Poly_Sense:
Modular Wireless Sensor Network

By:
Ian Andal, *Electrical Engineering*
James Shirley, *Computer Science*
Haleigh Vierra, *Electrical Engineering*

Senior Project

California Polytechnic State University
San Luis Obispo
Spring 2013

Table of Contents

List of Figures and Table(s)

Acknowledgements

The team would like to thank the staff and faculty of the Electrical Engineering and Computer Science departments for guiding and educating us during our time at Cal Poly. In particular, the team would like to especially thank Dr. Bridget Benson, for offering advice and direction when things were difficult, and Dr. John Ridgely, for providing a software stack for the microcontrollers used in this project.

Abstract

Poly_Sense provides a platform for developers to use for a wide range of Wireless Sensor Network (WSN) applications. This modular system supports different sensing applications by allowing the developer to easily change between supported sensors through the graphical user interface (GUI). The platform also allows developers to integrate new sensors by writing device drivers which follow the platform's guidelines and utilize the application programming interface (API). This low-power and cost-effective wireless solution not only provides a basic platform for entry-level developers, but also accommodates larger-scale applications.

I. Introduction

Wireless Sensor Networks (WSN) are continually expanding in technology and applications due to the increased need to collect data remotely using a wireless system. Many of these applications include monitoring temperatures, detecting light, measuring accelerations, etc. Implementing a WSN can prove to be a challenge because of the various aspects of a WSN architecture. These include determining the number of sensor nodes the architecture can support, designing a base station to handle and process the data from the nodes, and creating software that can communicate with the base station and interface with the user. There are products on the market that aim to solve these issues by offering a custom platform for specific WSN applications. However, these products can drive up the costs of the WSN architecture and require customers to have extensive technical knowledge. The goal of this project is to offer a cost-effective solution to these problems by creating a WSN platform that adapts to the user's WSN architecture and requires minimal technical knowledge. The project will achieve this by creating a modular system that can accommodate a wide range of sensors such as temperature, light detection, vibration, etc and providing a simple user interface to configure the network.

II. Background

A Wireless Sensor Network (WSN) is an ad hoc network that does not have a fixed infrastructure and consists of wirelessly connected sensor nodes, one or more base stations, and a host computer that interfaces with the base station as seen in **Figure 1**. Typical applications for a WSN include deployment in an environmental setting and measurement of physical characteristics within the environment [5].

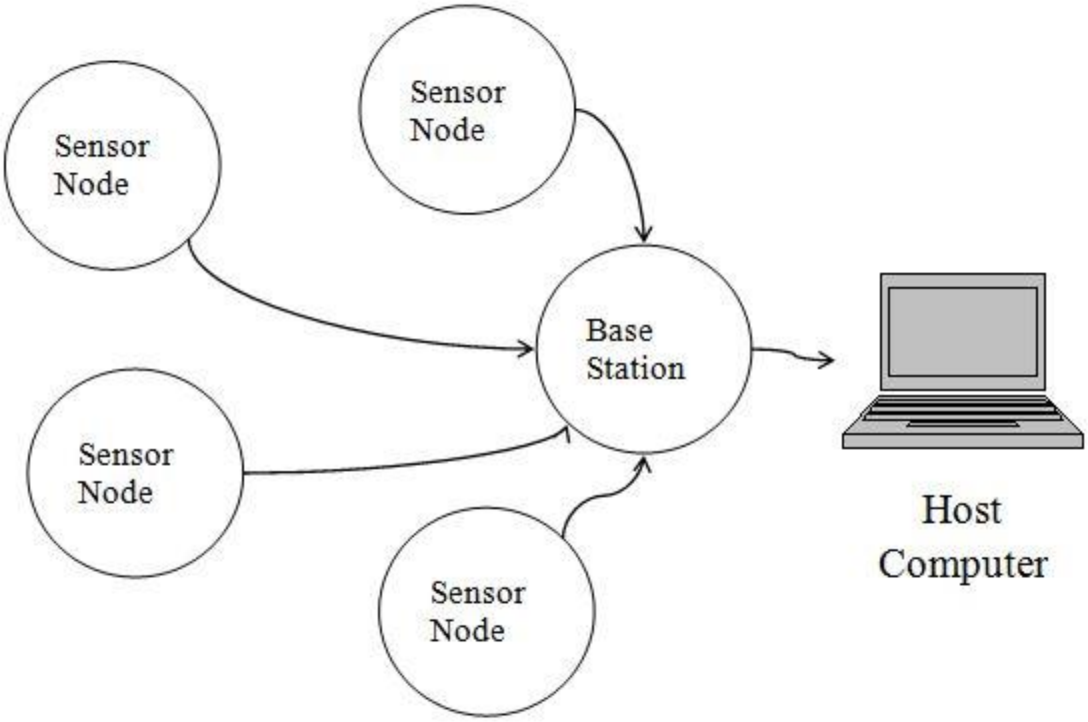


Figure 1: WSN Architecture

The key module of a WSN is the sensor node. **Figure 2** shows the fundamental building blocks of a WSN node. The roles of each sensor node are to acquire data, process data, and communicate data wirelessly to a base station. To accomplish this, each node contains one or more sensors, an RF module, a data processing unit (typically a microcontroller), and an external power source. See a list commercial and prototype sensor nodes available in the reference on the last page [7].

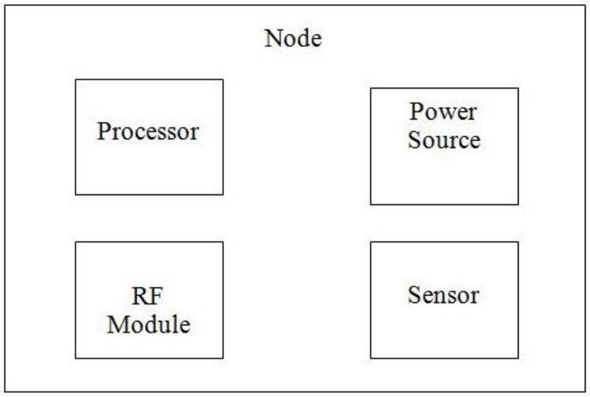


Figure 2: WSN Node Building Blocks

Another fundamental feature of WSN architecture is the base station. **Figure 3** shows the building blocks of a WSN base station. The base station acts as a central hub for the sensor nodes. Typically, sensor nodes have low power constraints and therefore have limited data processing power. This calls for the need of a base station to handle and process sensor data. The base station has a processor that can process the incoming sensor data. The base station can also export this data through a serial protocol to the host computer for the user to analyze. Like the sensor nodes, the base station consists of an RF module to communicate with each node.

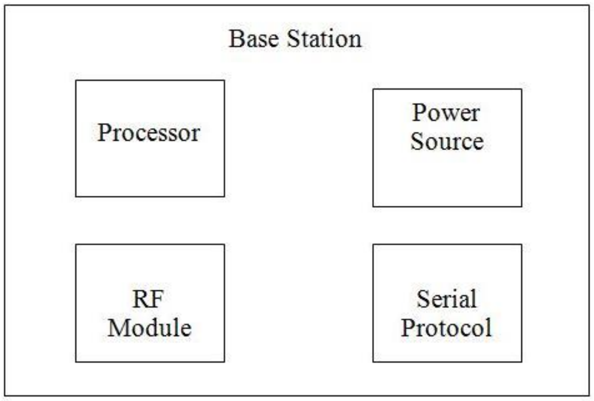


Figure 3: WSN Base Station Building Blocks

The final component of WSN architecture is the host computer. A host computer is an essential part of the analysis of the sensor data because it interfaces the user with the WSN. The host computer contains software that takes in the processed data from the base station and transforms it into a comprehensible form. Other aspects within this software may include settings for the WSN as well as live updates on the status of the WSN.

III. Requirements

In order to develop a modular wireless sensor network system, certain criteria for the project were defined. First, our system must be able to support a wide array of sensors and sensor types, such as ADC and I2C sensors. The system must also be able to relay sensor data and receive commands / configuration payloads wirelessly (as opposed to over a wired connection). The sensor nodes must be able to operate for long periods of time and therefore must have low power consumption. Another constraint for this project is that the cost to manufacture should be low so that, if it were commercialized, it could be competitively priced. To manipulate the wireless sensor network, a frontend GUI must be developed. The GUI and host program that connects to the base station must also be modular, so that they can operate on separate computers.

IV. Specifications

Board Specifications

The Poly_Sense board will have physical dimensions no larger than 4” x 4”. Two AA batteries will power the board; thus, the board should consume no more 30mA to give the board a 10 hour battery life.

System Level Functionality

The high-level system functionality of the WSN system is depicted in **Figure 4**. The system functions as follows:

1. User sets the settings for the WSN through the host computer using the java GUI
2. The host computer sends a query command to the base station via USB/UART
3. The base station sends a data query to the first sensor node from a list of sensor nodes using the Atmel Lightweight Mesh Software Stack [3]
4. The sensor node gathers data from the environment through I^2C, GPIOs, and/or ADC
5. The data is then relayed back from the sensor node to the host computer via the base station for data analysis using the Atmel Lightweight Mesh Software Stack [3] and serial communication.
6. The process repeats for each sensor node in the list

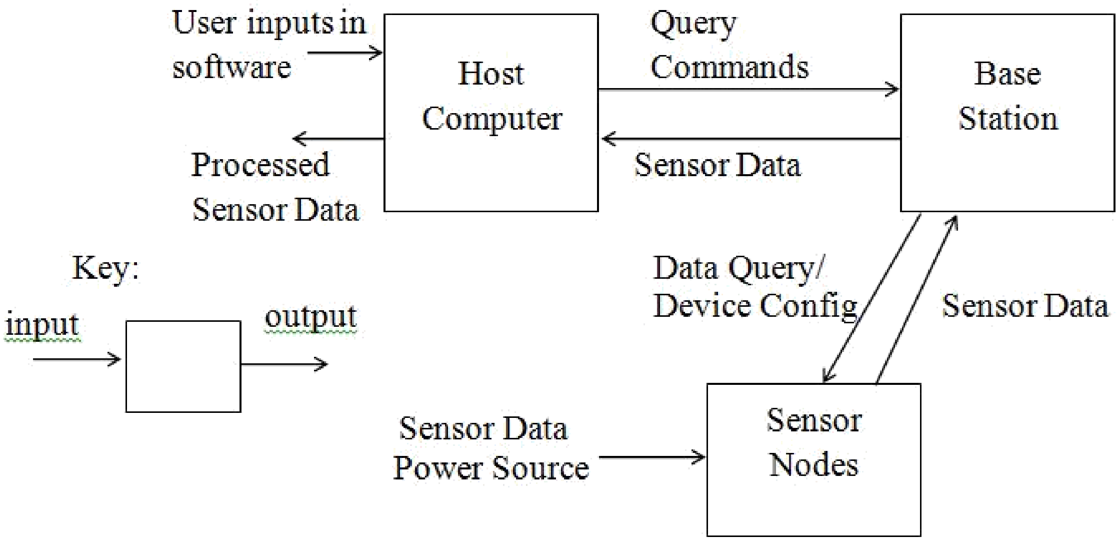


Figure 4: Poly_Sense System Block Diagram

Sensor Node

Sensor Interface

The sensor node PCB allows the user to customize the type of sensor the sensor node contains via the interface. The interface contains a sensor connector to accommodate different types of packages. Using this connector, the user will be able to route I^{2}C, SPI, GPIO, ADC, and power (PSU) connections to the sensor.

Main System

The main system on the sensor node PCB contains all the necessary data transfer and pre-processing components for the sensor node. This main system connects to the sensors via the sensor interface. At the core of the main system is a microcontroller that handles data transfer between the sensor and the RF module. A battery through a power supply unit (PSU) powers the board. The PSU contains boost voltage regulators (3.3V and 5V) to maintain a constant voltages required by all the onboard components. An onboard programmer/debugger will allow a computer to program the microcontroller.

Base Station

The base station handles the incoming sensor data from each node. The base station comprises of the same components as the sensor node PCB, but with additional hardware to interface with the host computer. At the core of the base station is a microcontroller to process the sensor data as well as communicate with the RF module to send data queries and device configurations to the sensor nodes. Also included in the base station is a USB serial communication block that handles data communications between the processor and the host computer. USB power will be used to power the various components on the base station.

Host Computer (Software)

The host computer will contain software that interfaces the user with the WSN. Through the host computer, the user can set the types and number of sensors used in the WSN. The software will also allow the user to export sensor data. These features will be displayed on a graphical user interface (GUI).

V. Design

A. Hardware

1. *Board Design Features*

To accommodate the projected needs of our end users, many design considerations were taken into account. The end design was an effort to provide a wide range of functionality for developers, while maintaining relative simplicity.

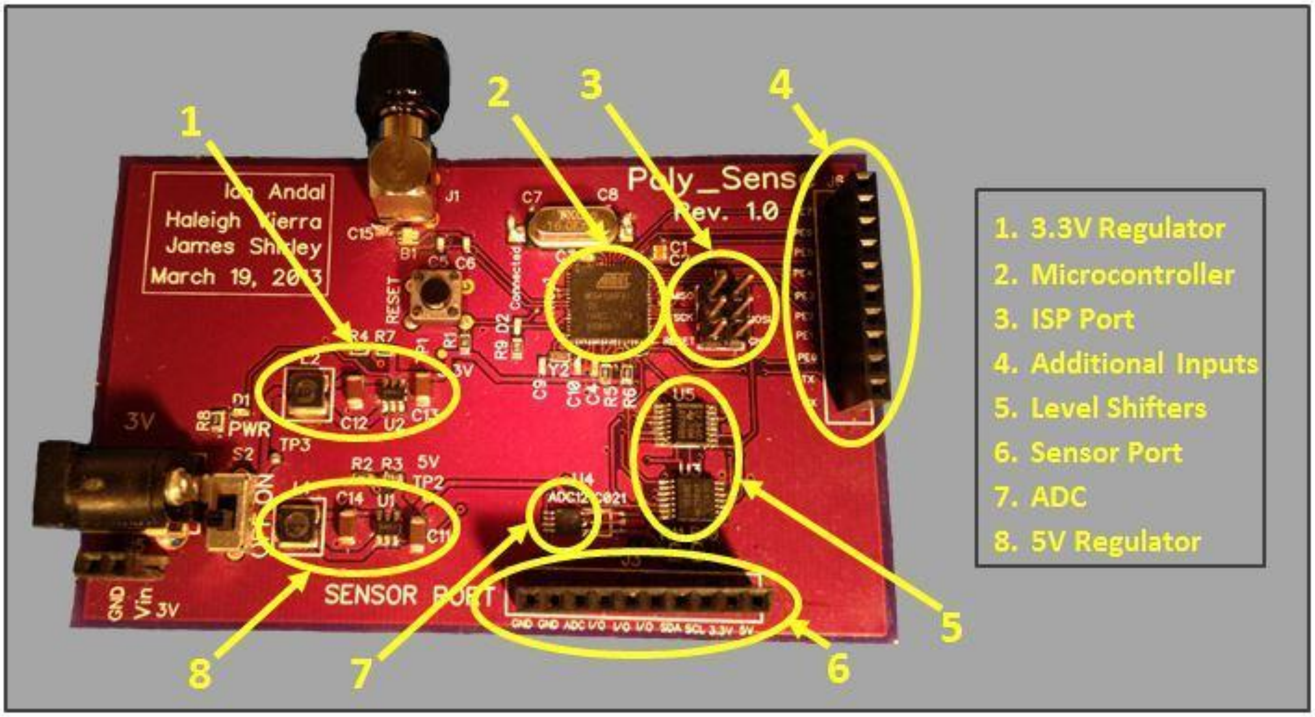


Figure 5: Poly_Sense Development Board Features

a. Sensor Port:

The sensor port was designed to provide access to specific pins on the onboard microcontroller to accommodate the needs of modern digital and analog sensors. Most digital MEMs sensors operate at or below 3.3V, utilize I^{2}C communication protocol, and contain numerous pins with optional features(configurable interrupt pins, synchronization pins, etc.), while most analog sensors operate in the range of 5V and require the use of a simple analog output pin.

The Poly_Sense sensor port was designed with these considerations in mind. The ten pin header seen in **Figure 5** includes pins for 5V and 3.3V power, two ground pins, SCL and SDA pins for I^{2}C communication, an ADC pin for access to the 12-bit onboard analog to digital converter, and 3 general purpose pins which map to GPIO/interrupt pins on the microcontroller.

b. Power Supply:

The Poly_Sense board features two boost regulators (MCP1640B) to create split voltage rails: one regulator creates a 3.3V rail and the other creates a 5V rail. The 3.3V rail supplies the main components on the board such as the microcontroller and the analog switch. The 3.3V rail also connects at the sensor port for 3.3V sensors. The 5V rail supplies power to the ADC and connects to the sensor port for 5V sensors.

c. Level Shifters:

To provide more flexibility for sensor support (such as analog sensors), the board features a bi-directional level shifter that shifts the digital signals from 3.3V to 5V. Specifically, this level shifter converts the I^{2}C data and clock lines and two GPIO's from the microcontroller.

d. Microcontroller:

The Poly_Sense board utilizes the Atmel ATmega128RFA1 microcontroller/RF transceiver to perform all node and base station application processing and wireless communication. This device was selected because it provided a single chip solution for this application's computational and wireless communication needs. Atmel provides developers with an open-source software stack for wireless application development. The Atmel Bitcloud stack, along with FreeRTOS provided the framework for application development on the Poly_Sense base station and node devices.

e. Analog to Digital Converter (ADC):

One additional feature that the Poly_Sense was designed to have was the use on an onboard ADC which could be used to quantize analog sensor readings. The in-chip ADC on the ATmega128RFA1 microcontroller has a very limited input range, so an additional ADC was added to the Poly_Sense design. This ADC is connected to the I^2C bus via level shifters to facilitate communication with the microcontroller. While the physical part is available in the current system, there is no software support for this device yet.

