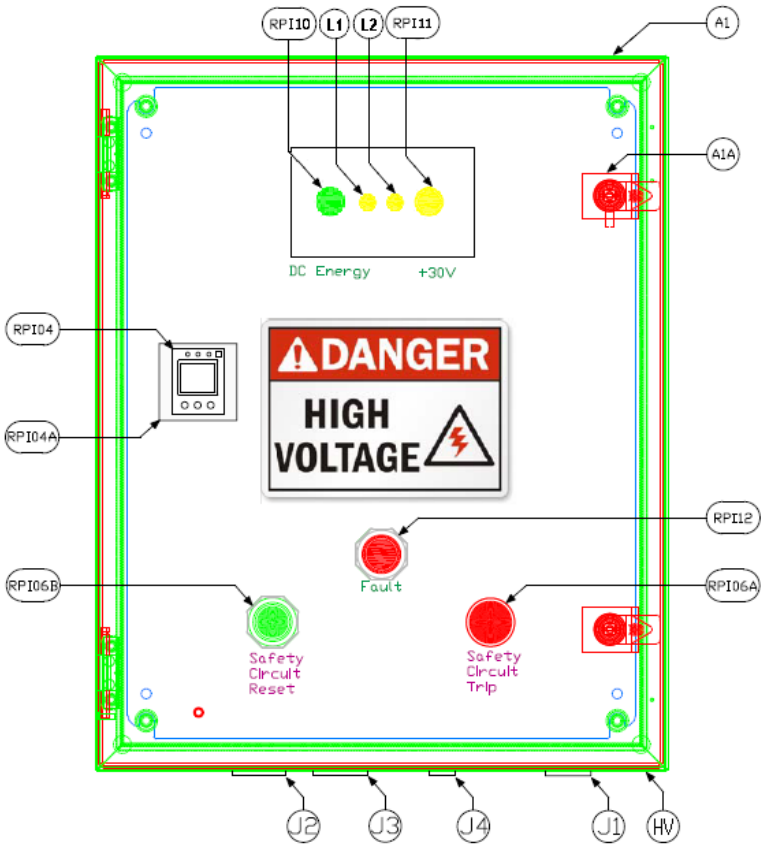


Figure 1: RPI Module

Key to designators in Figure 1:

A1	Enclosure
A1A	Thumb turn handle with key lock
RPI04	Ground fault monitor - Trips the safety relay in the event of a ground fault. Alarm 2 (Al2) goes off when the ground current exceeds the tripping point, Alarm 1 (Al1) goes off at half this level.
RPI04A	Mounting frame for the ground fault monitor.
RPI06A	Safety circuit trip button - Trips the safety circuit.
RPI06B	Safety circuit reset button - Resets the safety circuit.
RPI10	System power indicator - Lights when the RPI relay is closed and power is being provided to the system.

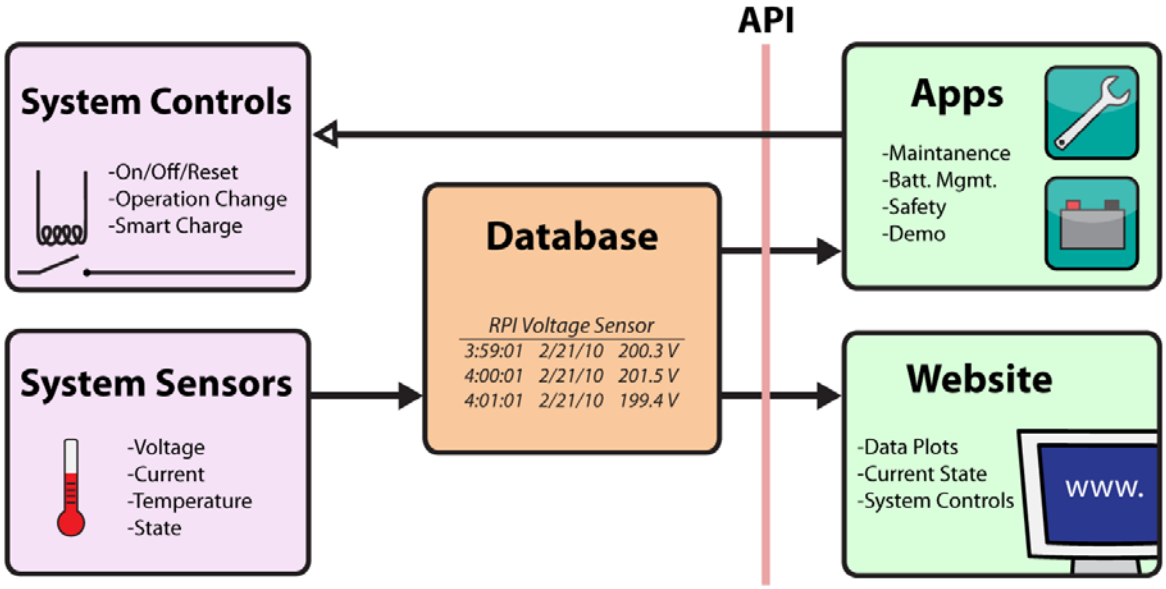
RPI11	30V indicator - Lights when 30V or more is being supplied by the PV array.
RPI12	Fault indicator light - Lights when there is a fault and the safety relay is tripped.
L1	PCB Power light - Lit when the PCB has 5V power supplied.
L2	PIC light - Blinks to show PIC is programmed and running.
J1	High voltage output receptacle
J2	Supply voltage receptacle
J3	Safety receptacle
J4	RJ45 receptacle
HV	High voltage input