2. Schematic

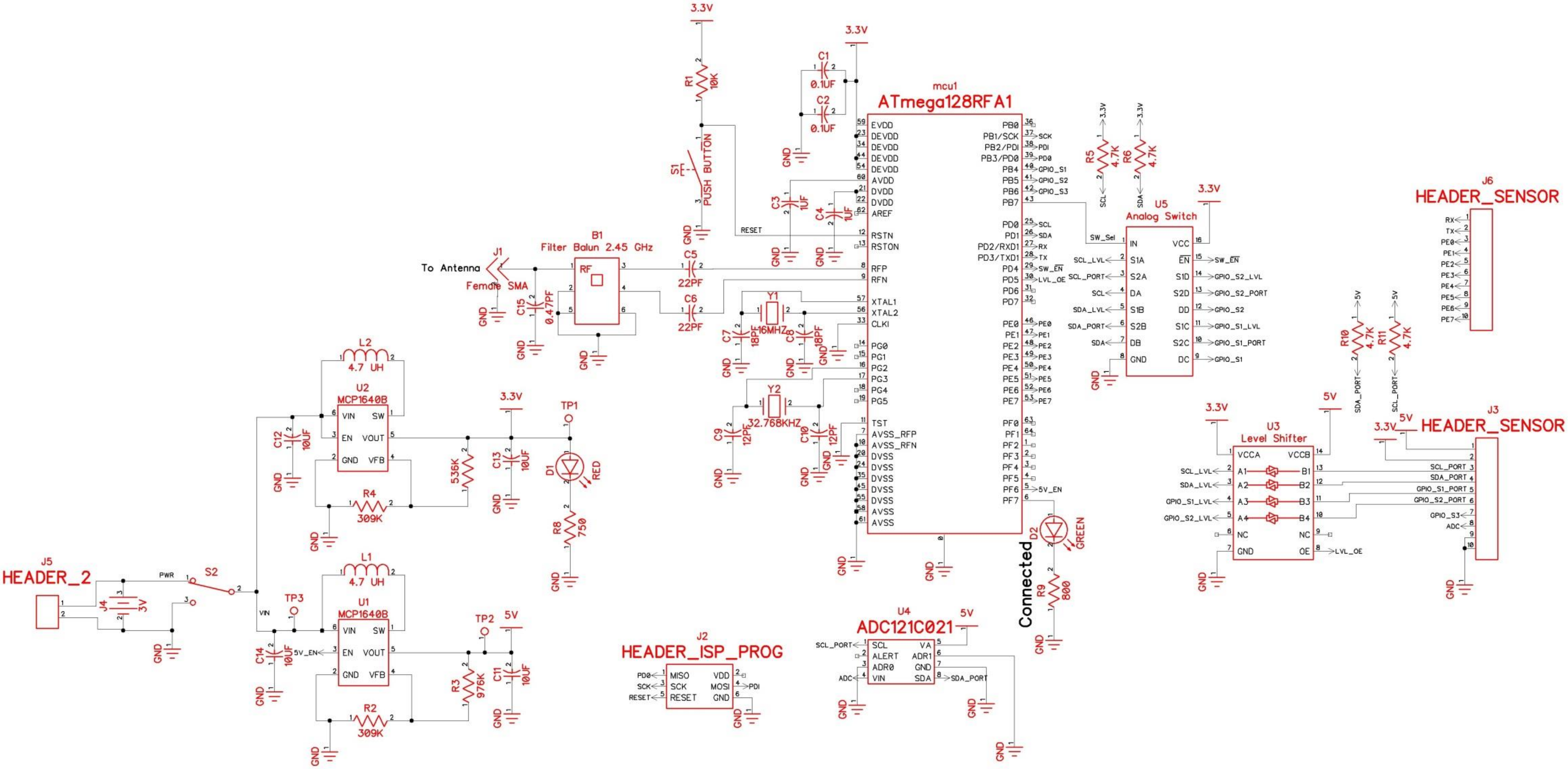


Figure 6: Poly_Sense Schematic

The schematic for the sensor node and base station board is shown in **Figure 6**. The schematic shows the two boost regulating circuits for 3.3V and 5V rails (see U2 and U1). Also included in the schematic is the microcontroller (mcu1) with the radio transmitter circuit (see B1). The schematic also features a pin connection (J2) port for in-system programming (ISP). In addition, the schematic contains a debugging port (J6) which contains RX/TX lines for serial communication. Lastly, the schematic shows the sensor port connections (J3) with several components attached: an analog-to-digital converter (U4), a level shifter (U3), and an analog switch (U5).

3. Part Selection

The components on the Poly_Sense board are surface mount devices with the exception of the DC barrel jack, SMA connector for the antenna, ON/OFF switch, and the pin header connections. This allows for the Poly_Sense board to have relatively small dimensions (2.25” x 3.6”).

The Poly_Sense board features two surface mount crystals, 16 MHz and 32.768 kHz, with external load capacitors used for timing of the internal oscillators of the microcontroller. The microcontroller also has a filter balun set for 2.45 GHz to balance the radio frequency (RF) signals to and from the antenna. This RF transmitter in the microcontroller specifies the correct transmission frequency.

The board also features two low-power LEDs. The red LED, D1, indicates that the board is powered by the 3.3V rail. The green LED, D2, indicates data wireless data transfer by setting PF7 on the microcontroller.

4. PCB Layout

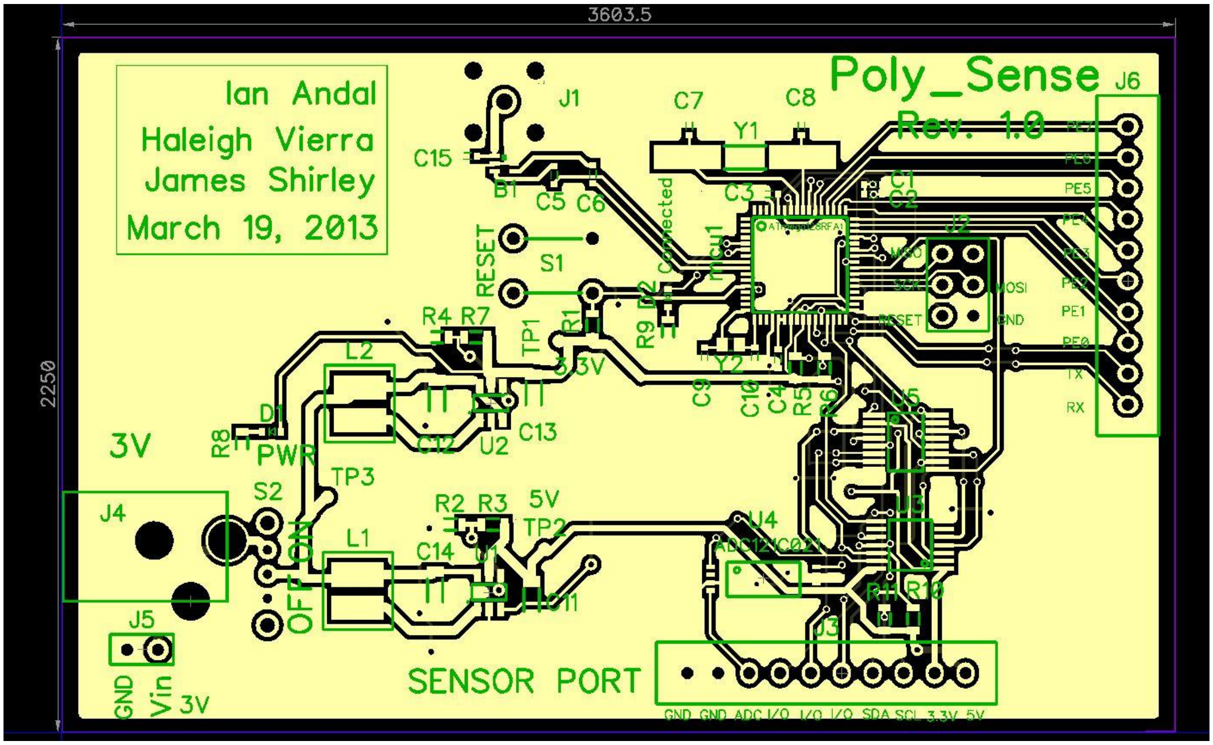


Figure 7: Poly_Sense PCB Top View

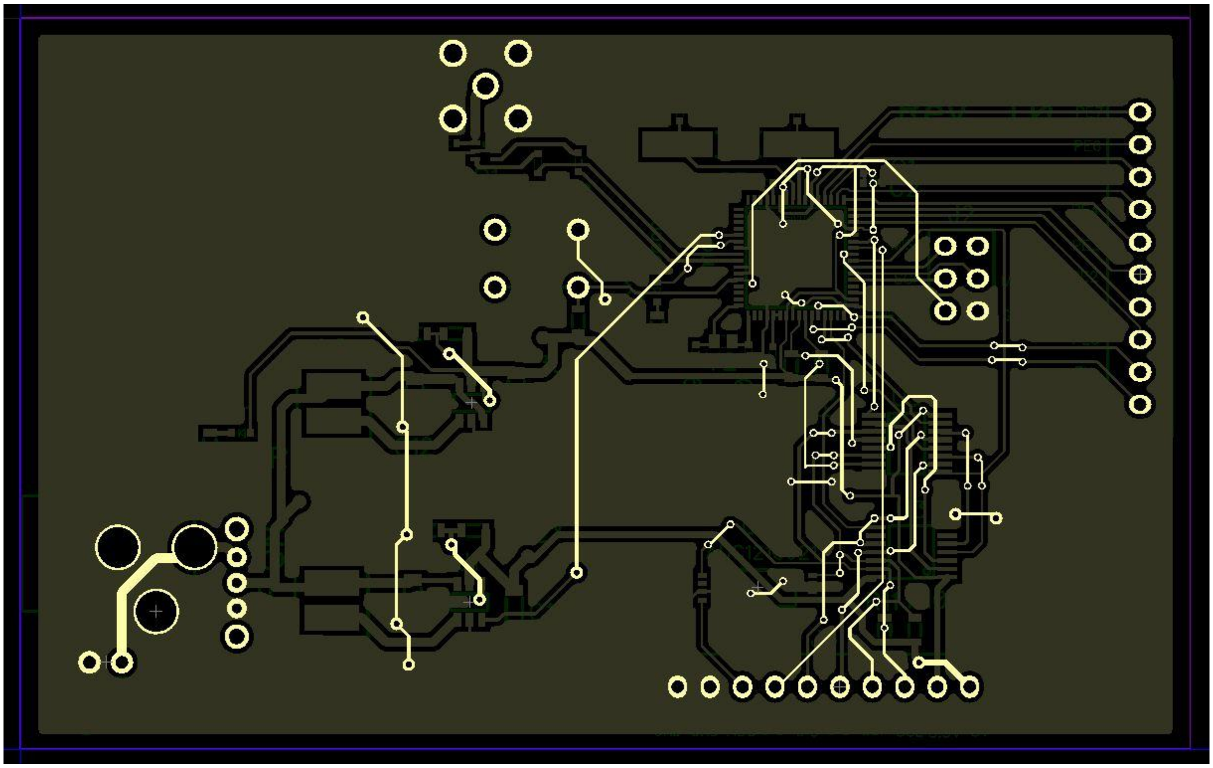


Figure 8: Poly_Sense PCB Bottom View

Figure 7 and **Figure 8** show the Poly_Sense board layout. The board comprises of two layers with the power supply circuits on the left side of the board. To power the board, two AA batteries must connect to either the DC barrel jack (J4) or the header pins (J5). A switch, S2, allows the user to turn the board ON or OFF. The power rail connections utilize thicker copper traces to reduce the resistances and support larger currents.

The radio frequency circuit situates at the top edge of the board. The SMA connector for the antenna faces outwards on the edge of the board to allow the antenna to avoid obstruction and interference from the other components as well as allow easier functionality in connecting the antenna.

The location of the sensor header port near the bottom of the board allows for easy access to the internal communication lines to the microcontroller and allows for users to connect a breakout board that hovers over the Poly_Sense board. This allows for a minimization of the form factor (in terms of overall length and width) of the fully integrated system.

5. Programming

Programming the base station and node boards is conducted using a usbtiny ISP programmer. The 6-pin ISP port seen above in **Figure 5** connects to the programming port on the ATmega128RFA1. Developers can use this port to easily reprogram the base station or node.

6. External Hardware

The current revision of the Poly_Sense board does not have an in-system Serial-USB converter, so all serial communication between the base station and host computer is handled with an external converter. Future revisions of the Poly_Sense board will incorporate the use of an onboard Serial-USB converter.

B. Software

The software used in the *Poly_Sense* platform is designed to be both modular as well as extensible. Since there are many separate components operating in the *Poly_Sense* platform and a large portion of this project is coordinating communication between different devices, below is high-level overview and following that are more elaborate explanations of each component.

1. Overview

The software of the *Poly_Sense* platform includes the following: base station driver, node driver, *Poly_Sense* server and the frontend data collection tool. The modularity of this system allows for data collection runs to be initiated in different ways however a typical system run may be as follows.

1. The user powers on the base station and all desired nodes.
2. The user connects the base station to a host computer and starts the *Poly_Sense* server.
3. The user launches the frontend and connects to the *Poly_Sense* server.
4. The user adds a node and sets its sensor type.
    a. This sends a packet through the system to the requested node and configures the node accordingly.
5. The user starts collecting data.
    a. This continuously sends packets through the system requesting and delivering data.

Communication between these components is done by sending uniform sized packets. The packet is comprised of the following:

1. A 1 byte flag which informs each component how to handle the other data in the packet.
2. A 2 byte address which informs each component which node to send the packet or which node it came from.

3. A 15 byte data array which contains data that will be interpreted differently at each component.
    a. Nodes which receive a packet with a sensor configuration request flag will interpret the data as the sensor it should be configured to.
    b. When the front end receives a packet with a sensor data flag it will interpret the data as values gathered from the sensor and display it on the GUI.

2. Base Station Program

The base station serves as an intermediary device which handles data transactions between the host computer and all wireless nodes. To achieve this goal, the base station requires the use of both wireless communication to perform transactions with nodes and wired communication to interact with the host computer. Using the FreeRtos kernel loaded on the onboard microcontroller, two tasks written in C++ are used facilitate this functionality.

The task_com_port task runs on the base station as the lowest priority task, and handles all serial wired communication with the host computer. The RS232int library is used to handle all read and write operations with the host computer by sending out bytes of data using UART protocol to the external serial to USB converter, which in turn sends the data via USB protocol to the host. This task runs while the task_bitcloud_base_station task is blocked. It waits for a full data packet to be received from the host, and then checks the flag byte. If the flag is a serial data request, then the base station sends back a data packet to the host -- either the same data packet that the host originally sent, or a new packet received from a node. If the flag indicates a sensor configuration request or sensor data request, then the base station puts a boolean true value into a single-element ‘send_packet’ queue which task_bitcloud_base_station is waiting to read. If the flag indicates a debug command, then the packet is sent back to the host, and a boolean true is put into the send_packet queue.

The task_bitcloud_base_station task is the highest priority task which runs on the base station. It is responsible for handling the wireless communication and utilizes the bitcloud wireless lightweight mesh API achieve communication with nodes. Task_bitcloud_base_station waits for a value to be stored into the send_packet queue, and blocks until a value is stored. Once data is received, it uses the value of the boolean as a configuration parameter; true indicates a transmit and false indicates a receive. Then, the state machine as described by **Figure 9** (which is shown and further explained in Section 4.A) is ran until either a wireless transmit or wireless receive is complete. Once the data packet has been sent to the node or host this task blocks until another value is input to the send_packet queue. It should be noted that the base station requires an APP_ADDR of '0' to configure it as coordinator.

3. Node Program

The purpose of the node is to acquire data from a connected sensor based on a wireless command packet received from the base station, and send a packet to the base station containing a sensor reading. Two FreeRTOS tasks written in C++ are run on the node; one is used to facilitate wireless communication with the base station, and another is used to run sensor drivers and acquire data from a given sensor.

The task_bitcloud_node task is used to handle wireless communication. Similar to task_bitcloud_base, it uses functions from the Atmel bitcloud stack and runs the state machine depicted in **Figure 10** (which is shown and further explained in Section 4.B) to conduct wireless transactions. The wireless aspect of this task performs exactly as task_bitcloud_base does, utilizing a single-element boolean queue called 'send_packet ' to block the task, then signal a transmit or receive. One major difference is the use of an additional boolean queue in the APP_STATE_RECEIEVE state: 'get_sensor_data,' which is used to communicate with the task responsible for data acquisition.Once task_bitcloud_node receives a packet from the base station, it determines whether it's a configuration request or sensor data request by checking the flag, then puts a false or true into the get_sensor_data queue respectively. Also, whenever a configuration request is received, the packet data contents have a string containing the name of the target sensor to set up; this string is stored as a global for the data acquisition task to use. This task blocks until another transmission is signal via the send_packet queue. It should be noted that the node requires an APP_ADDR between ‘0’ and ‘32768’ to configure it as router.

The task_data_acquisition task is used to configure a sensor driver, acquire the requested data from it, pack the data, then signal a transmit to the blocked task_bitcloud_node task. This task blocks waiting to read the get_sensor_data queue. If a false is read from this queue, and allocates memory for a new sensor driver object based on the sensor name global, then frees the memory used by the previously loaded driver if necessary. Then a true value is stored into the send_packet queue signaling a transmission is ready and the same packet received, is sent back to be relayed to the host to signal a successful configuration. If a true value is read from the get_sensor_data queue, then the respective data acquisition method from the initialized sensor driver, sensor data is packed. Then a true value is stored into the send_packet queue signaling a transmission is ready, and a packet containing the sensor data is transmitted via task_bitcloud_node.

a. Sensor Drivers

The current Poly_Sense system contains software for two I^{2}C sensors: the Invensense MPU6050 3-axis accelerometer and gyroscope, and the Bosch BMP085 atmospheric pressure/temperature sensor. These drivers were written in C++ and follow the same structural design as to set the standard for future driver development. Both sensor driver classes contain definitions for each of the 8-bit registers which can be accessed from the I^{2}C bus, as well as member variables to store received data, initialization methods, a constructor which calls the initialization methods, and methods to acquire data and store it in the member variables. These classes are descendants of the i2c_master class, which contains methods for I^{2}C read and write transactions on an AVR microcontroller. Task_data_acquisition utilizes pointers to these driver classes to initialize and extract data from these sensors. All sensor drivers are stored on the node.