Appendix A – Scada ReadMe

1 SCADA Software

1.1 Goal

The purpose of the Supervisory Control and Data Acquisition (SCADA) software was to allow monitoring and control of the Lafayette Photovoltaic Research and Development System (LPRDS). The software would run on a computer called the FIT PC which is located in the LPRDS Tower. One general requirement was that all monitoring and control interactions with the system from the applications would be through a defined Application Programming Interface (API). For monitoring, the software had to provide the ability to poll Voltage, Current, and Temperature readings and store them in a Database, which could be accessed by applications as well as by a website. For control, the software had to provide the ability to control the switches and relays in the system from applications. Here is a simple block diagram of our initial view of the software to control the system:

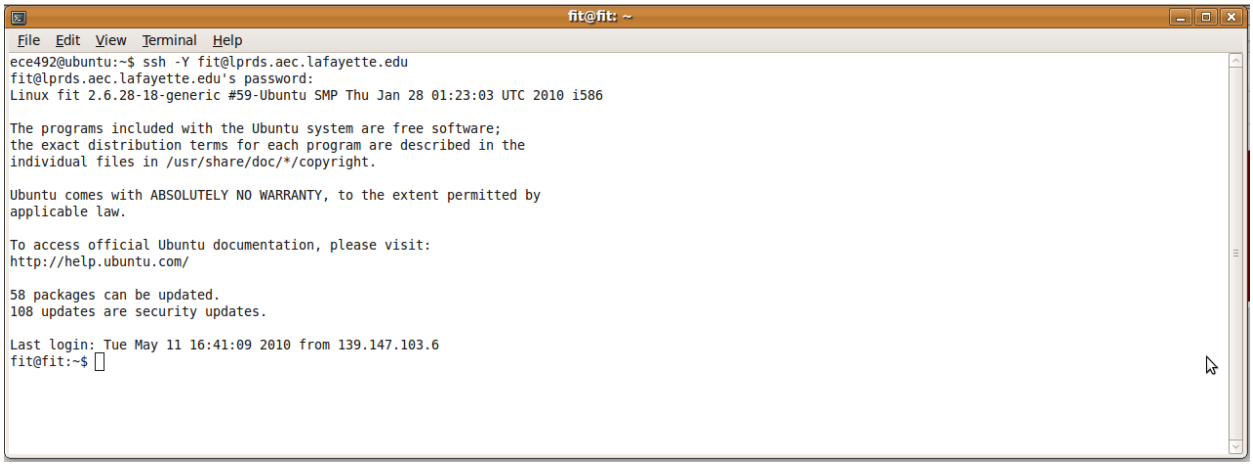


1.2 Basic Ubuntu Commands

The software is running on the FIT PC, which has Ubuntu 9.04 installed. For interacting with the software, there are various useful terminal commands listed below.

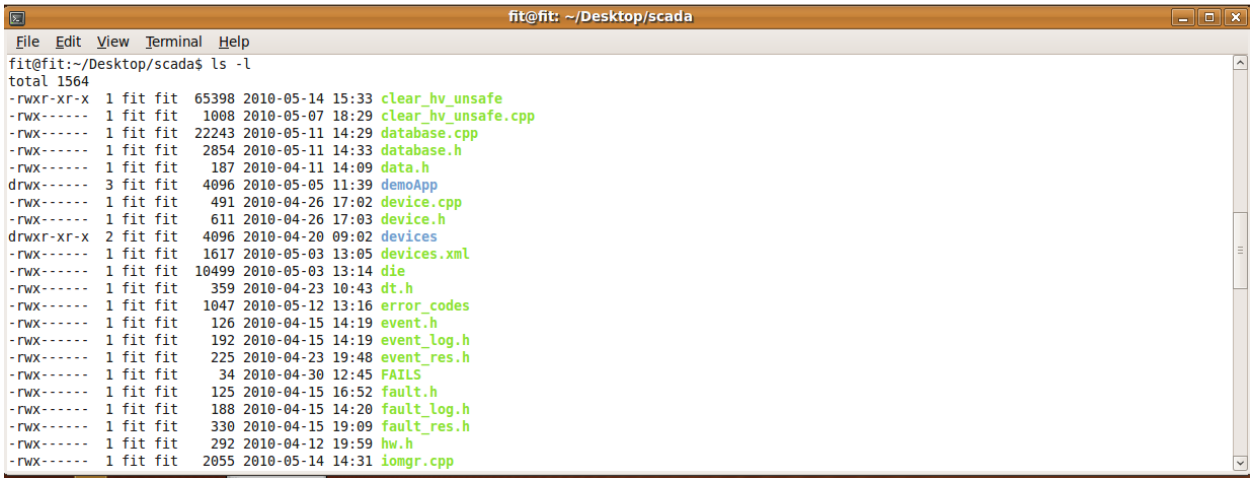
1.2.1 ssh –Y *address*

This command allows a user to connect to another computer using the Secure Shell protocol. For example, if the user wanted to connect to the FIT PC, the command would be *ssh –Y fit@lprds.aec.lafayette.edu*. This is essentially the same as opening a terminal on the FIT PC itself, which can be useful for interacting with the software and the system from outside the project room. The password for the FIT PC is 111111. An example is shown below.