4. *Wireless Communication*

a. Base Station

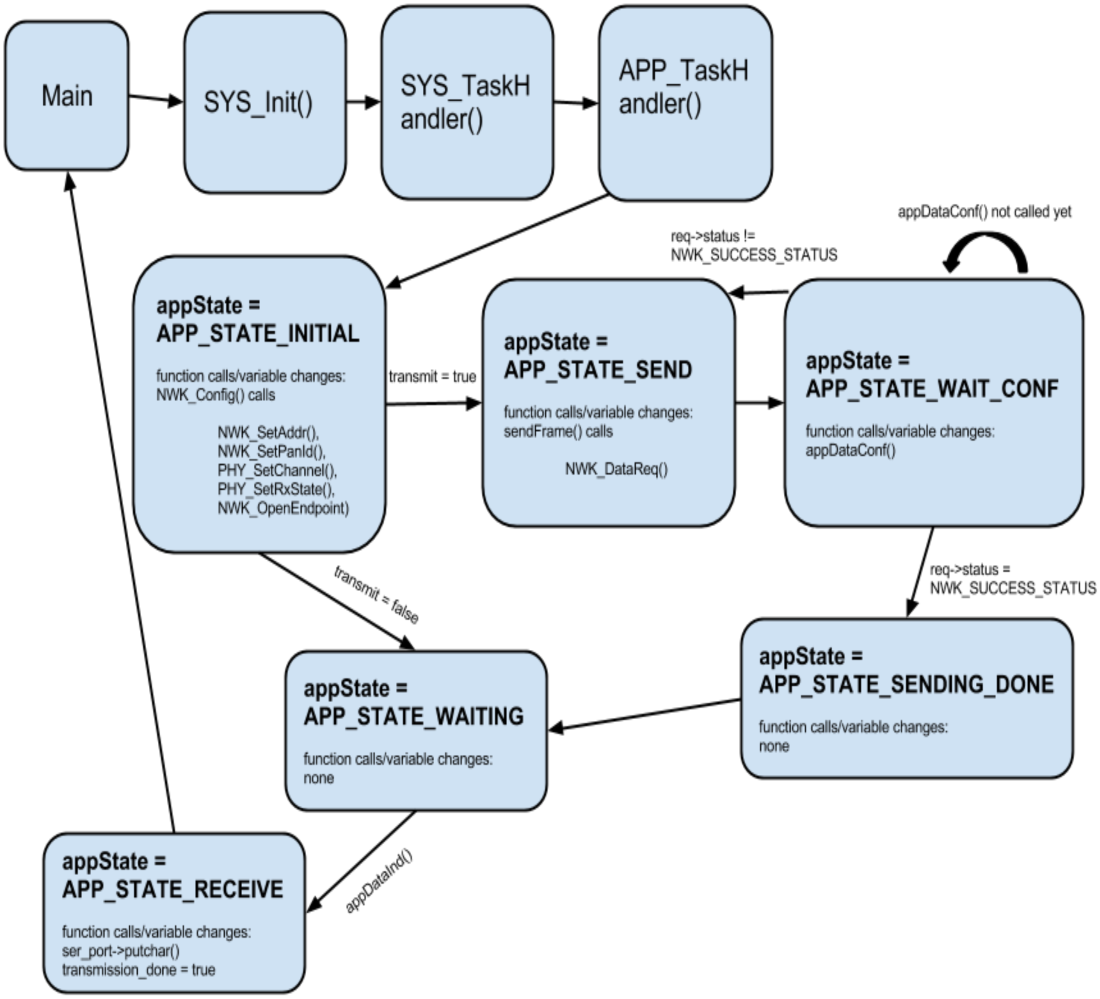

Main
SYS_Init()
SYS_TaskHandler()
APP_TaskHandler()
appState = APP_STATE_INITIAL
function calls/variable changes:
NWK_Config() calls
NWK_SetAddr(),
NWK_SetPanId(),
PHY_SetChannel(),
PHY_SetRxState(),
NWK_OpenEndpoint)
transmit = true
appState = APP_STATE_SEND
function calls/variable changes:
sendFrame() calls
NWK_DataReq()
req->status != NWK_SUCCESS_STATUS
appDataConf() not called yet
appState = APP_STATE_WAIT_CONF
function calls/variable changes:
appDataConf()
req->status = NWK_SUCCESS_STATUS
appState = APP_STATE_SENDING_DONE
function calls/variable changes:
none
transmit = false
appState = APP_STATE_WAITING
function calls/variable changes:
none
appDataInd()
appState = APP_STATE_RECEIVE
function calls/variable changes:
ser_port->putchar()
transmission_done = true

Figure 9: Base Station Wireless Communication State Diagram

Wireless communication between the base station and the sensor nodes is handled by the bitcloud_base_station task. This task uses Atmel's Lightweight Mesh software stack which is a streamline version of Atmel's BitCloud software stack [3]. This software stack works with the Atmega128RFA1 transceiver. A state diagram of the Bitcloud base station behavior is depicted in **Figure 9**.

Within the main function of the bitcloud_base_station task contains a state machine that acts as the core mechanism for transmitting and receiving data. Before the state machine runs, SYS_Init() and SYS_TaskHandler() must run first to enable the lower level functions that control the transmission to works. After these function calls, the function APP_TaskHandler() must execute to run the state machine. The variable appState determines the current state of the base station during the state machine.

The first state, APP_STATE_INITIAL, runs all the network configurations necessary for network communication. This includes setting the network address, APP_ADDR of the base station using the NWK_SetAddr() function. Another function, NWK_SetPanId() sets the network identifier, APP_PANID, that the base station will communicate with. To set the communication frequency of the transmitter, the state calls the function PHY_SetChannel(). Lastly, the state creates an endpoint used for data communication transfer using the NWK_OpenEndpoint() function. Once these functions execute, the base station either goes into APP_STATE_SEND if it will send data or APP_STATE_WAITING if it will receive data.

In APP_STATE_SEND, the base station calls the sendFrame() function for data transmission. This function sets the parameters of the nwkDataReq struct needed for data transmission to one of the sensor nodes. These parameters, which include the node address and the command data, come from the command_pack sent by the host computer. After the function sets these parameters, the base station sends the command frame to the node specified by the node address parameter. The base station then goes into APP_STATE_WAIT_CONF to wait for a confirmation from the sensor node that it received the transmitted data.

In APP_STATE_WAIT_CONF, the base station waits for the appDataConf() call back function to execute. This callback function receives the status parameter in nwkDataReq. If this parameter equals the typedef NWK_SUCCESS_STATUS, the base station goes into APP_STATE_SENDING_DONE. Otherwise, it goes back into APP_STATE_SEND to resend the command frame.

In APP_STATE_SENDING_DONE, the base station will print a message to the serial port if comman_pack.flag equal DEBUG. Then, it will go into APP_STATE_WAITING. In this state, the base station stays idle until the callback function appDataInd() executes. This callback function executes if a sensor node sends a data frame to the base station. The data frame contains either the sensor configuration that the base station sent to it or the sensor data depending on what flag the base station sent to the node. After receiving the data frame from the node, the base station goes into APP_STATE_RECEIVE.

In APP_STATE_RECEIVE, the base station sends the contents of command_pack to the host station via serial communication. Then it exits the state machine by setting transmission_done to true. Next, the base station waits for a command from the host computer.

b. Sensor Node

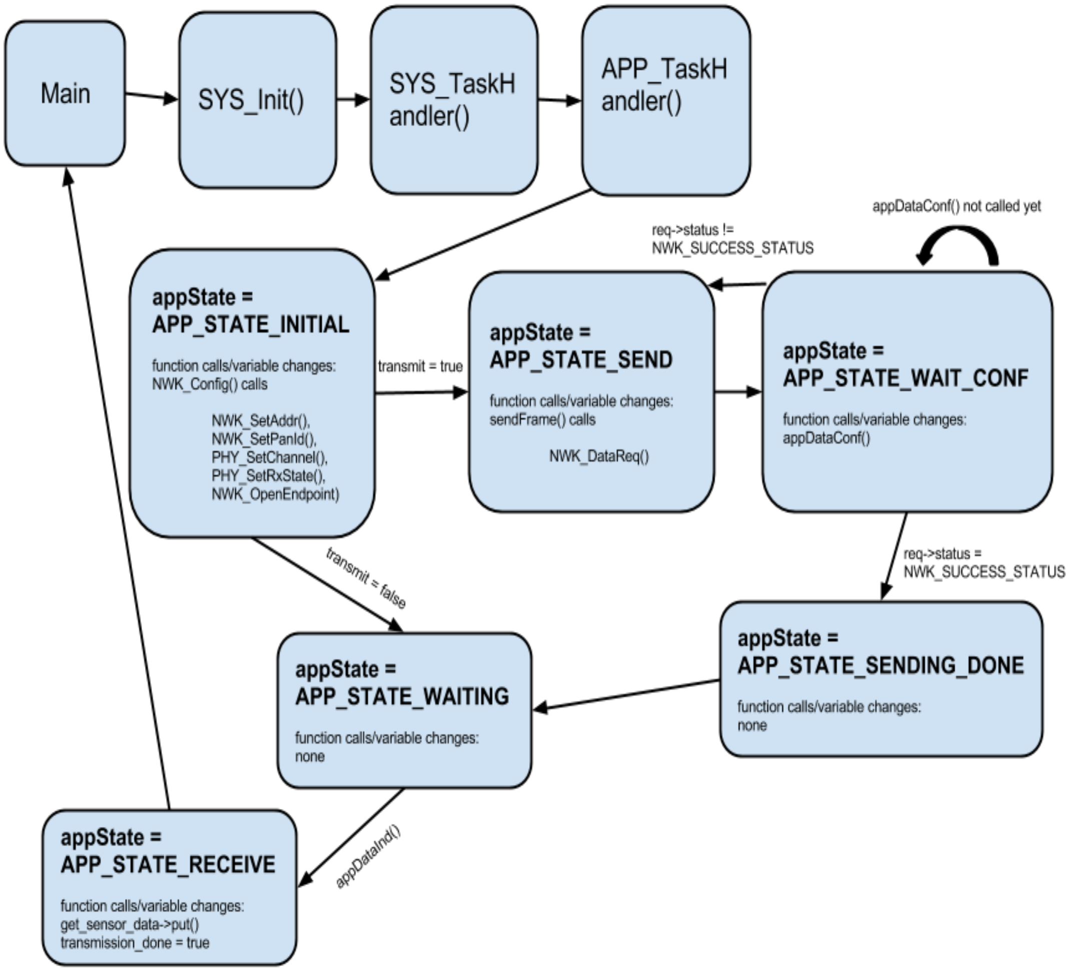

Main
SYS_Init()
SYS_TaskHandler()
APP_TaskHandler()
appState = APP_STATE_INITIAL
function calls/variable changes: NWK_Config() calls
NWK_SetAddr(), NWK_SetPanId(), PHY_SetChannel(), PHY_SetRxState(), NWK_OpenEndpoint()
transmit = true
appState = APP_STATE_SEND
function calls/variable changes: sendFrame() calls
NWK_DataReq()
req->status != NWK_SUCCESS_STATUS
appDataConf() not called yet
appState = APP_STATE_WAIT_CONF
function calls/variable changes: appDataConf()
req->status = NWK_SUCCESS_STATUS
transmit = false
appState = APP_STATE_SENDING_DONE
function calls/variable changes: none
appState = APP_STATE_WAITING
function calls/variable changes: none
appDataInd()
appState = APP_STATE_RECEIVE
function calls/variable changes: get_sensor_data->put() transmission_done = true

Figure 10: Sensor Node Wireless Communication State Diagram

Wireless communication between the sensor node to the base station is handled by the bitcloud_node task. This task uses Atmel's Lightweight Mesh software stack which is a streamline version of Atmel's BitCloud software stack [3].   This software stack works with the Atmega128RFA1 transceiver. A state diagram of the Bitcloud node behavior is depicted in **Figure 10.**

Within the main function of the bitcloud_node task contains a state machine that acts as the core mechanism for transmitting and receiving data. Before the state machine runs, SYS_Init() and SYS_TaskHandler() must run first to enable the lower level functions that control the transmission to works. After these function calls, the function APP_TaskHandler() must execute to run the state machine. The variable appState determines the current state of the base station during the state machine.

The first state, APP_STATE_INITIAL, runs all the network configurations necessary for network communication. This includes setting the network address, APP_ADDR of the node using the NWK_SetAddr() function. Another function, NWK_SetPanId() sets the network identifier, APP_PANID, that the node will communicate with. To set the communication frequency of the transmitter, the state calls the function PHY_SetChannel(). Lastly, the state creates an endpoint used for data communication transfer using the NWK_OpenEndpoint() function. Once these functions execute, the node either goes into APP_STATE_WAITING to receive a command from the base station.

In APP_STATE_WAITING, the node stays idle until it receives a command frame from the base station. Once the node receives a command frame, a callback function called appDataInd() runs. In this function, the node receives the data within the command frame and goes into APP_STATE_RECEIVE.

In APP_STATE_RECEIVE, the node checks the packet's flag. If the flag equals the typedef SENSOR_CONFIG_REQ, the state configures the node with the sensor driver specified in command_pack.data. Then, the state sets flags indicating that the node can read from the sensor. Next, it sets transmission_done to true to exit out of the state machine. If the flag equals the typedef SENSOR_DATA and the node is configured for a sensor, the node goes into APP_STATE_SEND.

In APP_STATE_SEND, the node calls the sendFrame() function for data transmission. This function sets the parameters of the nwkDataReq struct needed for data transmission to the base station. These parameters include the base station address and the command data. After the function sets these parameters, the node sends the command frame to the base station. The node then goes into

APP_STATE_WAIT_CONF to wait for a confirmation from the base station that it received the transmitted data.

In APP_STATE_WAIT_CONF, the node waits for the appDataConf() call back function to execute. This callback function receives the status parameter in nwkDataReq. If this parameter equals the typedef NWK_SUCCESS_STATUS, the node goes into APP_STATE_SENDING_DONE. Otherwise, it goes back into APP_STATE_SEND to resend the command frame.

In APP_STATE_SENDING_DONE, the node puts false into the queue and sets transmission_done to true to exit from a while loop. Next, the state machine restarts with the node entering APP_STATE_INITIAL again.

5. Poly_Sense Server

The *Poly_Sense* server is a C++ program responsible for communication between the base station and the *Poly_Sense* Monitor.

During the design process the introduction of a separate server and front-end was made. By introducing a separate server which handled communication along the serial port the project became more modular. The separation of server and frontend allows users to host the server on a separate computer in a remote location while still being able to retrieve data from the sensors over TCP connections. The separation of server and frontend also allowed the use of a language more suited for serial communication on the server (C++) and the use of a language more suited for UI on the frontend (Java).

a. TCP Communication

When the server launches it opens a port and listens for connections. Once a connection has been established on this port (presumably from the frontend) the server opens a connection on the serial port and waits for packets from the frontend. Once a packet is received from the frontend, the server writes the packet to the serial line. When a packet is received along the serial line the server performs necessary actions to the data to put it in Network Order for the Java frontend and then sends it via the port opened with the frontend**. Figure 11** below is a state diagram that illustrates this process.

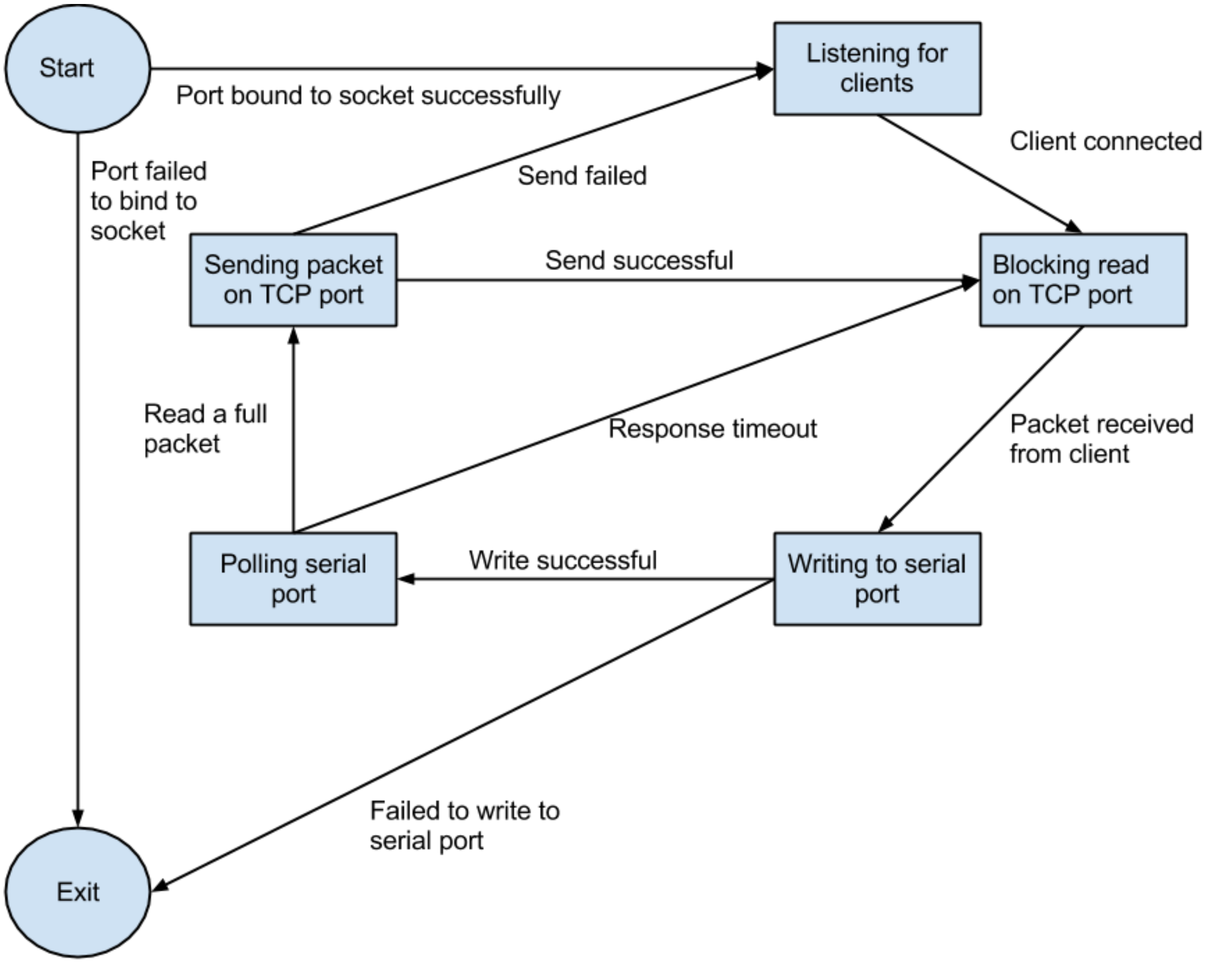

Start
Port bound to socket successfully
Listening for clients
Client connected
Send failed
Port failed to bind to socket
Sending packet on TCP port
Send successful
Blocking read on TCP port
Read a full packet
Response timeout
Packet received from client
Polling serial port
Write successful
Writing to serial port
Failed to write to serial port
Exit

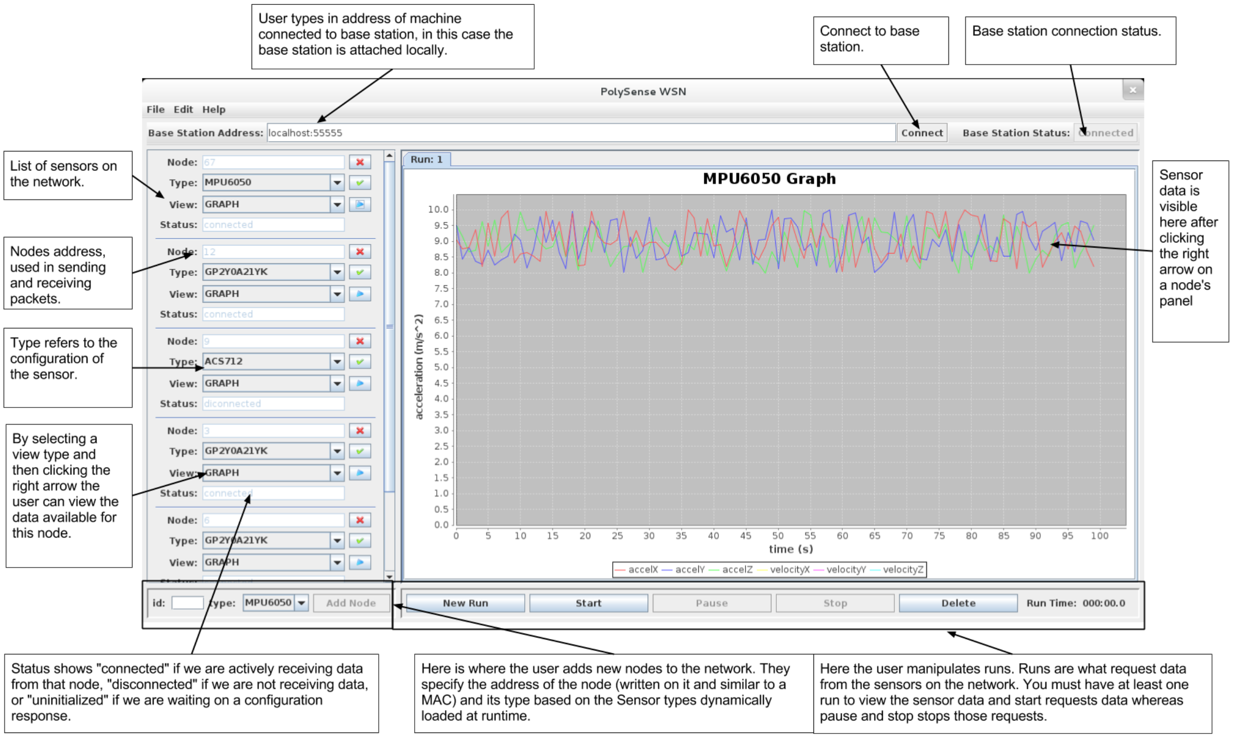
Figure 11: Poly_Sense Server State Diagram

b. Serial Communication

Reading and writing packets to the serial port was accomplished using the RS232 library. Writing packets was especially simple with this library, it allowed the server to write full byte arrays at a time. Reading was more challenging as the library provided no blocking read function. Implementing a pseudo-blocking read was accomplished by continuously calling the read function while checking how much was actually read and saving the bytes read to the packet accordingly.

6. *Poly_Sense Monitor*

The *Poly_Sense* Monitor is a standalone java application that connects to the *Poly_Sense* server, requests and sends data to the server and displays gathered sensor data. Because it is a java application, the *Poly_Sense* Monitor should be usable on any architecture that has a JVM implemented for it. We can officially support both Windows 8 and Archlinux. **Figure 12** shows a labeled screen capture of the frontend.



User types in address of machine connected to base station, in this case the base station is attached locally.

Connect to base station.

Base station connection status.

Base Station Address: localhost:55555

Base Station Status: connected

List of sensors on the network.

Nodes address, used in sending and receiving packets.

Type refers to the configuration of the sensor.

By selecting a view type and then clicking the right arrow the user can view the data available for this node.

Sensor data is visible here after clicking the right arrow on a node's panel

Node: 67
Type: MPU6050
View: GRAPH
Status: connected

Node: 12
Type: GP2Y0A21YK
View: GRAPH
Status: connected

Node: 9
Type: ACS712
View: GRAPH
Status: diconnected

Node: 3
Type: GP2Y0A21YK
View: GRAPH
Status: connected

Node: 6
Type: GP2Y0A21YK
View: GRAPH

id: type: MPU6050

Run Time: 000:00.0

Status shows "connected" if we are actively receiving data from that node, "disconnected" if we are not receiving data, or "uninitialized" if we are waiting on a configuration response.

Here is where the user adds new nodes to the network. They specify the address of the node (written on it and similar to a MAC) and its type based on the Sensor types dynamically loaded at runtime.

Here the user manipulates runs. Runs are what request data from the sensors on the network. You must have at least one run to view the sensor data and start requests data whereas pause and stop stops those requests.

Figure 12: User Interface Elements

a. Monitor Logic

When the frontend starts there is no communication between it and the server. Once the user connects to a server the frontend periodically "pings" the server. After the user adds a node and sets its type the frontend sends a configuration packet to the server, which is passed along to the specified node. The frontend then waits for a confirmation that the node was configured successfully. At this point the frontend visually updates that the node has been configured and is part of the system. If the user creates a run at this point and starts it the frontend will periodically send data requests to each node in the system and expect data packets back.

b. TCP Communication

The *Poly_Sense* Monitor interfaces the sensor network via the *Poly_Sense* Server. This communication is established using Java sockets and a TCP connection. The TCP connection provides many benefits including:

- Delivery assurance
- Flow control
- Sequencing
- Acknowledgements

One thing that TCP connections do not supply is a means of testing when the endpoint has become unreachable. In this project that would be the case of the *Poly_Sense* server becoming unavailable. To address this shortcoming of TCP the *Poly_Sense* Monitor periodically "pings" the server. If an acknowledgement is not received after some time then the connection will be terminated.

c. Sensor Modularity

Since the aim of this project was to make the system as extensible as possible, the *Poly_Sense* Monitor was designed to allow end users to create and add new sensors easily. For a end user to create a new sensor they must extend the Sensor abstract class and implement the methods outlined in appendix E.1.

Once the user creates a class extending the Sensor class the user should place this class in the wsn.sensor package. If the user is using an IDE such as Eclipse they can simply add the *Poly_Sense* Monitor executable, WSN.jar, as a library and then launch the executable. To do this without an IDE the user first needs to compile their java source while linking against the executable:

```
javac -cp WSN.jar HelloSensor.java
```

Then they must then create the directory structure representing the wsn.sensor package and move the compiled Sensor into that folder:

```
mkdir wsn && mkdir wsn/sensor/ && mv HelloSensor.class
wsn/sensor/
```

Finally the user must update the WSN.jar with the new sensor:

```
jar uf WSN.jar wsn/sensor/HelloSensor.class
```

The next time that the user executes the program, the Sensor that they created will be available.

VI. Testing

Poly_Sense Board Testing

The Poly_Sense went through several tests for verification of its operation. The first test ensured that the two boost regulator circuits operated as expected before placement of the other components on the board. This test consisted of soldering the regulator circuits to the board and soldering wires at test points TP1, TP2, TP3, and ground. A power supply set at 3V connected to the wire at TP3 and ground. To verify correct operations of the regulator circuits, a multimeter attached to TP1 and TP2. In addition, a wire connected TP3 to the EN pin of U1 (pin 3) to enable the output of the 5V regulator. The results of the test showed that the regulator circuits kept a rail voltage close to 3.3V at TP1 and 5V at TP2.

The next test verified working operations of the microcontroller. This test included soldering the rest of the components to the Poly_Sense board. Next, an in-system programmer (ISP) connected to the ISP programming port (J2). To verify that the microcontroller operated as expected, the fuses were set using the ISP. In the first iteration of the test, the ISP could not read the microcontroller's id and therefore could not set the fuses. Upon inspection, poor solder connections to the microcontroller caused this error. This made the board difficult for rework, thus the remaining two Poly_Sense boards were used for the remainder of the tests. To mitigate this issue, a reflow oven was used to solder the components to the Poly_Sense board. The results of a second iteration of the tests demonstrated functionality of the microcontrollers on the remaining boards.

Another test included the electrical characteristics of the board. This test included measuring the power consumption of the board during typical operations and the capabilities of the regulating circuit. During typical operations (wireless transmission), the board consumed 30mA of current which equated to 90mW of power consumption. The board could also operate at low input voltages (1.5V), and at higher voltages (3.3V), but the board draws more current at lower voltages. The board also loses voltage regulations at supply voltages higher 3.3V due to the limitations of the boost converter. These results show that the board shows some capabilities for operating on voltage sources that vary from nominal value of 3V, but the system should run at close to 3V to ensure proper functionality of the board.

The final test included a wireless range test of the board. The location of this test consisted of a house with other typical household wireless devices (including wireless routers, laptops, computers, etc.) which potentially added noise to the wireless signal. The system was configured in a typical data gathering configuration with a node transmitting sensor data to the base station. To test the range of the system, the sensor node was slowly moved away from the base station until the base station no longer received data from the node. The results of the test demonstrated that the board could effectively transmit data at approximately 50 feet. However, further testing could be done in a noiselessly free environment, such as an anechoic chamber, to determine the absolute maximum range of the board. **Table 1** below shows a listing of the board test results.

Table 1: Table Summary of the Poly_Sense Board Test Results

Parameters	Sym	Min	Typ	Max	Units	Conditions
Supply voltage	Vin	1.5	3	3.3*	V	No sensor connected, Note 1
3.3V Rail	3.3V	3.27	3.3	3.36	V	No sensor connected
5V Rail	5V	4.98	5V	5.02	V	No sensor connected
Supply current	Iin	10	30	50	mA	No sensor connected, Note 2
Wireless Range	-	-	-	50	ft.	Note 3

Note 1: For Vin>3.3V, the 3.3V rail will not remain in regulation

2: Min current measured at Vin =3.3V, Max current measured at Vin = 1.5V. Typ measured with Vin = 3V

3: Measured with MPU6050 sensor connected, Sparkfun dev. board as base station

***Poly_Sense* Server**

The *Poly_Sense* Server met expectations. It accurately transmitted data between the *Poly_Sense* Monitor and the wireless sensor network. It is robust and can gracefully handle dropped clients (the Frontend) as well as being disconnected from the base station.

One possible concern related to the server can be seen in the fact that it only polls for new data on the serial port every 100ms. If the latency of the network were lower this would possibly become a bottleneck but in the system's current state this is of small concern.

Another small problem with the Server is that if it starts with an incorrectly specified com port then there is no method of changing that com port besides restarting the program. One possible solution to this problem would be to allow the user to send a packet from the *Poly_Sense* Monitor to the server which specifies the com port for the server to connect to. Because of the lack of time, this feature was not implemented.

***Poly_Sense* Monitor**

The *Poly_Sense* Monitor was able to accurately display sensor data, dynamically change sensor configuration, provide a means for persisting data while remaining mostly stable, however, there are some places that where it falls short.

Once the wireless sensor network becomes unreachable the *Poly_Sense* Monitor has no way of recovering. This is different from the state where the user removes all nodes from the frontend however is still problematic since in some cases the nodes may temporarily go out of range. This issue could most likely be resolved relatively easily but was pushed back because of more pressing issues.

VII. Conclusion

The Poly_Sense system met the proposed project requirements. The end product adequately demonstrated its ability to wirelessly acquire data from different types of sensors at significant ranges and display relayed data in the graphical user interface. This system also met its power consumption specification and came at a low manufacturing cost relative to similar products on the market. The front end provides an easy to use API which enables the user to integrate new sensors.

The Poly_Sense system has areas where further development could bolster its functionality and ease of use. Communication failure with the onboard analog to digital converter has left the current board design in a state which does not allow it to accommodate the use of analog sensors. A simple solution for this issue is the integration of an external analog to digital converter; however another revision of the Poly_Sense board would allow for an opportunity to replace this part with a fully tested device. If a board revision were made, a usb to uart converter chip would be added to eliminate the need for an external device for base station-to-host computer serial communication.

Another issue with the current system is that the time for a response to a request was larger than expected. The time for a packet to propagate to the requested node and return was sometimes larger than 1 second. Furthermore, because the data requests happen sequentially, this delay increases as the number of nodes in the system does. There are a few ways to address this problem but the most viable solution would be to add separate transmission and receiving radios so that the data transmission would not have to be sequential.

With regards to the node software, an overall system optimization could be made by configuring the node to transparently configure and acquire data from any sensor connected on the I^{2}C bus solely based upon data packets sent from the base station. This would eliminate the need to ever reprogram a node, and would allow developers to implement device drivers through applications built on the host computer as opposed to having the sensor drivers loaded on each node as they currently are.

Bibliography

[1] L. Gürgen et al. “Data Management Solutions in Networked Sensing Systems” in *Wireless Sensor Network Technologies for the Information Explosion Era*, vol 278. Berlin/Heidelberg, Germany: Springer, 2010 pp 111-137

[2] J. Daniel J et al. “Industrial Grade Wireless Base Station for Wireless Sensor Networks,” in *Electronics Computer Technology (ICECT), 2011 3rd International Conference*, 2011, pp. 245 – 249

[3] Atmel, *Lightweight Mesh Software Stack* [Online]. Available: http://www.atmel.com/tools/LIGHTWEIGHT_MESH.aspx

[4] P. Kinney “ZigBee Technology: Wireless Control that Simply Works”
[Online]
http://hometoys.com/emagazine.php?url=/htinews/oct03/articles/kinney/zigbee.htm

[5] E. Gaura et al. *Wireless Sensor Networks: Deployments and Design Frameworks*, New York: Springer, 2010

[6] MEMSIC Inc., *Wireless Sensor Networks* [Online]. Available: http://www.memsic.com/technology/wireless-sensor-networks.html

[7] Wikipedia, *List of wireless sensor nodes* [Online]. Available: http://en.wikipedia.org/wiki/List_of_wireless_sensor_nodes

[8] Atmel, *ATmega128RFA1 Datasheet* [Online]. Available: http://www.atmel.com/Images/doc8266.pdf

APPENDIX A: Senior Project Analysis

Summary of Functional Requirements

Poly_Sense provides a platform for developers to use for a wide range of Wireless Sensor Network (WSN) applications. This modular system supports different sensing applications by allowing the developer to easily change between supported sensors through the graphical user interface (GUI). The platform also allows developers to integrate new sensors by writing device drivers which follow the platform's guidelines and utilize the application programming interface (API). This low-power and cost-effective wireless solution not only provides a basic platform for entry-level developers, but also accommodates larger-scale applications.

Primary Constraints

During development, several design challenges were faced and overcome. Establishing serial communication between the base station board and host computer in an efficient manner proved to be challenging. Ideally the host computer would utilize function to read from the com port which could block until a specified data buffer was filled; however a library containing such a function was not found, and our own development efforts were not able to achieve this goal. Ultimately, a polling approach was taken which was effective, but inefficient.

Another challenge was the assembly of the base station and node printed circuit boards. Our team had limited prior experience with surface mount soldering, and actual assembly was difficult. One board was rendered unusable due to difficulties in assembly, and application specific embedded development was delayed due to the setbacks caused during assembly.

One of the biggest difficulties was working with the Atmel lightweight mesh. The pre-built software stack was dense and uncommented, making fundamental application development incredibly difficult. This was by far the largest setback faced in the project because it was difficult to determine whether the initial connectivity issues were due to hardware issues with the custom printed circuit boards, or with the driving software. Even after basic read and write operations were possible using the lightweight mesh, it still had to be integrated with all of the existing C++ code used to communicate with host computer and sensors which was running in FreeRTOS. This required us to strip away any unused functionality, and wrap the state machines used in the lightweight mesh into a FreeRTOS C++ class. This integration process took much longer than expected, resulting in a shift in emphasis from robust and efficient software design to functional design.

Economic

Project Costs

Below, table 1 shows the original estimated project cost.

Table 1: Estimated Project Cost

Item	Quantity	Cost Per	Total
MRF24J40MA	5	$10.87	$54.35
Base Station	1	$150.00	$150.00
Batteries	4	$1.00	$4.00
Microcontroller	4	$3.00	$12.00
Sensors	4	$3	$12.00
Estimated Cost			**$232.5**

Below, table 2 shows the final project cost.