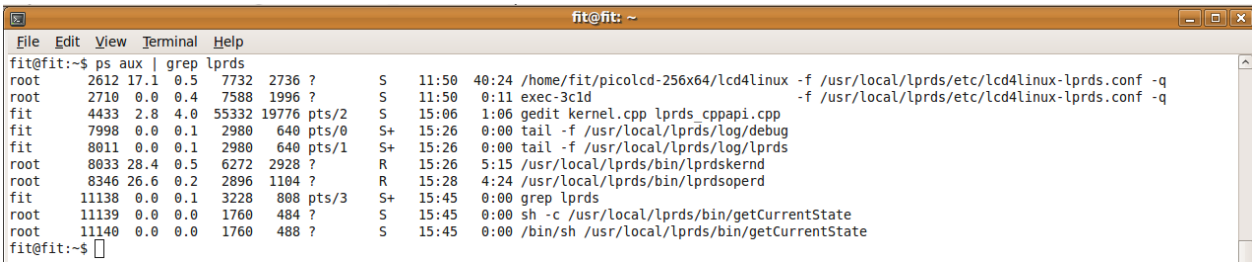
1.2.2 ls

This command lists all of the files in a directory. Adding –l switches it to a “long listing format”, essentially putting in a list format.



1.2.3 ps aux | grep *searchterm*

This command allows a user to search the list of all running processes for a specific term. For example, if the user wanted to see if any process with “lprds” in the name was running, the command would be *ps aux | grep lprds*. The command to view all running process is *ps aux*, and the command to search for a specific term is *grep searchterm*, and the | “pipes” the results of *ps aux* into *grep searchterm*. An example is shown below.

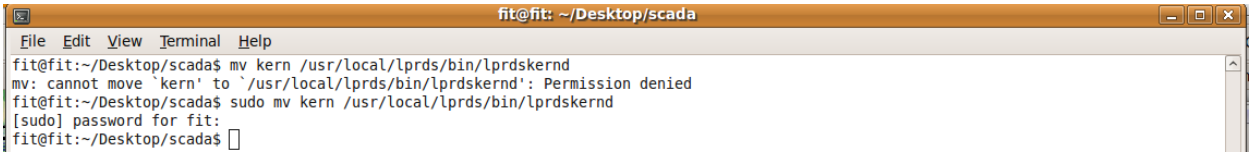


1.2.4 mv *fromLoc toLoc*

This command moves a file or files from the *fromLoc* to the *toLoc*. For example, if you were in the /Desktop/scada directory, and typed *mv kern /usr/local/lprds/bin/lprdskernd*, it would move the file “kern” in the /Desktop/scada directory to the /usr/local/lprds/bin/ directory, and would rename it to lprdskernd. An example is shown in the *sudo* section below.

1.2.5 sudo *command*

Putting *sudo* before a command allows the command to be run with root access (runs it as administrator). For example, if the directory you are trying to move a file to has restricted permissions, the mv command might give you a permissions error. However, if you put “sudo” before it, it won’t give you an error about permissions. Sudo often asks for a password before executing the command, which, on the FIT PC, is 111111. An example is shown below.



1.2.6 tail *filename*

This command shows the last 10 lines of the file specified by *filename*. If you add the –f option, it continues to show the end of the file. An example is shown below.

1.2.7 apt-get install *packagename*

This command is used to install many packages in Ubuntu. It normally requires *sudo* before the command. For example, if you were to type *sudo apt-get install randomPackage*, it would install *randomPackage*.

2 Starting, Stopping, Compiling the Software

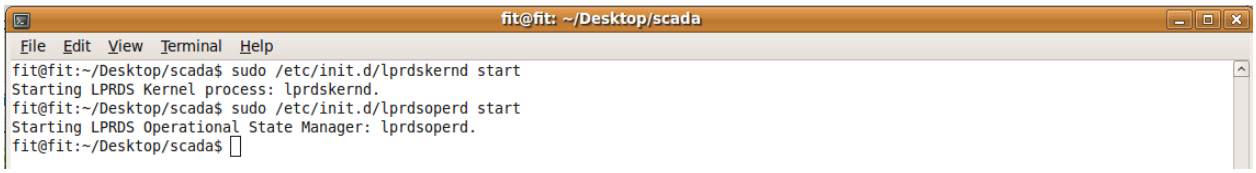
2.1 Starting and Stopping Kernel and the Operational State Manager