Table 2: Final Project Cost

Item	Quantity	Cost Per	Total
Sensor Node	3	$50.53	151.59
Base station	1	$54.95	$59.54
MPU6050 (Sensor)	1	$39.95	$39.95
BMP085 (Sensor)	1	$19.95	$19.95
In-System Programmer	1	$14.95	$14.95
ACS712	1	$14.95	$14.95
USB-to-Serial	1	$14.95	$14.95
Poster board	1	$63.86	$63.86
Final Cost			**$379.74**

Bill of Materials

Ref Designator	Part Name	Type	Qty	Unit Cost	Sub-Total	Manufacturer
U4	IC ADC 12BIT I2C ALERT 8-MSOP	ADC	1	$3.16	$3.16	TI
None	ANTENNA 2.4GHZ SMA MALE BLACK 3"	Antenna	1	$3.78	$3.78	Pulse Electronics
B1	FILTER BALUN ATMEL 2.45GHZ	Balun	1	$1.65	$1.65	Johanson
C5, C6	CER 22PF 50V 5% NP0 0603	Capacitor	2	$0.11	$0.22	AVX
C3, C4	CER 1UF 16V 10% X5R 0603	Capacitor	2	$0.12	$0.24	AVX
C9, C10	CER 12PF 50V 5% NP0 0603	Capacitor	2	$0.10	$0.20	AVX
C15	CER 0.47PF 50V 5% NP0 0603	Capacitor	1	$0.63	$0.63	Samsung
C7, C8	CER 18PF 50V 5% NP0 0603	Capacitor	2	$0.10	$0.20	Kemet
C1, C2	CAP CER 0.1UF 25V 10% X7R 0603	Capacitor	2	$0.10	$0.20	TDK
C11, C12, C13, C14	CAP CER 10UF 10V 20% X5R 1206	Capacitor	4	$0.27	$1.08	TDK
J1	FEMALE SMA	Connector	1	$3.31	$3.31	Linx
Y1	CRYSTAL 16.000MHZ 18PF SMD	Crystal	1	$0.57	$0.57	TXC
Y2	CRYSTAL 32.768KHZ 12.5PF SMD	Crystal	1	$2.36	$2.36	Abracon
J4	CONN PWR JACK DC 2.1X5.5 8A T/H	DC Jack	1	$1.37	$1.37	Kobiconn
N/A	2 AA Batteries	Battery	1	$2.50	$2.50	
	2 AA Battery Pack		1	$2.49	$2.49	
L1, L2	4.7µH	Inductor	2	$2.02	$4.04	Wurth Electronics Inc
D1	LED 630NM RED WTR CLR 0603 SMD	LED	1	$0.55	$0.55	Rohm
D2	LED SMARTLED GREEN 570NM 0603	LED	1	$0.22	$0.22	OSRAM
U3	IC 4BIT NON-INV TRANSLTR 14TSSOP	Level Shifter	1	$1.87	$1.87	TI
mcu1	IC AVR MCU 2.4GHZ XCEIVER 64QFN	Microcontroller		$9.46	$0.00	ATMEL
S1	Mini Push Button Switch	Push Button	1	$0.35	$0.35	
U1, U2	MCP1640B	Regulator	2	$0.44	$0.88	Microchip Technology
R1	10K OHM 1/8W 1% 0805 SMD	Resistor	1	$0.10	$0.10	Yageo
R3	976K OHM 1/8W 1% 0805 SMD	Resistor	1	$0.10	$0.10	Yageo
R2, R4	309K OHM 1/8W 1% 0805 SMD	Resistor	2	$0.10	$0.20	Yageo
R5, R6, R10, R11	4.7K OHM 1/8W 1% 0805 SMD	Resistor	4	$0.10	$0.40	Panasonic
R7	536K OHM 1/8W 1% 0805 SMD	Resistor	1	$0.10	$0.10	Yageo
R8	750 OHM 1/8W 1% 0805 SMD	Resistor	1	$0.10	$0.10	Panasonic
R9	820 OHM 1/8W 1% 0805 SMD	Resistor	1	$0.10	$0.10	Yageo
S2	SWITCH SLIDE SPDT 0.3A 30V	Switch	1	$0.45	$0.45	C&K
U5	IC VIDEO SWITCH QUAD SPDT 16QSOP	Switch	1	$0.54	$0.54	TI
	Misc Parts				$3.00	
N/A	Board Fabrication	N/A	1	$13.57	$13.57	OshPark
				Total	**$50.53**	

Additional Costs

In addition to the costs stated above, the project also required a laptop computer (estimated at $300) to run the host computer software. However, any modern computer with an available USB could run the host computer software.

Development Time

Figure 1,below, shows the estimated development time of this project.

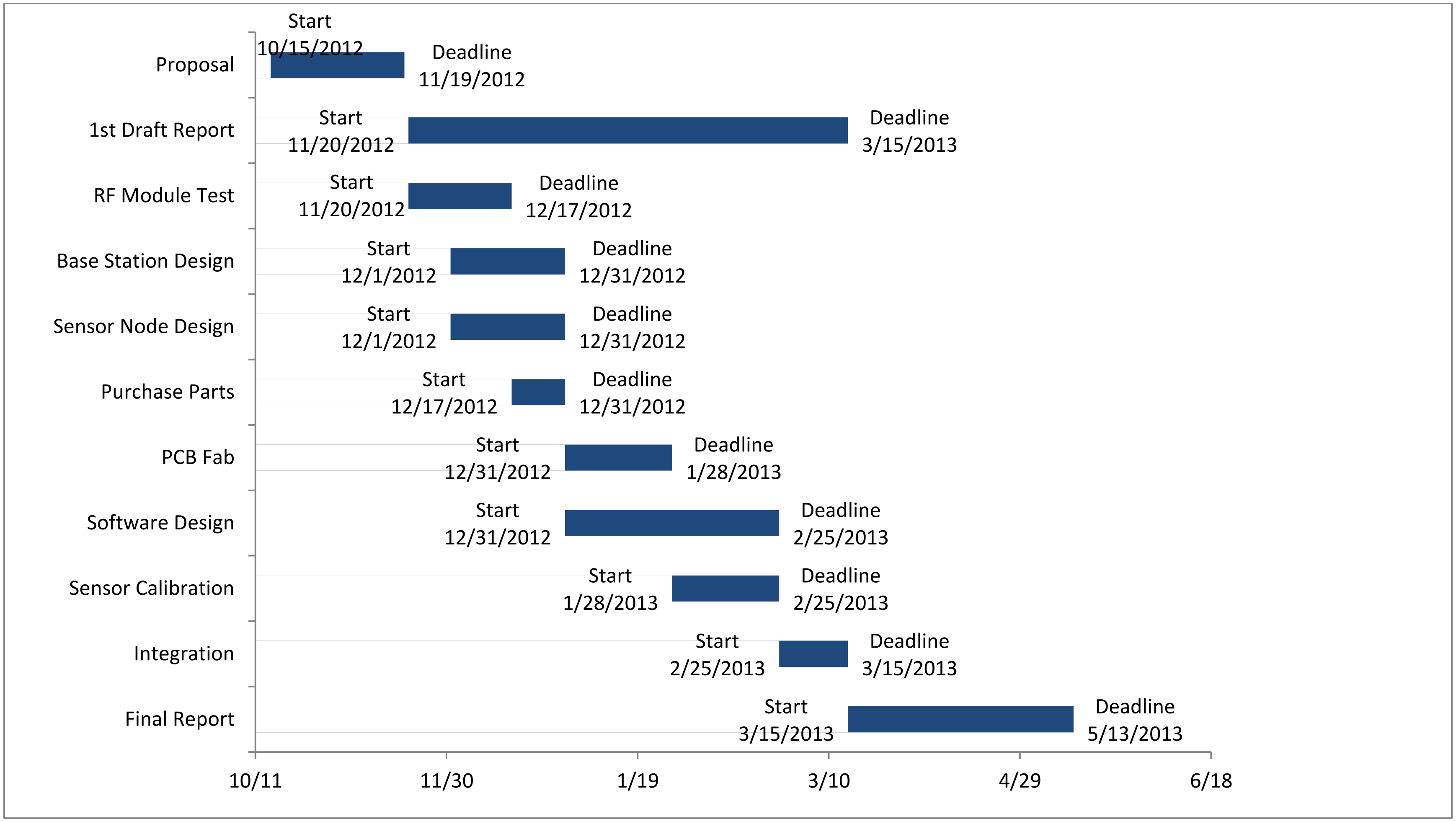


Figure 1: Estimated Project Timeline

Figure 2, below, shows the actual development time of this project.

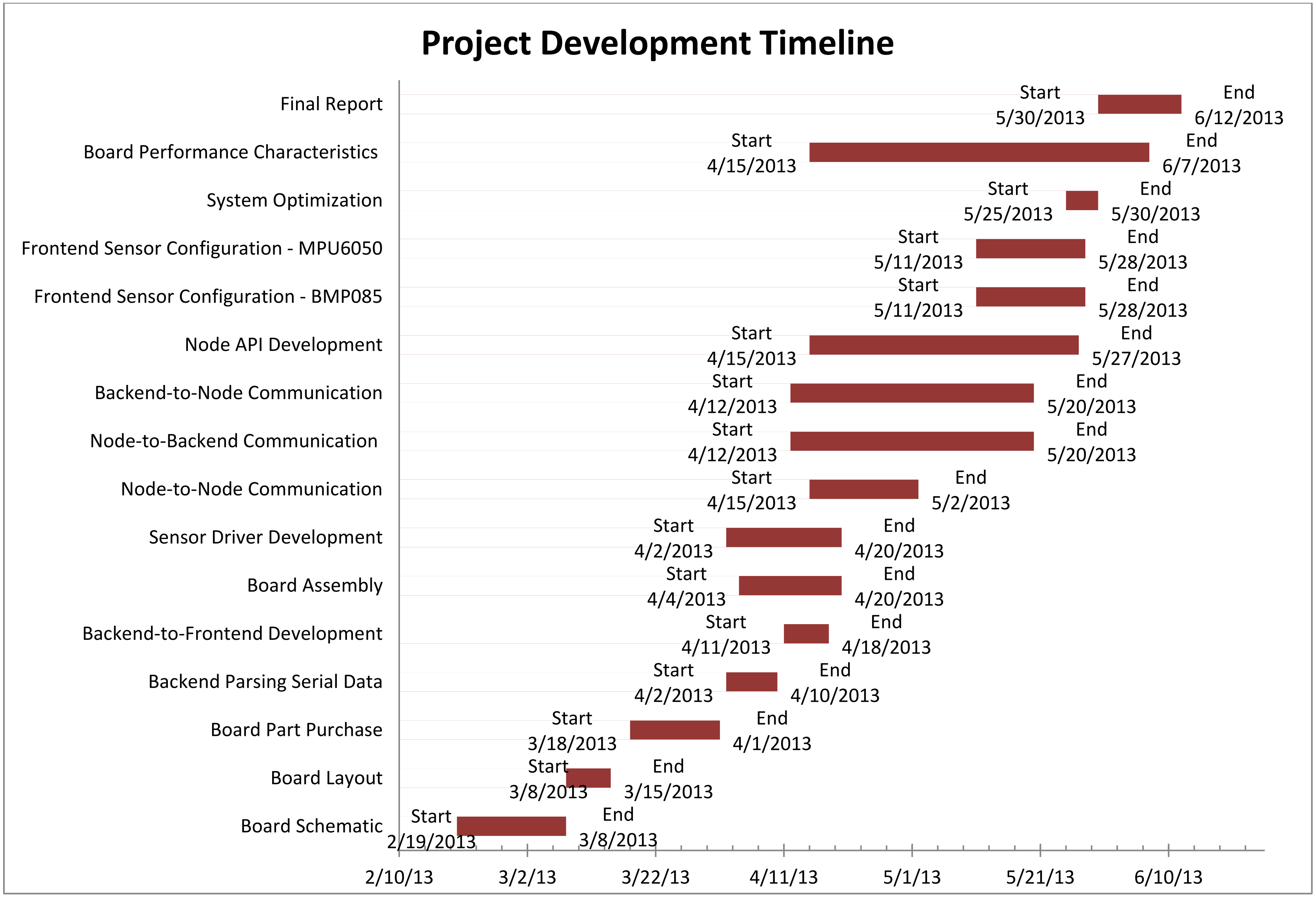


Figure 2: Actual Project Timeline

If manufactured on a commercial basis

If this system were commercialized, each product would be made to order as to reduce initial production costs, with an estimated annual sale of 10,000 units. The current production cost is $50.53 per unit, however prior to commercial production another hardware revision would be necessary to include additional hardware features and reduce overall size; this redesign has as estimated production cost of $55 per unit. The devices would be sold at $75, yielding an estimated $20 profit per unit, and an annual profit of $200,000. The only cost deferred to users would be batteries for the device and the external sensor which they wish to interface.

Environmental

In its current implementation, the *Poly_Sense* system has completely RoHS compliant components besides the solder used. If this project were commercialized, the manufacturer could easily, at a slightly increased price, use a non-lead based solder. The sensor nodes currently use 2 disposable AA batteries for power. If the system was left running for long amounts of time or a user had a large amount of nodes then the consumption of disposable batteries may become a concern. To remedy this, the manufacturer could release a rechargeable version of the *Poly_Sense* system; this would also increase manufacturing costs while possibly lowering the maximum run time of the system.

Because the *Poly_Sense* system can be deployed to remote locations, there is a possibility that the system's presence could disrupt natural environments. Since one functional requirement of the project was to minimize power consumption, it is unlikely that electrical shock will interfere with the Node's environment. A larger concern may be the fact that the components are not biodegradable. If an end user were to place a node in a remote location and be unable to locate it later the node could stay in the environment for a long time.

Manufacturability

Since the Poly_Sense board features surface mount devices, the production cost for assembling the board could go up if the processed was fully automated. This involves purchasing an expensive machine that will pick and place the components and another machine to apply solder paste to the surface mount pads. In addition, a reflow oven would also be required to solder the components to the board. These costs would only be justified should the market require a high volume production of the Poly_Sense board. For low volume, the board can be soldered by applying solder paste by hand and using a reflow oven.

Sustainability

Since the Poly_Sense draws 30mA of average current, the device can run for approximately 100 hours on two AA batteries rated at 1500 mAh. This would require the user to change the batteries every four days if the board runs continuously. For larger scale applications, this would result in significant maintenance cost to replace batteries for each node.

In addition to maintenance cost, this also impacts the sustainability of the device because of the consumption of resources to manufacture the batteries. In a year, a board can use approximately 180 batteries. This number scales linearly for each node added to the system.

Overall, although the board consumes relatively low power, the power consumption of the device must be improved to further support a sustainable design with minimal use of batteries. One alternative to the design would incorporate the use of rechargeable batteries as an alternative to standard batteries. However, this does not improve on the overall performance of the board since the batteries would still need to be replaced every four days. A better solution to this challenge would call for a redesign of the power supply unit with more efficient and less power intensive regulators. However, this would require the designer to have more experience and background with power electronics.

Ethical

The *Poly_Sense* system is a cheaper alternative to other wireless sensor network system. This would empower sensor developers who would otherwise be unable to purchase a wireless sensor network system. The IEEE Code of Ethics asks its members to “assist colleagues and coworkers in their professional development”. Our project would assist sensor driver developers in testing their software, in other words; their professional development. In this sense, the *Poly_Sense* system is ethical.

As with most Engineering projects, a primary concern of the *Poly_Sense* system was the negative effects that manufacturing may have on the environment. As mentioned in the Environmental section, there may be some ethically detrimental aspects of this project in regards to its effect on the environment. The IEEE code of ethics asks its members to “disclose promptly factors that might endanger the public or the environment.” By releasing these concerns to the public, this project has maintained ethical professionalism.

Health and Safety

One of the functional requirements of this project was to minimize power usage. Because of this, the *Poly_Sense* system could only administer a small electrical shock that would cause no serious harm to a person.

The Node and Base Station boards allow for a user to incorrectly connect a sensor. If this were to occur, it is possible that the sensor could become damaged.

Social and Political

As with any means of obtaining remote data, application development using the *Poly_Sense* system should always exemplify discretion. Depending on the type of sensor connected to the system and the scope of the embedded application, public privacy issues could arise.

Development

Because this project was taken on by an interdisciplinary team of Electrical Engineers and a Computer Scientist, members gained insight into fields other than their own, and delved deeper into specific areas of study to address application-specific issues while developing a product.

The Computer Scientist had an opportunity to apply inter-process communication practices to remote communication between a Java Virtual Machine (JVM) application and a binary executable as well as between that binary executable and a microcontroller across a serial port. The Computer Scientist also learned how to use java reflections to dynamically instantiate objects with classes that are unknown at compile-time. The Computer Scientist was also required to learn new technologies for advanced UI elements such as MiGLayout, a layout manager similar to HTML.

The Electrical Engineers gained more experience in object-oriented software design in embedded applications through the development of sensor drivers, and FreeRTOS tasks which also required the application of many advanced software design consideration such as dynamic memory allocation and task synchronization with semaphores. To address the issues of wireless and wired serial communication the team had to learn the fundamental of 802.15.4 protocol, and UART protocol on multiple platforms. The Electrical Engineers also gained experience in PCB layout design and surface mount soldering.

APPENDIX B: Parts list and cost

See bill of materials on page 37.

APPENDIX C: Project code

Poly_Sense Monitor
Source Code

```java
package wsn.sensor.types;

/**
 * A simple wrapper class that creates and object from a float which represents
 * a acceleration value.
 *
 * @author jshirley
 *
 */
public class Acceleration {
        float acceleration;

        public float getAcceleration() {
                return acceleration;
        }

        public void setAcceleration(float acceleration) {
                this.acceleration = acceleration;
        }

        public Acceleration(float acc) {
                this.acceleration = acc;
        }
}
```

```java
package wsn.panels;

import java.awt.GridBagConstraints;
import java.awt.GridBagLayout;
import java.util.Observable;
import java.util.Observer;

import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JSeparator;
import javax.swing.ScrollPaneConstants;
import javax.swing.SwingConstants;

import wsn.Globals;
import wsn.sensor.Sensor;
import wsn.util.Tuple;

/**
 * This Class is a container for all of the PrefixActiveNodePanels and
 * SensorSideViews (if implemented for the Sensor).
 *
 * @author jshirley
 *
 */
public class ActiveNodesPanel extends JScrollPane implements Observer {
        /**
         * The Panel that contains all of the NodePrefixPanels and each active
         * Sensor/Node pairing's SideViewPanel (if implemented).
         */
        private JPanel panel;

        /**
         * Creating a new ActiveNodesPanel lays out its components and starts
         * watching Globals for Events.
         */
        public ActiveNodesPanel() {
                setHorizontalScrollBarPolicy(ScrollPaneConstants.HORIZONTAL_SCROLLBAR_NEVER);
                panel = new JPanel();
                panel.setLayout(new GridBagLayout());
                setViewportView(panel);
                setVerticalScrollBarPolicy(ScrollPaneConstants.VERTICAL_SCROLLBAR_ALWAYS);

                Globals.getInstance().addObserver(this);
        }

        /**
         * updatePanel redraws the side panel and should be called when nodes are
         * either added or removed.
         */
        public void updatePanel() {
                panel.removeAll();

                GridBagConstraints gbc = new GridBagConstraints();
                gbc.gridx = 0; // single col
                gbc.fill = GridBagConstraints.HORIZONTAL;

                int indx = 0;
                for (Tuple<Short, Sensor> t : Globals.getInstance().getNodes()) {
                        // add seperators if more than one node
                        if (indx > 0) {
                                gbc.gridy = indx++;
                                panel.add(new JSeparator(SwingConstants.HORIZONTAL), gbc);
                        }

                        gbc.gridy = indx++;
                        panel.add(new PrefixActiveNodePanel(t.y, t.x), gbc);

                        if (t.y.getSidePanel() != null) {
                                gbc.gridy = indx++;
                                panel.add(t.y.getSidePanel(), gbc);
```

```
			}

		}
		revalidate();
		repaint();
	}

	/*
	 * update is called when Globals fires an Event and updates this Panel
	 * accordingly.
	 *
	 * @see java.util.Observer#update(java.util.Observable, java.lang.Object)
	 */
	@Override
	public void update(Observable o, Object arg) {
		if (o instanceof Globals) {
			if (arg == Globals.Events.NODE_ADDED) {
				updatePanel();
			} else if (arg == Globals.Events.NODE_REMOVED) {
				updatePanel();
			} else if (arg == Globals.Events.RUN_ADDED) {
				updatePanel();
			} else if (arg == Globals.Events.RUN_REMOVED) {
				updatePanel();
			}
		}

	}

}
```

```java
package wsn.sensor.view;

import java.util.Observable;

import javax.swing.JPanel;

import org.jfree.chart.ChartFactory;
import org.jfree.chart.ChartPanel;
import org.jfree.chart.JFreeChart;
import org.jfree.chart.plot.PlotOrientation;
import org.jfree.data.xy.XYSeries;
import org.jfree.data.xy.XYSeriesCollection;

import wsn.Globals;
import wsn.sensor.BMP085;
import wsn.sensor.Sensor;
import wsn.sensor.types.Pressure;
import wsn.sensor.types.Temperature;
import wsn.sensor.types.Time;
import wsn.util.Triple;


/**
 * @see SensorView
 *
 * @author jshirley
 *
 */
public class BMP085GraphView implements SensorView {

        /**
         * The panel that represents this BMP085GraphView.
         */
        private JPanel panel = new JPanel();
        /**
         * The temperature chart.
         */
        private JFreeChart tempChart;
        /**
         * The pressure chart.
         */
        private JFreeChart pressureChart;

        /**
         * Graph label.
         */
        private static String pressureXMeasurement = "time (ms)";
        /**
         * Graph label.
         */
        private static String pressureYMeasurement = "pressure (pA)";
        /**
         * Graph label.
         */
        private static String tempXMeasurement = "time (ms)";
        /**
         * Graph label.
         */
        private static String tempYMeasurement = "temperature (c)";

        /**
         * Graph label.
         */
        public static String sensorName = BMP085.sensorName;

        /**
         * Collection wrapper of the one series that is present on the pressure
         * graph.
         */
        private XYSeriesCollection pressureDataCollection;
        /**
         * Series of X and Y pairs that represent the pressure and time pairs.
         */
        private XYSeries pressureData;
        /**
         * Collection wrapper of the one series that is present on the temperature
         * graph.
         */
        private XYSeriesCollection tempDataCollection;
        /**
         * Series of X and Y pairs that represent the temperature and time pairs.
         */
        private XYSeries tempData;

        /**
         * Represents the maximum number of time measurements allowed on the graph
         * at a given time. If this is exceeded the oldest elements are removed.
         */
        private static final int xRange = 100;

        /**
         * The Sensor that this BM085GraphView uses as a model.
         */
        private BMP085 host;

        /**
         * Creating a new BMP085GraphView lays out and initializes its components.
         * It also sets the Sensor that it is modeling to the Sensor parameter s.
         *
         * @param s
         *            The Sensor this SensorView uses as a model.
         */
        public BMP085GraphView(Sensor s) {
                host = (BMP085) s;
                s.addObserver(this);

                pressureDataCollection = new XYSeriesCollection();
                pressureData = new XYSeries("pressure"); // give better keys
                pressureData.setMaximumItemCount(xRange);

                tempDataCollection = new XYSeriesCollection();
                tempData = new XYSeries("pressure"); // give better keys
                tempData.setMaximumItemCount(xRange);

                pressureDataCollection.addSeries(pressureData);
                tempDataCollection.addSeries(tempData);

                pressureChart = ChartFactory.createXYLineChart(sensorName
                                + " Graph : Pressure", // chart title
                                pressureXMeasurement, // x axis label
                                pressureYMeasurement, // y axis label
                                pressureDataCollection, // data
                                PlotOrientation.VERTICAL, true, // include legend
                                true, // tooltips
                                false // urls
                                );

                tempChart = ChartFactory.createXYLineChart(sensorName
                                + " Graph : Temperature", // chart title
                                tempXMeasurement, // x axis label
                                tempYMeasurement, // y axis label
                                tempDataCollection, // data
                                PlotOrientation.VERTICAL, true, // include legend
                                true, // tooltips
                                false // urls
                                );

                if (host.getRequestType() == BMP085.DataReqType.PRESSURE) {
                        panel = new ChartPanel(pressureChart);
                } else {
                        panel = new ChartPanel(tempChart);
                }
        }

        /*
         * (non-Javadoc)
         *
         * @see wsn.sensor.view.SensorView#getPanel()
         */
        @Override
        public JPanel getPanel() {
                return panel;
        }

        /*
         * (non-Javadoc)
         *
         * @see wsn.sensor.view.SensorView#setSensor(wsn.sensor.Sensor)
         */
        @Override
        public void setSensor(Sensor s) {
                host = (BMP085) s;

        }

        /*
         * (non-Javadoc)
         *
         * @see wsn.sensor.view.SensorView#getSensor()
         */
        @Override
        public Sensor getSensor() {
                return host;
        }

        /*
         * (non-Javadoc)
         *
         * @see wsn.sensor.view.SensorView#addData(java.lang.Object)
         */
        @Override
        public void addData(Object o) {
                if (o instanceof Triple) {
                        Time t = (Time) ((Triple) o).x;
                        pressureData.add(t.getTime() / 10,
                                        ((Pressure) ((Triple) o).y).getPressure());
                        tempData.add(t.getTime() / 10,
                                        ((Temperature) ((Triple) o).z).getTemperature());
                }

        }

        /*
         * (non-Javadoc)
         *
         * @see java.util.Observer#update(java.util.Observable, java.lang.Object)
         */
        @Override
        public void update(Observable o, Object arg) {
                if (host.getRequestType() == BMP085.DataReqType.PRESSURE) {
                        panel = new ChartPanel(pressureChart);
                } else {
                        panel = new ChartPanel(tempChart);
                }
                System.err.println("chaning view for MPU6050 graph to \n"
                                + host.getRequestType());
                Globals.getInstance().fireEvent(Globals.Events.ACTIVE_SENSOR_CHANGED);

        }
}
```

```java
package wsn.sensor;

import java.nio.ByteBuffer;
import java.util.ArrayList;

import javax.swing.JPanel;

import wsn.Globals;
import wsn.network.Network;
import wsn.network.NetworkListener;
import wsn.sensor.sideview.BMP085SideView;
import wsn.sensor.sideview.SensorSideView;
import wsn.sensor.types.Pressure;
import wsn.sensor.types.Temperature;
import wsn.sensor.types.Time;
import wsn.sensor.view.BMP085GraphView;
import wsn.sensor.view.BMP085TableView;
import wsn.sensor.view.SensorView;
import wsn.util.Triple;

import com.google.gson.JsonArray;
import com.google.gson.JsonElement;
import com.google.gson.JsonObject;
import com.google.gson.JsonParser;

/**
 * @see Sensor
 *
 * @author jshirley
 *
 */
public class BMP085 extends Sensor {
        /**
         * This Sensor's name.
         */
        public static String sensorName = "BMP085";
        /**
         * The current view that this Sensor will display if it is the active
         * sensor.
         */
        private viewType activeView = null;

        /**
         * Represents the list of views that this sensor supports.
         */
        private ArrayList<viewType> sensorViewTypes = new ArrayList<viewType>();

        /**
         * Panel for graph view.
         */
        private BMP085GraphView graphPanel;
        /**
         * Panel for table view.
         */
        private BMP085TableView tablePanel;
        /**
         * Panel for specific view.
         */
        private SensorView specificPanel;

        /**
         * All of the data that has been collected by this Sensor.
         */
        private ArrayList<Triple<Time, Pressure, Temperature>> data;

        /**
         * Panel for the sideView.
         */
        private SensorSideView sideView;

        /**
         * The current data request type that this sensor is using, can be either
         * pressure or temperature. Both values are retrieved from the node but only
         * one can be viewed.
         */
        private DataReqType reqType = DataReqType.PRESSURE;

        /**
         * @author jshirley
         *
         */
        public enum DataReqType {
                TEMPERATURE, PRESSURE
        };

        /**
         * The number of times this sensor has not recieved data when it expected
         * data. If > 5 then the Sensor state can be described as "Disconnected".
         */
        private int numFailedExpects;

        /**
         * Creating a new BMP085 initializes its supported types and its backing
         * data field as well as preparing its supported viewTypes.
         */
        public BMP085() {
                sensorViewTypes.add(viewType.GRAPH);
                sensorViewTypes.add(viewType.TABLE);
                sensorViewTypes.add(viewType.SPECIFIC);

                data = new ArrayList<Triple<Time, Pressure, Temperature>>();

                graphPanel = new BMP085GraphView(this);
                tablePanel = new BMP085TableView(this);
                specificPanel = null;
                sideView = new BMP085SideView(this);

                numFailedExpects = 0;
        }

        /*
         * (non-Javadoc)
         *
         * @see wsn.sensor.Sensor#getSensorName()
         */
        @Override
        public String getSensorName() {
                return sensorName;
        }

        /*
         * (non-Javadoc)
         *
         * @see wsn.sensor.Sensor#getViewTypes()
         */
        @Override
        public ArrayList<viewType> getViewTypes() {
                return sensorViewTypes;
        }

        /*
         * (non-Javadoc)
         *
         * @see wsn.sensor.Sensor#setActiveView(wsn.sensor.Sensor.viewType)
         */
        @Override
        public void setActiveView(viewType v) {
                activeView = v;
        }