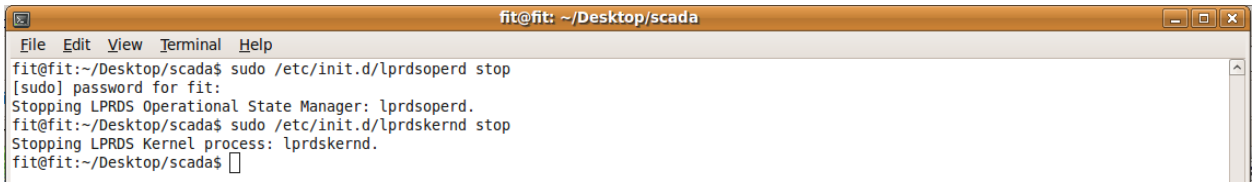
Both Kernel and the Operational State Manager are supposed to automatically start when the FIT PC is turned on. You can check if they are running using the *ps aux | grep lprds* command, and looking for *lprdskernd* and *lprdsoperd*.

Manually Starting and stopping the Kernel or the Operational State Manager is done using scripts located in the /dev/init.d/ directory on the FIT PC. Always check to make sure they are not running before trying to start either Kernel or the Operational State Manager.

The command to start the kernel is *sudo /etc/init.d/lprdskernd start*, and the command to the start the Operational State Manager is *sudo /etc/init.d/lprdsoperd start*. Stopping the kernel or Operational State Manager is done through *sudo /etc/init.d/lprdskernd stop* and *sudo /etc/init.d/lprdsoperd stop*.

Note: Any application, including the Operational State Manager, must be started after Kernel, and must be stopped before Kernel for the software to work correctly.

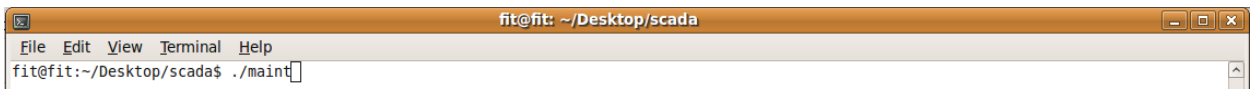




2.2 Starting and Stopping Applications

All applications other than the Operational State Manager should be started from the /Desktop/scada directory on the FIT PC. To start them, you simply need to type *./AppName*. For example, the commands to start the existing applications are: *./maint*, *./mgmt*, *./clear_hv_unsafe*. Keep in mind, all applications require Kernel to be started in order for them to work properly.

Stopping the applications varies slightly between the applications. The maintenance application has an interface, so that the user can simply type “exit” to close the application. The battery management application has no interface, so to stop it the user would need to hold Ctrl and press C to stop it. The clear HV Unsafe application exits automatically after the user input request.



2.3 Compiling the Software

Compiling the software is done through a Makefile located in the /Desktop/scada directory. Before compiling an application, you should make sure that the application is not currently running. Once in the directory, simply type "make *app*" in order to compile the specific application. The *make kern* command will compile the kernel, and should be followed by *sudo mv kern /usr/local/lprds/bin/lprdskernd*. The *make lprds* command will compile the Operational State Manager, and should be followed by *sudo mv /usr/local/lprds/bin/lprdsoperd*. The *make maint* command will compile the maintenance application, and does not need to be followed by a mv command. The *make mgmt* command will compile the battery management application, and does not need to be followed by a mv command. Finally, the *make clear_hv_unsafe* command will compile the clear HV Unsafe application, and does not need to be followed by a mv command.

2.4 Troubleshooting

One possible problem with starting an application can arise from the application being shut down improperly. When this happens, the pipe which connects the application to the kernel will be left, and can interfere with the application starting up correctly. The pipes are located in the /usr/local/lprds/var/ directory, and the application specific pipes have the names "Results*_AppName*". So, if the Maintenance application isn't starting correctly, and typing *ls –l /usr/local/lprds/var/* reveals that the *Results_Maint* pipe still exists, even though the Maintenance Application isn't running, you should remove the pipe with *sudo rm /usr/local/lprds/var/Results_Maint*.

If the programs are running correctly, but the DAQ boards aren't responding, one possible problem is that the SIB is using different USB ports than specified in the xcr.h file. For example, the software may be set to use /dev/ttyUSB0 (for the DAQ boards) and /dev/ttyUSB1 (for the Sunny Boy), while the SIB is set to use /dev/ttyUSB1 and /dev/ttyUSB2. To check the ports the SIB is set to use, type *tail –f /var/log/messages,* and then unplug the SIB from the USB port on the FIT PC, and then plug it back in. In the xcr.h file, the USB port should be set to the lower numbered USB port, and in the yasdi.ini file the Device should be set to the high numbered USB port. If these numbers have changed, the software will need to be stopped, recompiled, and started again.

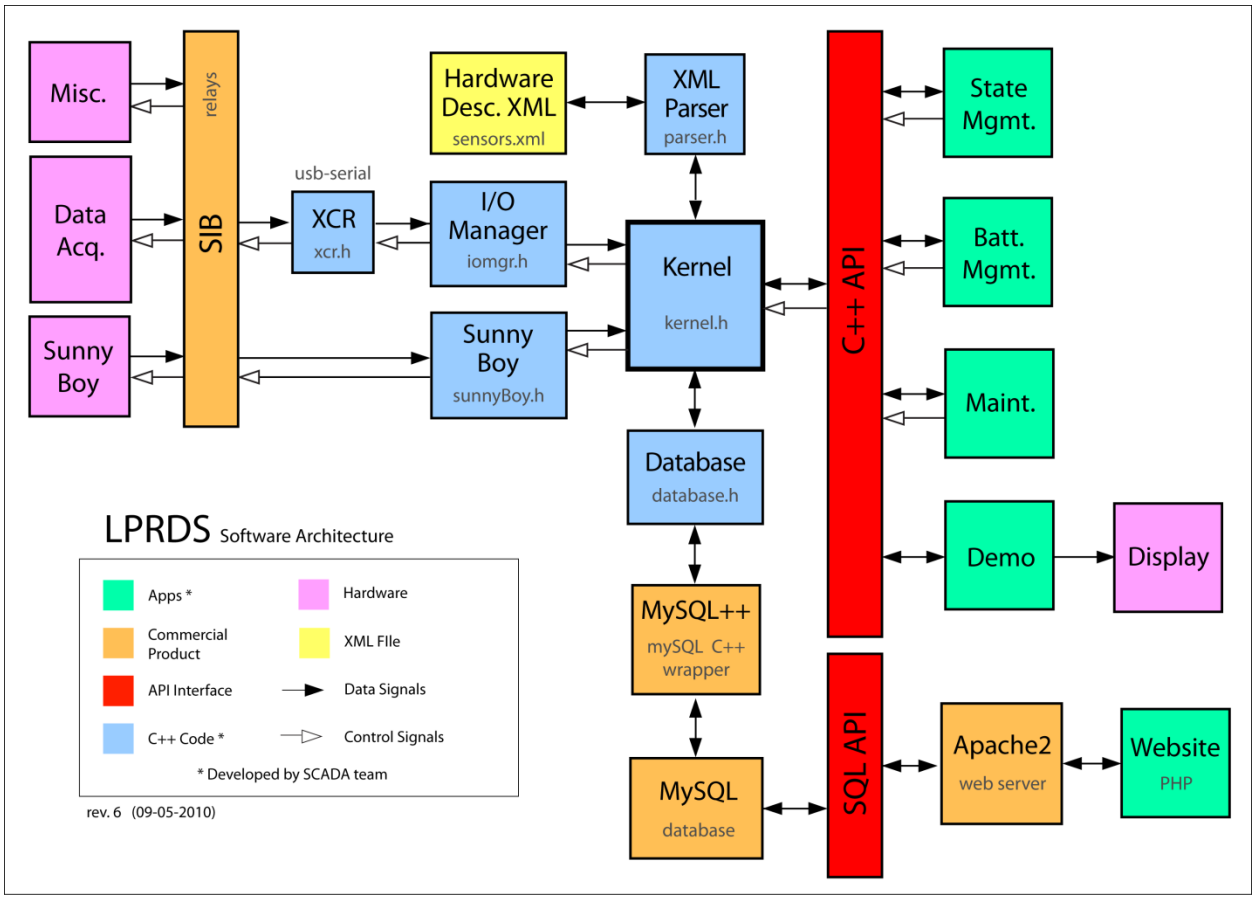
If after using the scripts to stop the Operational State Manager or Kernel, the program is still running (you can still see *lprdsoperd* or *lprdskernd* using *ps aux | grep lprds*), you can still shutdown the process using the *kill* command. This can be used for any application which refuses to shut down. The command to shut it down would be *sudo kill -9 pid,* where *pid* is the Process ID. The Process ID can be found using *ps aux | grep <name>*, it is the number in the second column.

To fully shutdown and restart all LPRDS software, here are the steps to follow:

1. Stop any applications that are running using the methods listed under *Starting and Stopping Applications* and *Starting and Stopping Kernel and the Operational State Manager*. If these methods don't work, then shutdown the processes using the *kill* command mentioned above.
2. Type *sudo rm /usr/local/lprds/var/** to ensure that all pipes have been removed.
3. Start the Kernel, then the Operational State Manager, and then any applications that you want to run (assuming they are allowed to run: Note that the battery management application is only allowed to run in the Operational State).

3 Using the Software

3.1 Software Block Diagram



For a description of the individual blocks, see the SCADA Software Architecture document.

When an application wants to get data from the Data Acquisition boards, it calls the appropriate C++ API function. This request goes through the Kernel pipe (explained in a later section) to the Kernel. Kernel then calls the appropriate function in IOMgr, which formats the packet that needs to be sent to the Data Acquisition (DAQ) boards. XCR is responsible for physically writing the packet to the USB port, which gets sent to the SIB. The SIB passes the packet along to the DAQ boards using RS-485, which have sensors that are controlled by a microcontroller. The microcontroller sends a response back with the requested value, which is passed through the SIB to the USB port. XCR reads the packet, and passes it to IOMgr which decodes the packet, sending only the requested value back to Kernel. Finally, Kernel returns the requested value through Results pipe to the API, and to the application.