```

```java
	/*
	 * (non-Javadoc)
	 * 
	 * @see wsn.sensor.Sensor#getActiveView()
	 */
	@Override
	public JPanel getActiveView() {
		if (activeView == viewType.GRAPH) {
			return graphPanel.getPanel();
		} else if (activeView == viewType.TABLE) {
			return tablePanel.getPanel();
		} else if (activeView == viewType.SPECIFIC) {

		}
		return null;
	}

	/*
	 * (non-Javadoc)
	 * 
	 * @see wsn.sensor.Sensor#getSidePanel()
	 */
	@Override
	public JPanel getSidePanel() {
		return sideView.getPanel();
	}

	/*
	 * (non-Javadoc)
	 * 
	 * @see wsn.sensor.Sensor#getStatusString()
	 */
	@Override
	public String getStatusString() {
		if (numFailedExpects < 5) {
			return Sensor.sensorStates.CONNECTED.toString();
		}
		return Sensor.sensorStates.DISCONNECTED.toString();
	}

	/*
	 * (non-Javadoc)
	 * 
	 * @see wsn.sensor.Sensor#requestData(wsn.network.NetworkListener, short)
	 */
	@Override
	public void requestData(NetworkListener n, short addr) {
		Globals.getInstance().getNetworkLock();
		n.sendPacket(Network.flags.SENSOR_DATA.getValue(), addr, "b");
		if (n.expectPacket(Network.flags.SENSOR_DATA.getValue())) {
			if (NetworkListener.getInstance().getPacketNodeId() == addr) {
				addData(NetworkListener.getInstance().getPacketData());
				numFailedExpects = 0;
			} else {
				System.err.println("unexpected SensorId: "
						+ NetworkListener.getInstance().getPacketNodeId()
						+ " expected: " + addr);
			}
		} else {
			numFailedExpects++;
		}
		Globals.getInstance().releaseNetworkLock();
	}

	/*
	 * (non-Javadoc)
	 * 
	 * @see wsn.sensor.Sensor#getBackingData()
	 */
	@Override
```

```java
        public String getBackingData() {
                JsonArray jAllData = new JsonArray();
                JsonObject jData;

                for (Triple<Time, Pressure, Temperature> triple : data) {
                        jData = new JsonObject();
                        if (triple.x != null) {
                                jData.addProperty("time", triple.x.getTime());
                        } else {
                                jData.addProperty("time", "");
                        }
                        if (triple.y != null) {
                                jData.addProperty("p", triple.y.getPressure());
                        } else {
                                jData.addProperty("p", "");
                        }
                        if (triple.z != null) {
                                jData.addProperty("t", triple.z.getTemperature());
                        } else {
                                jData.addProperty("t", "");
                        }
                        jAllData.add(jData);

                }
                return jAllData.toString();
        }

        /*
         * (non-Javadoc)
         *
         * @see wsn.sensor.Sensor#setData(java.lang.String)
         */
        @Override
        public void setData(String data) {
                this.data.clear();

                JsonElement p = new JsonParser().parse(data);
                JsonArray jArray = p.getAsJsonArray();
                for (JsonElement jEle : jArray) {
                        JsonObject jObj = jEle.getAsJsonObject();
                        long time = jObj.get("time").getAsLong();
                        long pressure = jObj.get("p").getAsLong();
                        long temperature = jObj.get("t").getAsLong();

                        this.data.add(new Triple<Time, Pressure, Temperature>(
                                        new Time(time), new Pressure(pressure), new Temperature(
                                                        temperature)));
                }
        }

        /*
         * Expects [i,data,data,data,data,i,data,data,data,data]
         *
         * @see wsn.sensor.Sensor#addData(byte[])
         */
        @Override
        public void addData(byte[] bytes) {
                System.err.println("Packet recieved in BMP085\n");
                for (int i = 0; i < NetworkListener.DATA_SIZE; i++) {
                        System.err.println(bytes[i] + " ");
                }
                long time = Globals.getInstance().getActiveRun().getTime();

                int vals[] = new int[2];

                int position, numVals;
                for (numVals = position = 0; (position < NetworkListener.DATA_SIZE)
                                && (numVals < 2);) {
                        char type = (char) bytes[position];
                        if (type == 'i') {
                                byte sh[] = new byte[4];
                                sh[0] = bytes[position + 4];
                                sh[1] = bytes[position + 3];
                                sh[2] = bytes[position + 2];
                                sh[3] = bytes[position + 1];

                                vals[numVals++] = ByteBuffer.wrap(sh).asIntBuffer().get();

                                position += 5;
                        } else if (type == 0) {
                                break;
                        } else {
                                System.err.println("Got a unexpected data flag in BMP085: "
                                                + type);
                                return;
                        }
                }

                float conv1 = 0f, conv2 = 0f;

                conv1 = pressureConversion(vals[0]);
                conv2 = temperatureConversion(vals[1]);

                Triple<Time, Pressure, Temperature> curData = new Triple<Time, Pressure,
Temperature>(
                                new Time(time), new Pressure(conv1), new Temperature(conv2));
                data.add(curData);

                if (activeView == viewType.GRAPH) {
                        graphPanel.addData(curData);
                } else if (activeView == viewType.TABLE) {
                        tablePanel.addData(curData);
                } else if (activeView == viewType.SPECIFIC) {
                        specificPanel.addData(curData);
                }
        }

        /*
         * (non-Javadoc)
         *
         * @see wsn.sensor.Sensor#getNewInstance()
         */
        @Override
        public Sensor getNewInstance() {
                return new BMP085();
        }

        /**
         * @deprecated
         *
         * @param val
         * @return The converted pressure value
         */
        private float pressureConversion(int val) {
                return val;
        }

        /**
         * temperatureConversion converts the raw sensorData val to its correct
         * value.
         *
         * @param val
         *            The raw sensor data.
         * @return The converted sensor data.
         */
        private float temperatureConversion(int val) {
                return val / 10f;
        }

        /**
```

```
         * @param type
         *            The requestType to change the sensor to request.
         */
        public void setRequestType(DataReqType type) {
                if (reqType != type) {
                        reqType = type;

                        setChanged();
                        notifyObservers();
                }
        }

        /**
         * @return The current requestType this Sensor is using.
         */
        public DataReqType getRequestType() {
                return reqType;
        }
}
```

```java
package wsn.sensor.sideview;

import java.awt.Dimension;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

import javax.swing.ButtonGroup;
import javax.swing.JPanel;
import javax.swing.JRadioButton;

import net.miginfocom.swing.MigLayout;
import wsn.sensor.BMP085;
import wsn.sensor.Sensor;

/**
 * @author jshirley
 *
 */
public class BMP085SideView extends JPanel implements SensorSideView {

        /**
         * The sensor that this sideView is using for a model.
         */
        private BMP085 host;
        /**
         * The group of radio buttons including rdbtnPressue and rdbtnTemperature.
         */
        private ButtonGroup bg;
        /**
         * The ActionListener that is triggered when either rdbtnPressue or
         * rdbtnTemperature are clicked
         */
        private ActionListener buttonClicked;
        /**
         * The JRadioButton for specifying that Pressure should be requested and
         * viewed.
         */
        private JRadioButton rdbtnPressure;
        /**
         * The JRadioButton for specifying that Temperature should be requested and
         * viewed.
         */
        private JRadioButton rdbtnTemperature;

        /**
         * Creating a new BMP085SideView lays out and initializes its components. It
         * also sets the Sensor that it is modeling to the Sensor parameter s.
         *
         * @param s
         *            The Sensor this SideView uses as a model.
         */
        public BMP085SideView(Sensor s) {
                if (s instanceof BMP085) {
                        this.host = (BMP085) s;
                }

                bg = new ButtonGroup();
                buttonClicked = new ActionListener() {
                        @Override
                        public void actionPerformed(ActionEvent arg0) {
                                if (rdbtnPressure.isSelected()) {
                                        host.setRequestType(BMP085.DataReqType.PRESSURE);
                                } else {
                                        host.setRequestType(BMP085.DataReqType.TEMPERATURE);
                                }
                        }
                };

                this.setMinimumSize(new Dimension(300, 60));
                this.setMaximumSize(new Dimension(300, 60));
                this.setPreferredSize(new Dimension(300, 60));
                setLayout(new MigLayout("", "[grow]", "[][]"));

                rdbtnPressure = new JRadioButton("Pressure");
                rdbtnPressure.addActionListener(buttonClicked);
                add(rdbtnPressure, "cell 0 0");
                rdbtnPressure.setSelected(true);

                rdbtnTemperature = new JRadioButton("Temperature");
                rdbtnTemperature.addActionListener(buttonClicked);
                add(rdbtnTemperature, "cell 0 1");
                rdbtnTemperature.setSelected(false);

                bg.add(rdbtnPressure);
                bg.add(rdbtnTemperature);
        }

        /*
         * (non-Javadoc)
         *
         * @see wsn.sensor.sideview.SensorSideView#setSensor(wsn.sensor.Sensor)
         */
        @Override
        public void setSensor(Sensor s) {
                if (s instanceof BMP085) {
                        this.host = (BMP085) s;
                }
        }

        /*
         * (non-Javadoc)
         *
         * @see wsn.sensor.sideview.SensorSideView#getSensor()
         */
        @Override
        public Sensor getSensor() {
                return this.host;
        }

        /*
         * (non-Javadoc)
         *
         * @see wsn.sensor.sideview.SensorSideView#getPanel()
         */
        @Override
        public JPanel getPanel() {
                return this;
        }
}
```

```
package wsn.sensor.view;

import java.util.Observable;

import javax.swing.JPanel;

import wsn.sensor.Sensor;



/**
 * @deprecated
 * @author jshirley
 *
 */
public class BMP085TableView implements SensorView
{
        public BMP085TableView(Sensor s)
        {

        }

        @Override
        public JPanel getPanel() {
                // TODO Auto-generated method stub
                return null;
        }

        @Override
        public void setSensor(Sensor s) {
                // TODO Auto-generated method stub

        }

        @Override
        public Sensor getSensor() {
                // TODO Auto-generated method stub
                return null;
        }

        @Override
        public void addData(Object o) {
                // TODO Auto-generated method stub

        }

        @Override
        public void update(Observable o, Object arg) {
                // TODO Auto-generated method stub

        }

}
```

```java
package wsn.popups;

import java.awt.FlowLayout;
import java.awt.GridBagConstraints;
import java.awt.GridBagLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.ItemEvent;
import java.awt.event.ItemListener;
import java.io.FileWriter;
import java.io.IOException;
import java.util.ArrayList;

import javax.swing.JButton;
import javax.swing.JCheckBox;
import javax.swing.JDialog;
import javax.swing.JFileChooser;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JSeparator;
import javax.swing.SwingConstants;
import javax.swing.border.EmptyBorder;
import javax.swing.filechooser.FileNameExtensionFilter;

import net.miginfocom.swing.MigLayout;
import wsn.Globals;
import wsn.sensor.Sensor;
import wsn.util.Triple;
import wsn.util.Tuple;

import com.google.gson.JsonArray;
import com.google.gson.JsonObject;

/**
 * A popup that manages saving Sensor backup files (.slog)
 *
 * @author jshirley
 *
 */
public class ExportDialog extends JDialog {

        /**
         * The panel to contain the ExportDialog components on.
         */
        private final JPanel contentPanel = new JPanel();
        /**
         * A panel containing all of the active nodes address/Sensor pairings and
         * checkmarks which determine if that pairing will be exported.
         */
        private JPanel panelNodes;
        /**
         * Represents the number of checkmarks that are "checked".
         */
        private int itemsSelected;

        /**
         * Represents the Triple of the short node address, the Sensor and the
         * JCheckbox pairing.
         */
        private ArrayList<Triple<JCheckBox, Short, Sensor>> nodes;
        /**
         * The button that triggers the export process when clicked
         */
        private JButton okButton;

        /**
         * Creating a new ExportDialog lays out and initializes its components.
         */
        public ExportDialog() {
                setTitle("Export");
                itemsSelected = 0;
                nodes = new ArrayList<Triple<JCheckBox, Short, Sensor>>();

                setBounds(100, 100, 450, 300);
                getContentPane().setLayout(
                                new MigLayout("", "[448px]", "[232px][35px]"));
                contentPanel.setBorder(new EmptyBorder(5, 5, 5, 5));
                getContentPane().add(contentPanel, "cell 0 0,grow");
                contentPanel.setLayout(new MigLayout("", "[grow]", "[][grow]"));
                {
                        JLabel lblSelectNodeData = new JLabel("Select Node Data to Export");
                        contentPanel.add(lblSelectNodeData, "cell 0 0");
                }
                {
                        JScrollPane scrollPane = new JScrollPane();
                        contentPanel.add(scrollPane, "cell 0 1,grow");
                        {
                                panelNodes = new JPanel();
                                panelNodes.setLayout(new GridBagLayout());
                                scrollPane.setViewportView(panelNodes);
                        }
                }
                {
                        JPanel buttonPane = new JPanel();
                        buttonPane.setLayout(new FlowLayout(FlowLayout.RIGHT));
                        getContentPane().add(buttonPane, "cell 0 1,growx,aligny top");
                        {
                                okButton = new JButton("OK");
                                okButton.setEnabled(false);
                                okButton.addActionListener(new ActionListener() {
                                        @Override
                                        public void actionPerformed(ActionEvent arg0) {
                                                export();
                                        }
                                });
                                okButton.setActionCommand("OK");
                                buttonPane.add(okButton);
                                getRootPane().setDefaultButton(okButton);
                        }
                        {
                                JButton cancelButton = new JButton("Cancel");
                                cancelButton.addActionListener(new ActionListener() {

                                        @Override
                                        public void actionPerformed(ActionEvent arg0) {
                                                setVisible(false);

                                        }
                                });
                                cancelButton.setActionCommand("Cancel");
                                buttonPane.add(cancelButton);
                        }
                }
                this.setModal(true);

                updatePanel();
        }

        /**
         * updatePanel redraws the node address, Sensor and checkbox elements and
         * should be called if a new node is added while this dialog is visible.
         */
        public void updatePanel() {
                GridBagConstraints gbc = new GridBagConstraints();
                gbc.gridx = 0; // single col
                gbc.fill = GridBagConstraints.HORIZONTAL;
                nodes.clear();

                int indx = 0;
        for (Tuple<Short, Sensor> t : Globals.getInstance().getNodes()) {
            // add seperators if more than one node
            if (indx > 0) {
                gbc.gridy = indx++;
                panelNodes.add(new JSeparator(SwingConstants.HORIZONTAL), gbc);
            }

            gbc.gridy = indx++;

            JCheckBox chk = new JCheckBox(t.x + ": " + t.y.getSensorName());
            chk.addItemListener(new ItemListener() {

                @Override
                public void itemStateChanged(ItemEvent arg0) {
                    if (arg0.getStateChange() == ItemEvent.DESELECTED) {
                        itemsSelected--;
                        if (itemsSelected == 0) {
                            okButton.setEnabled(false);
                        }
                    } else if (arg0.getStateChange() == ItemEvent.SELECTED) {
                        itemsSelected++;
                        okButton.setEnabled(true);

                    }

                }
            });
            nodes.add(new Triple<JCheckBox, Short, Sensor>(chk, t.x, t.y));
            panelNodes.add(chk, gbc);
        }
        revalidate();
        repaint();
    }

    /**
     * export launches a dialog querying the user for a file. It then writes all
     * of the checked node address and Sensor pairings to the file specified.
     */
    private void export() {
        JFileChooser fc = new JFileChooser();
        fc.setFileFilter(new FileNameExtensionFilter("Sensor Logs", "slog"));
        int ret = fc.showSaveDialog(null);

        if (ret == JFileChooser.APPROVE_OPTION) {
            JsonArray jNodes = new JsonArray();
            JsonObject jNode;
            // write it in JSON
            for (Triple<JCheckBox, Short, Sensor> node : nodes) {

                if (node.x.isEnabled()) {
                    jNode = new JsonObject();
                    jNode.addProperty("nodeId", node.y);
                    jNode.addProperty("nodeType", node.z.getSensorName());
                    jNode.addProperty("nodeData", node.z.getBackingData());
                    jNodes.add(jNode);
                }
            }
            if (!fc.getSelectedFile().exists()) {
                try {
                    fc.getSelectedFile().createNewFile();
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
            FileWriter fw;
            try {
                fw = new FileWriter(fc.getSelectedFile());
                fw.write(jNodes.toString());
                fw.close();
            } catch (IOException e) {
```

```
                                e.printStackTrace();
                        }

                }
                setVisible(false);
        }
}
```

```java
package wsn;

import java.lang.reflect.Constructor;
import java.util.ArrayList;
import java.util.Observable;
import java.util.Set;
import java.util.TreeSet;
import java.util.concurrent.Semaphore;
import java.util.logging.Level;
import java.util.logging.Logger;
import java.util.prefs.Preferences;

import javax.swing.SwingWorker;

import org.reflections.Reflections;

import wsn.network.Network;
import wsn.network.NetworkListener;
import wsn.run.Run;
import wsn.sensor.Sensor;
import wsn.util.Tuple;

/**
 * Globals contains methods and data that are needed in most parts of the
 * project. This mainly consists of the active sensor/node address pairings.
 * Globals can also be used as a central event dispatching class.
 *
 * @author jshirley
 *
 */
public class Globals extends Observable {
	/**
	 * Represents the run that is currently displayed.
	 */
	private Run activeRun = null;
	/**
	 * Represents the sensor whose view is currently displayed.
	 */
	private Sensor activeSensor = null;

	/**
	 * Represents the list of runs that are currently active, this is currently
	 * limited to 1 run.
	 */
	private ArrayList<Run> runs = new ArrayList<>();
	/**
	 * Represents the pairing of node address to sensor.
	 */
	private static TreeSet<Tuple<Short, Sensor>> nodes = new TreeSet<Tuple<Short, Sensor>>();

	/**
	 * Represents the Sensor classes that were found via reflection.
	 */
	private static ArrayList<Sensor> availableSensors = new ArrayList<Sensor>();

	/**
	 * Represents the single instance of this Class (Globals) that exists.
	 */
	private static Globals instance = null;
	/**
	 * Represents the preferences for the logged in user.
	 */
	private Preferences prefs = Preferences.userRoot();

	/**
	 * Represents the lock that is used to guarantee only one user can access
	 * the network at a given time. Not necessary if the hardware allows for 2
	 * way communication.
	 */
	private static final Semaphore sem = new Semaphore(1);
```

```
        /**
         * Represents whether the frontend is requesting data from the nodes.
         */
        private static boolean nodeRequests = false;

        /**
         * Represents the thread that is requesting data from the nodes.
         */
        private static Thread dataRequestThread;

        public enum Events {
                NODE_ADDED, NODE_REMOVED, ACTIVE_SENSOR_CHANGED, RUN_ADDED, RUN_REMOVED
        };

        /**
         * Represents the largest number of active runs allowed.
         */
        public final static int MAX_RUNS = 1;

        /**
         * Constructing the Globals class adds an instance of each Sensor class
         * found in the wsn.sensor to availableSensors.
         */
        private Globals() {
                // use wsn.sensor package for base of reflections
                Reflections reflections = new Reflections("wsn.sensor");

                Set<Class<? extends Sensor>> subTypes = reflections
                                .getSubTypesOf(Sensor.class);

                for (Class<? extends Sensor> subType : subTypes) {
                        try {
                                Constructor<? extends Sensor> constructor = subType
                                                .getConstructor();
                                availableSensors.add(constructor.newInstance());
                        } catch (Exception ex) {
                                Logger.getLogger(Globals.class.getName()).log(Level.SEVERE,
                                                null, ex);
                        }
                }
        }

        /**
         * addNode attempts to configure the node at nodeId address to the sensor
         * specified by the String sensor parameter.
         *
         * @param nodeId
         *            represents the address of the node.
         * @param sensor
         *            represents the name of the sensor to configure the node as.
         */
        public void addNode(final short nodeId, final String sensor) {

                // use the SwingWorker thread so we don't lock up the UI thread
                SwingWorker<Boolean, Object> sw = new SwingWorker<Boolean, Object>() {

                        @Override
                        protected Boolean doInBackground() throws Exception {
                                Globals.getInstance().getNetworkLock();
                                NetworkListener.getInstance().sendPacket(
                                                Network.flags.SENSOR_CONFIG_REQ.getValue(), nodeId,
                                                sensor);

                                if (NetworkListener.getInstance().expectPacket(
                                                Network.flags.SENSOR_CONFIG_REQ.getValue())) {
                                        System.err.println("expecting sensor: "
                                                        + NetworkListener.getInstance
().getPacketData());
                                        for (Sensor s : availableSensors) {
                                                if (s.getSensorName().equals(sensor)) {
```

```java
                                                        nodes.add(new Tuple<Short, Sensor>(nodeId, s
                                                                        .getNewInstance()));
                                                        Globals.getInstance().fireEvent
(Events.NODE_ADDED);
                                                        break;
                                                }
                                        }
                                }
                                Globals.getInstance().releaseNetworkLock();
                                return true;
                        }

                };
                sw.execute();
        }

        /**
         * removeNode stops current data requests from the node identified by
         * nodeId. All future data requests do not include this node. This also