The data flow for controlling the digital outputs on the DAQ boards is exactly the same as for requesting data. Sunny Boy requests are also done in a similar fashion, except that the Sunny Boy class is used instead of the IOMgr and XCR.

Every time that data is received from the DAQ boards or Sunny Boy, as well as when switches are set, the data or switch position is recorded in the database. This data is available to the website using PHP to access the database.

3.2 Pipes

The C++ API uses a UNIX feature called pipes to allow communication between the applications and the Kernel, which is constantly running. Pipes are essentially a FIFO between two running processes. One application can write to the pipe, and then another application can read from it. Writing to and reading from pipes is very similar to interacting with a file, as pipes are essentially special files that are optimized for having multiple processes write to them.

In the interactions between the Kernel and the applications, there are two kinds of pipes: the kernel pipe, and the Results pipes. There is only 1 kernel pipe, which is meant to be read by Kernel, and written to by applications. There is one Results pipe for each application, which is created when the application calls the Connect function in the API, and deleted when the application calls the Disconnect function in the API. All pipes are created in the /usr/local/lprds/var/ directory.

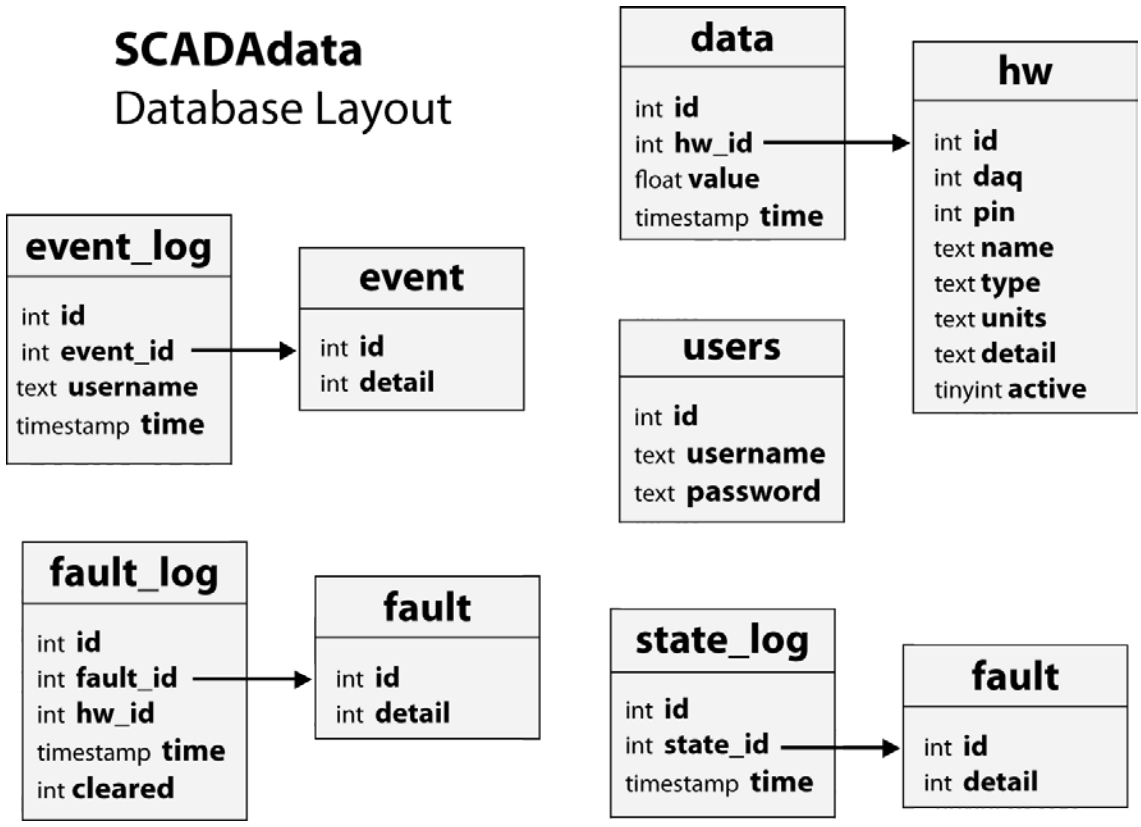
3.3 Database

The LPRDS software has the ability to automatically log system information to a MySQL database. This was accomplished by creating a MySQL database locally on the fitPC and using MySQL++, which is a commercial package that allows communication from C++ to MySQL. The current database can be viewed on the fitPC with a graphical user interface within firefox. A ‘database’ link is available in the bookmark bar, or you can type in the following address:

http://localhost/phpmyadmin/

Username: root	Database: SCADAdata
Password: 111111	Server = localhost

The database is arranged into subsections known as tables with each entry in the table known as a row. The SCADAdata database is arranged into 9 tables, each with an auto-incrementing index named ‘id’. Tables such as event_log, fault_log, state_log and data have additional fields ending in ‘_id’ (event_id, hw_id etc.) that correspond to the ‘id’ number of the table named in the prefix. For example, the ‘hw’ table stores detailed information about each sensor and assigns each a unique index ‘id’ number. When a sensor reading is stored in the ‘data’ table, it only needs to store the matching ‘id’ number from the ‘hw’ table in the ‘hw_id’ field to identify the sensor. This was done to minimize the amount of space used by the database.



3.4 SQL Commands

To manipulate data in the database we issue commands written in the Structured Query Language (SQL). In the LPRDS system, SQL commands are commonly used to retrieve data, write new data, and update previous entries. Below are a few SQL command examples:

*SELECT * FROM hw WHERE daq = 4 AND pin = 7*

The 'SELECT' command is used to retrieve information back from the database. The '*' means retrieve every field from the table. 'FROM hw' specifies the table to pull data from and 'WHERE' is used to filter information by finding entries that match the following specifications.

SELECT data.id, data.hw_id, hw.daq, hw.pin, hw.name, data.value, hw.units, hw.type, hw.active, data.time FROM hw, data WHERE data.hw_id = hw.id

In this 'SELECT' command, information is gathered from two separate tables. To do this, each desired field must be prefixed with its table name, (ex. data.id for the id field in the data table) and 'FROM' is used to specify the tables. In this case, WHERE is used to match up each 'data' entry with its detailed sensor information in the 'hw' table by matching data.hw_id to hw.id.

INSERT INTO state_log (state_id, time) values (3 , NOW())

The ‘INSERT INTO’ command allows us to add a new entry to the database. It is best to specify the fields you intent to populate after the table name within parenthesis, leaving out ‘id’ to avoid problems with the automatically incremented index field in each table. Use NOW() for the timestamp value to write the current time. Any text must be enclosed in single quotes ‘’.

UPDATE hw SET active = 1 WHERE name = 'sensor_name'

To change the value of information already in the database, use the ‘UPDATE’ and ‘SET’ command. The values to be updated can be specified using ‘WHERE’ as seen before. Any text must be enclosed in single quotes ‘’.

ORDER BY event_log.time ASC

The ‘ORDER BY’ option can be added to the end of a ‘SELECT’ command to arrange the results by their timestamp field. Use ‘ASC’ for ascending time and ‘DESC’ for descending.

Tip: When formulating a new SQL command, it is convenient to use the phpmyadmin user interface in firefox. You will find a tab near the top of the page labeled ‘SQL’ where you can type in commands and get feedback if there is an error in the formatting.

3.5 Installing MySQL and MySQL++

MySQL and MySQL++ are already configured on the fitPC and right-most Ubuntu computer along the windows in AEC400. To install MySQL from scratch (as well as PHP and Apache for the website) follow this tutorial:

https://help.ubuntu.com/community/ApacheMySQLPHP

To install MySQL++, run the following commands in terminal:

```
wget http://www.tangentsoft.net/mysql++/releases/mysql++-3.0.9.tar.gz
tar xvfz mysql++-3.0.9.tar.gz
cd mysql++-3.0.9
./configure
make
make install
```

Note: New versions may be found at: http://www.tangentsoft.net/mysql++/

To use MySQL++ in a C++ program, you first will need to modify the Makefile with the following lines

```
CFLAGS := -I /usr/include/mysql -I /usr/local/include/mysql++
LDFLAGS := -L /usr/local/lib -lmysqlpp -lmysqlclient -lnsl -lz –lm
$(CFLAGS) $(LDFLAGS)
```

Then within the .h file include…

```
#include <mysql++.h>
#include <stdlib.h>
using namespace mysqlpp;
```

3.6 Sunny Boy

3.6.1 Using the yasdi library:

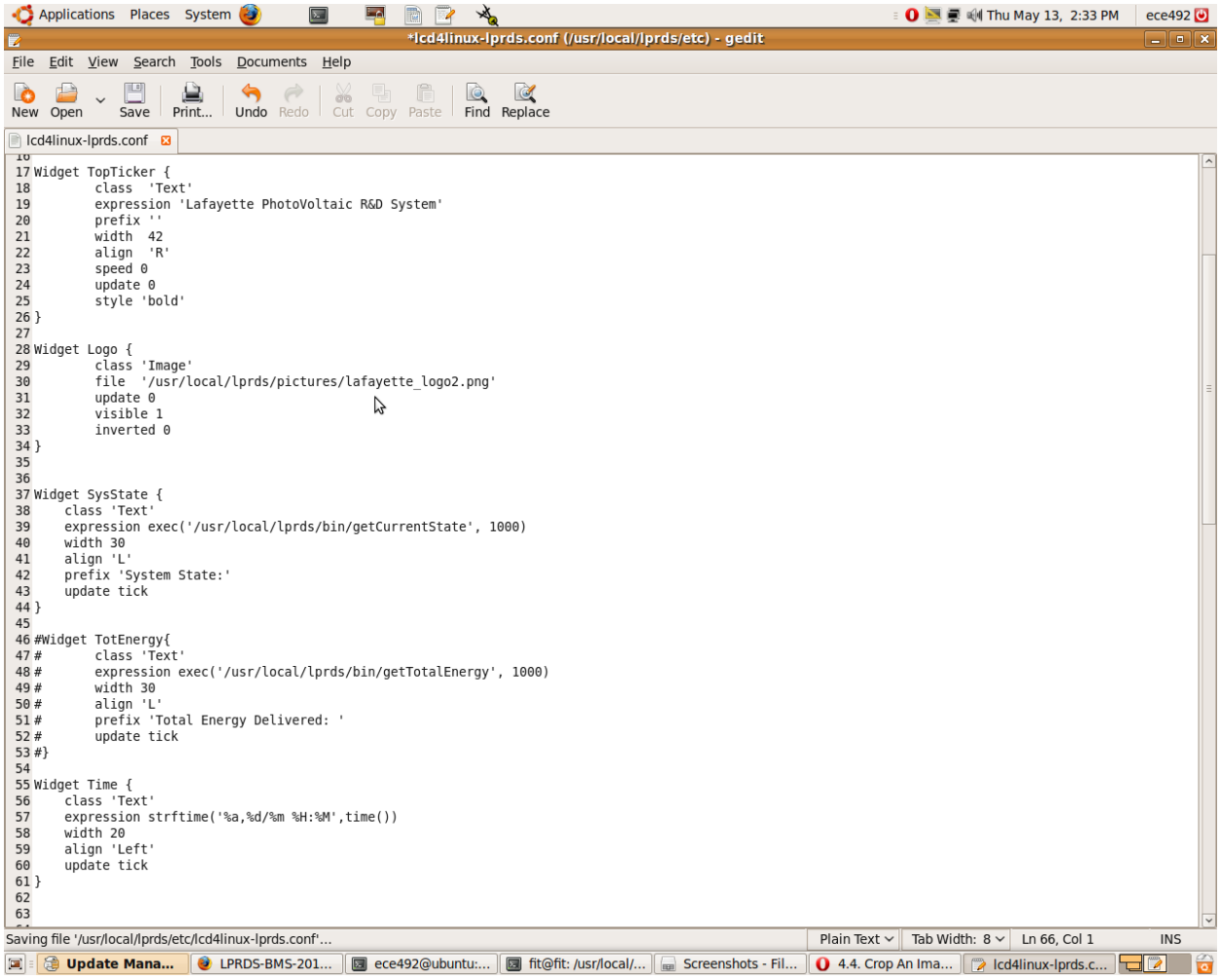
The Sunny Boy communication protocol is implemented using the YASDI (yet another SMA data implementation) library which was provided by the Sunny Boy manufacturer. The Sunny Boy must be connected to the FIT PC in order to be properly communicated with via an RS-485 connection. The current Sunny Boy class utilizes only a few of the provided functions from the YASDI library; however, only a select few are required in order to poll data from it. These methods and their definitions are located within the sunnyBoy.cpp file. Since these methods draw upon the YASDI library, those libraries must be compiled along with the Sunny Boy code. Consult the Sunny Boy Communication Manual to see how to use this code to communicate with the Sunny Boy.

3.7 Pico-LCD

3.7.1 Viewing the configuration file and the contents:

The file that determines the layout for the Pico-LCD is located in /usr/local/lprds/etc/, the file is named lcd4linux-lprds.conf. By modifying this file, the displayed elements of the Pico-LCD screen can be changed. In the beginning of the file there are various widgets that are currently being displayed on the Pico-LCD screen. There are additional widgets available to add and a list of available widget is

located at http://ssl.bulix.org/projects/lcd4linux/wiki/Layout along with their respective fields and examples. Near the bottom of the screen is the layout which describes where the widgets will be displayed on the Pico-LCD screen. Here is an example screenshot of how the widgets are put into the file:



The exec expression executes a terminal command and prints the result on the LCD screen at a refresh rate specified by the second argument.

3.7.2 Starting and Stopping the configuration file:

The configuration file mentioned above is currently set to be run on startup so when the fit pc is booted up the layout described in the file will be displayed. The startup script, lcd4linux, located in the /etc/init.d directory on the FIT PC, is responsible for calling the correct commands to configure the layout. It has a start and stop command which can be called manually from the terminal by *sudo /etc/init.d/lcd4linux start* and *sudo /etc/init.d/lcd4linux stop*. These are useful if the configuration file has been changed and the changes are desired to be displayed immediately.

4 Package List

Here is a list of the packages that were installed on the FIT PC to run the SCADA software. Many packages can be installed using the Synaptic Package Manager found in the System>Administration menu.

GNUPLOT	MySQL++	G++
php5-mysql	mysql-server	mysql-client-5.0
phpmyadmin	mysql-server-core-5.0	mysql-client
mysql-common	mysql-server-5.0	libmysqlclient15off
libqt4-sql-mysql	libsoci-mysql-gcc	libmysqlclient15-dev