         * results in the node being removed from the GUI.
         *
         * @param nodeId
         *            represents the address of the node.
         */
        public void removeNode(short nodeId) {
                for (Tuple<Short, Sensor> t : nodes) {
                        if (t.x == nodeId) {
                                nodes.remove(t);
                                fireEvent(Events.NODE_REMOVED);
                                break;
                        }
                }
                if (nodes.size() == 0) {
                        stopNodeRequests();
                }
        }

        /**
         * startNodeRequests causes requests for data to be sent out on all of the
         * nodes that have been added via the addNode method.
         */
        public void startNodeRequests() {
                nodeRequests = true;

                // use a different thread besides SwingWorker so that nodes can be added
                // at the same time.
                dataRequestThread = new Thread() {
                        @Override
                        public void run() {
                                while (nodeRequests) {
                                        for (final Tuple<Short, Sensor> t : nodes) {
                                                // request data for each node address/Sensor pairing
                                                t.y.requestData(NetworkListener.getInstance(), t.x);
                                        }
                                }
                                return;
                        };
                };
                dataRequestThread.start();
        }

        /**
         * stopNodeRequests causes requests for data to stop after the latest cycle
         * of all Node requests has finished.
         */
        public void stopNodeRequests() {
                nodeRequests = false;
        }

        /**
```

```java
         * @return TreeSet of the currently configured node address and sensor
         *         pairings.
         */
        public TreeSet<Tuple<Short, Sensor>> getNodes() {
                return new TreeSet<Tuple<Short, Sensor>>(nodes);
        }

        /**
         * @return The currently viewed Run.
         */
        public Run getActiveRun() {
                return activeRun;
        }

        /**
         * @return The list of instances of the Sensor Classes found at runtime.
         */
        public ArrayList<Sensor> getAvailableSensors() {
                return new ArrayList<Sensor>(availableSensors);
        }

        /**
         * @return The Rreferences object associated with the logged in user.
         */
        public Preferences getUserPreferences() {
                return prefs;
        }

        /**
         * @return A list of the names of all of the Sensors found via reflection.
         */
        public ArrayList<String> getSensorNames() {
                ArrayList<String> sensorNames = new ArrayList<String>();
                for (Sensor s : Globals.getInstance().getAvailableSensors()) {
                        sensorNames.add(s.getSensorName());
                }
                return sensorNames;
        }

        /**
         * @return The single instance of this Class allowed.
         */
        public static Globals getInstance() {
                if (instance == null) {
                        instance = new Globals();
                }
                return instance;
        }

        /**
         * @return ArrayList of all of the active Runs.
         */
        public ArrayList<Run> getRuns() {
                return runs;
        }

        /**
         * setActiveSensor changes the view to the activeView of the Sensor sensor
         * parameter.
         *
         * @param sensor
         *            Represents the Sensor whose active view should be displayed.
         */
        public void setActiveSensor(Sensor sensor) {
                if (sensor != activeSensor) {
                        activeSensor = sensor;
                        fireEvent(Events.ACTIVE_SENSOR_CHANGED);
                }

        }

        /**
         * @return The Sensor whose activeView is currently displayed.
         */
        public Sensor getActiveSensor() {
                return activeSensor;
        }

        /**
         * @param e
         *            The Event for Globals to fire.
         */
        public void fireEvent(Events e) {
                this.setChanged();
                this.notifyObservers(e);
        }

        /**
         * addRun Adds a new Run for Globals to watch. It also sets that run as the
         * active run.
         *
         * @param run
         *            Represents the Run to add and make active.
         */
        public void addRun(Run run) {
                activeRun = run;
                runs.add(run);

                fireEvent(Events.RUN_ADDED);
        }

        /**
         * removeActiveRun Removes the currently displayed Run and sets the
         * activeRun to the first Run.
         */
        public void removeActiveRun() {
                runs.remove(activeRun);
                if (runs.size() > 0) {
                        activeRun = runs.get(0);
                } else {
                        activeRun = null;
                }
                fireEvent(Events.RUN_REMOVED);
        }

        /**
         * getNetworkLock Acquires a lock for communicating to the Poly_Sense
         * Server.
         */
        public void getNetworkLock() {
                try {
                        sem.acquire(1);
                } catch (InterruptedException e) {
                        e.printStackTrace();
                }
        }

        /**
         * releaseNetworkLock Releases the lock for communicating to the Poly_Sense
         * Server.
         */
        public void releaseNetworkLock() {
                sem.release(1);
        }
}
```

```java
package wsn.popups;

import javax.swing.JFileChooser;
import javax.swing.filechooser.FileNameExtensionFilter;

/**
 * A popup that manages importing a Sensor backup file (.slog).
 * 
 * @author jshirley
 * 
 */
public class ImportDialog {
        /**
         * Creating a new ImportDialog launches a window that query's the user for a
         * file. It then parses the file for node address/Sensor pairings and
         * attempts reinitialize the node address/Sensor pairing to the state
         * described in the file. This includes configuring nodes.
         */
        public ImportDialog() {
                JFileChooser fc = new JFileChooser();
                FileNameExtensionFilter ff = new FileNameExtensionFilter("Sensor Logs",
                                "slog");
                fc.setFileFilter(ff);

                int ret = fc.showOpenDialog(null);
                if (ret == JFileChooser.APPROVE_OPTION) {
                        // TODO: based on sensor names, create new instances of those
                        // sensors, initialize them to node id in file, set thedata
                }
        }
}
```

```java
package wsn.sensor.view;

import java.util.Observable;
import java.util.Observer;

import javax.swing.JPanel;

import org.jfree.chart.ChartFactory;
import org.jfree.chart.ChartPanel;
import org.jfree.chart.JFreeChart;
import org.jfree.chart.plot.PlotOrientation;
import org.jfree.data.xy.XYSeries;
import org.jfree.data.xy.XYSeriesCollection;

import wsn.Globals;
import wsn.sensor.MPU6050;
import wsn.sensor.Sensor;
import wsn.sensor.types.Acceleration;
import wsn.sensor.types.Time;
import wsn.sensor.types.Velocity;
import wsn.util.Septuple;

/**
 * @see SensorView
 *
 * @author jshirley
 *
 */
public class MPU6050GraphView implements SensorView, Observer{

        /**
         * The panel that represents this MPU6050GraphView.
         */
        private JPanel panel = new JPanel();

        /**
         * The acceleration chart.
         */
        private JFreeChart chartAccel;
        /**
         * The velocity chart.
         */
        private JFreeChart chartVelocity;

        /**
         * Graph label.
         */
        private static String accelXMeasurement = "time (s)";
        /**
         * Graph label.
         */
        private static String accelYMeasurement = "acceleration (m/s^2)";
        /**
         * Graph label.
         */
        private static String velocityXMeasurement = "time (s)";
        /**
         * Graph label.
         */
        private static String velocityYMeasurement = "radsps (rad/s)";

        /**
         * Graph label.
         */
        public static String sensorName = MPU6050.sensorName;

        /**
         * Collection wrapper of the one series that is present on the acceleration graph.
         */
        private XYSeriesCollection accelDataCollection;
        /**
         * Collection wrapper of the one series that is present on the velocity graph.
         */
        private XYSeriesCollection velocityDataCollection;

        /**
         * Series of X and Y pairs that represent the acceleration in the x direction and time pairs.
         */
        private XYSeries accelX;
        /**
         * Series of X and Y pairs that represent the acceleration in the y direction and time pairs.
         */
        private XYSeries accelY;
        /**
         * Series of X and Y pairs that represent the acceleration in the z direction and time pairs.
         */
        private XYSeries accelZ;

        /**
         * Series of X and Y pairs that represent the velocity in the x direction and time pairs.
         */
        private XYSeries velocityX;
        /**
         * Series of X and Y pairs that represent the velocity in the y direction and time pairs.
         */
        private XYSeries velocityY;
        /**
         * Series of X and Y pairs that represent the velocity in the z direction and time pairs.
         */
        private XYSeries velocityZ;
        /**
         * Represents the maximum number of time measurements allowed on the graph
         * at a given time. If this is exceeded the oldest elements are removed.
         */
        private static final int xRange = 100;

        /**
         * The Sensor that this MPU6050GraphView uses as a model.
         */
        private MPU6050 host;

        /**
         * The dataRequestType determines whether this graph is showing accelerometer data or gyroscope data.
         */
        private MPU6050.DataReqType dataType = MPU6050.DataReqType.ACCELEROMETER;

        /**
         * Creating a new MPU6050GraphView lays out and initializes its components. It
         * also sets the Sensor that it is modeling to the Sensor parameter s.
         *
         * @param s
         *            The Sensor this SensorView uses as a model.
         */
        public MPU6050GraphView(Sensor s) {
                host = (MPU6050) s;
                host.addObserver(this);

                accelDataCollection = new XYSeriesCollection();
                velocityDataCollection = new XYSeriesCollection();

                accelX = new XYSeries("accelX");
                accelY = new XYSeries("accelY");
                accelZ = new XYSeries("accelZ");

                velocityX = new XYSeries("velocityX");
                velocityY = new XYSeries("velocityY");
                velocityZ = new XYSeries("velocityZ");

                accelX.setMaximumItemCount(xRange);
                accelY.setMaximumItemCount(xRange);
                accelZ.setMaximumItemCount(xRange);

                velocityX.setMaximumItemCount(xRange);
                velocityY.setMaximumItemCount(xRange);
                velocityZ.setMaximumItemCount(xRange);

                accelDataCollection.addSeries(accelX);
                accelDataCollection.addSeries(accelY);
                accelDataCollection.addSeries(accelZ);

                velocityDataCollection.addSeries(velocityX);
                velocityDataCollection.addSeries(velocityY);
                velocityDataCollection.addSeries(velocityZ);

                chartAccel = ChartFactory.createXYLineChart(sensorName + " Graph: Acceleration", // chart
title
                                accelXMeasurement, // x axis label
                                accelYMeasurement, // y axis label
                                accelDataCollection, // data
                                PlotOrientation.VERTICAL, true, // include legend
                                true, // tooltips
                                false // urls
                                );

                chartVelocity = ChartFactory.createXYLineChart(sensorName + " Graph: Velocity", // chart
title
                                velocityXMeasurement, // x axis label
                                velocityYMeasurement, // y axis label
                                velocityDataCollection, // data
                                PlotOrientation.VERTICAL, true, // include legend
                                true, // tooltips
                                false // urls
                                );

                if (host.getRequestType() == MPU6050.DataReqType.ACCELEROMETER) {
                        panel = new ChartPanel(chartAccel);
                } else {
                        panel = new ChartPanel(chartVelocity);
                }
        }

        /* (non-Javadoc)
         * @see wsn.sensor.view.SensorView#getPanel()
         */
        @Override
        public JPanel getPanel() {
                return panel;
        }

        /* (non-Javadoc)
         * @see wsn.sensor.view.SensorView#setSensor(wsn.sensor.Sensor)
         */
        @Override
        public void setSensor(Sensor s) {
                this.host = (MPU6050) s;

        }

        /* (non-Javadoc)
         * @see wsn.sensor.view.SensorView#getSensor()
         */
        @Override
```

```java
        public Sensor getSensor() {
                return host;
        }

        /* (non-Javadoc)
         * @see wsn.sensor.view.SensorView#addData(java.lang.Object)
         */
        @Override
        public void addData(Object o) {
                if (o instanceof Septuple) {
                        Time t = (Time) ((Septuple) o).t;
                        if (host.getRequestType() == MPU6050.DataReqType.ACCELEROMETER) {
                                Acceleration ax = (Acceleration) ((Septuple) o).x;
                                Acceleration ay = (Acceleration) ((Septuple) o).y;
                                Acceleration az = (Acceleration) ((Septuple) o).z;

                                accelX.add(t.getTime()/10, ax.getAcceleration());
                                accelY.add(t.getTime()/10, ay.getAcceleration());
                                accelZ.add(t.getTime()/10, az.getAcceleration());
                        } else if (host.getRequestType() == MPU6050.DataReqType.GYROSCOPE) {
                                Velocity vx = (Velocity) ((Septuple) o).u;
                                Velocity vy = (Velocity) ((Septuple) o).v;
                                Velocity vz = (Velocity) ((Septuple) o).w;

                                velocityX.add(t.getTime()/10, vx.getVelocity());
                                velocityY.add(t.getTime()/10, vy.getVelocity());
                                velocityZ.add(t.getTime()/10, vz.getVelocity());
                        }
                }

        }
        /* (non-Javadoc)
         * @see java.util.Observer#update(java.util.Observable, java.lang.Object)
         */
        @Override
        public void update(Observable o, Object arg) {
                dataType = host.getRequestType();
                if (dataType == MPU6050.DataReqType.ACCELEROMETER) {
                        panel = new ChartPanel(chartAccel);
                } else {
                        panel = new ChartPanel(chartVelocity);
                }
                System.err.println("chaning view for MPU6050 graph to \n" + dataType);
                Globals.getInstance().fireEvent(Globals.Events.ACTIVE_SENSOR_CHANGED); // hacky...
        }
}
```

```java
package wsn.sensor;

import java.nio.ByteBuffer;
import java.util.ArrayList;

import javax.swing.JPanel;

import wsn.Globals;
import wsn.network.Network;
import wsn.network.NetworkListener;
import wsn.sensor.sideview.MPU6050SideView;
import wsn.sensor.sideview.SensorSideView;
import wsn.sensor.types.Acceleration;
import wsn.sensor.types.Time;
import wsn.sensor.types.Velocity;
import wsn.sensor.view.MPU6050GraphView;
import wsn.sensor.view.MPU6050TableView;
import wsn.sensor.view.SensorView;
import wsn.util.Septuple;

import com.google.gson.JsonArray;
import com.google.gson.JsonObject;

/**
 * @see Sensor
 *
 * @author jshirley
 *
 */
public class MPU6050 extends Sensor {

        /**
         * This Sensor's name.
         */
        public static final String sensorName = "MPU6050";
        /**
         * The current view that this Sensor will display if it is the active
         * sensor.
         */
        private viewType activeView = null;

        /**
         * Represents the list of views that this sensor supports.
         */
        private ArrayList<viewType> sensorViewTypes = new ArrayList<viewType>();

        /**
         * All of the data that has been collected by this Sensor.
         */
        private ArrayList<Septuple<Time, Velocity, Velocity, Velocity, Acceleration, Acceleration,
Acceleration>> data;
        /**
         * The current data request type that this sensor is using, can be either
         * accelerometer or gyroscope. Only one type can be retrieved from the
         * Sensor at a time and only one type can be viewed at a time.
         */
        private DataReqType reqType = DataReqType.ACCELEROMETER;

        /**
         * Panel for graph view.
         */
        private MPU6050GraphView graphPanel;
        /**
         * Panel for table view.
         */
        private MPU6050TableView tablePanel;
        /**
         * Panel for specific view.
         */
        private SensorView specificPanel;
        /**
         * Panel for the sideView.
         */
        private SensorSideView sidePanel;

        /**
         * Value used in conversion.
         */
        private static final double gyro_sensitivity = 66.5;
        /**
         * Value used in conversion.
         */
        private static final double accel_sensitivity = 0.00006;
        /**
         * Value used in conversion.
         */
        private static final double dt = 0.002;
        /**
         * Value used in conversion.
         */
        private static final double dps_to_radps = 57.29578;
        /**
         * Value used in conversion.
         */
        private static final double g = 9.81;

        public enum DataReqType {
                GYROSCOPE('g'), ACCELEROMETER('a');

                private char val;

                private DataReqType(char c) {
                        this.val = c;
                }

                public char getValue() {
                        return val;
                }
        };

        /**
         * The number of times this sensor has not recieved data when it expected
         * data. If > 5 then the Sensor state can be described as "Disconnected".
         */
        private int numFailedExpects;

        /**
         * Creating a new MPU6050 initializes its supported types and its backing
         * data field as well as preparing its supported viewTypes.
         */
        public MPU6050() {
                sensorViewTypes.add(viewType.GRAPH);
                sensorViewTypes.add(viewType.TABLE);
                sensorViewTypes.add(viewType.SPECIFIC);

                data = new ArrayList<Septuple<Time, Velocity, Velocity, Velocity, Acceleration, Acceleration, Acceleration>>();

                graphPanel = new MPU6050GraphView(this);
                tablePanel = new MPU6050TableView(this);
                specificPanel = null;

                sidePanel = new MPU6050SideView(this);

                numFailedExpects = 0;
        }

        /**
         * @return The current requestType this Sensor is using.
         */
        public DataReqType getRequestType() {
                return reqType;
        }

        /**
         * @param r
         *            The requestType to change the sensor to request.
         */
        public void setRequestType(DataReqType r) {
                if (r != reqType) {
                        reqType = r;
                        this.setChanged();
                        this.notifyObservers();
                }
        }

        /*
         * (non-Javadoc)
         *
         * @see wsn.sensor.Sensor#getSensorName()
         */
        @Override
        public String getSensorName() {
                return sensorName;
        }

        /*
         * (non-Javadoc)
         *
         * @see wsn.sensor.Sensor#getViewTypes()
         */
        @Override
        public ArrayList<viewType> getViewTypes() {
                return sensorViewTypes;
        }

        /*
         * (non-Javadoc)
         *
         * @see wsn.sensor.Sensor#setActiveView(wsn.sensor.Sensor.viewType)
         */
        @Override
        public void setActiveView(viewType v) {
                activeView = v;

        }

        /*
         * (non-Javadoc)
         *
         * @see wsn.sensor.Sensor#getActiveView()
         */
        @Override
        public JPanel getActiveView() {
                if (activeView == viewType.GRAPH) {
                        return graphPanel.getPanel();

                } else if (activeView == viewType.TABLE) {
                        return tablePanel.getPanel();
                } else if (activeView == viewType.SPECIFIC) {
                        return specificPanel.getPanel();
                }
                return null;
        }

        /*
         * (non-Javadoc)
         *
         * @see wsn.sensor.Sensor#getSidePanel()
         */
```

```java
    @Override
    public JPanel getSidePanel() {
        return sidePanel.getPanel();
    }

    /*
     * (non-Javadoc)
     *
     * @see wsn.sensor.Sensor#getStatusString()
     */
    @Override
    public String getStatusString() {
        if (numFailedExpects < 5) {
            return Sensor.sensorStates.CONNECTED.toString();
        }
        return Sensor.sensorStates.DISCONNECTED.toString();
    }

    /*
     * (non-Javadoc)
     *
     * @see wsn.sensor.Sensor#requestData(wsn.network.NetworkListener, short)
     */
    @Override
    public void requestData(NetworkListener n, short addr) {
        Globals.getInstance().getNetworkLock();
        n.sendPacket(Network.flags.SENSOR_DATA.getValue(), addr, new String(
                reqType.getValue() + ""));
        if (n.expectPacket(Network.flags.SENSOR_DATA.getValue())) {
            if (NetworkListener.getInstance().getPacketNodeId() == addr) {
                addData(NetworkListener.getInstance().getPacketData());
                numFailedExpects = 0;
            } else {
                System.err.println("unexpected SensorId: "
                        + NetworkListener.getInstance().getPacketNodeId()
                        + " expected: " + addr);
            }
        } else {
            numFailedExpects++;
        }
        Globals.getInstance().releaseNetworkLock();
    }

    /*
     * (non-Javadoc)
     *
     * @see wsn.sensor.Sensor#getBackingData()
     */
    @Override
    public String getBackingData() {
        JsonArray jAllData = new JsonArray();
        JsonObject jData;

        for (Septuple<Time, Velocity, Velocity, Velocity, Acceleration, Acceleration, Acceleration> septuple : data) {
            jData = new JsonObject();
            if (septuple.t != null) {
                jData.addProperty("time", septuple.t.getTime());
            } else {
                jData.addProperty("time", "");
            }
            if (septuple.u != null) {
                jData.addProperty("vx", septuple.u.getVelocity());
            } else {
                jData.addProperty("time", "");
            }
            if (septuple.v != null) {
                jData.addProperty("vy", septuple.v.getVelocity());
            } else {
                jData.addProperty("time", "");
            }
            if (septuple.w != null) {
                jData.addProperty("vz", septuple.w.getVelocity());
            } else {
                jData.addProperty("time", "");
            }
            if (septuple.x != null) {
                jData.addProperty("ax", septuple.x.getAcceleration());
            } else {
                jData.addProperty("time", "");
            }
            if (septuple.y != null) {
                jData.addProperty("ay", septuple.y.getAcceleration());
            } else {
                jData.addProperty("time", "");
            }
            if (septuple.z != null) {
                jData.addProperty("az", septuple.z.getAcceleration());
            } else {
                jData.addProperty("time", "");
            }
            jAllData.add(jData);

        }
        return jAllData.toString();
    }

    /*
     * (non-Javadoc)
     *
     * @see wsn.sensor.Sensor#setData(java.lang.String)
     */
    @Override
    public void setData(String data) {
        // TODO Auto-generated method stub

    }

    /*
     * Expects [s,data,data,s,data,data,s,data,data]
     *
     * @see wsn.sensor.Sensor#addData(byte[])
     */
    @Override
    public void addData(byte[] bytes) {
        long time = Globals.getInstance().getActiveRun().getTime();

        short vals[] = { 0, 0, 0 };

        int position, numVals;

        for (numVals = position = 0; (position < NetworkListener.DATA_SIZE)
                && (numVals < 3);) {
            byte type = bytes[position];
            if (type == 's') {
                byte sh[] = new byte[2];
                sh[0] = bytes[position + 2];
                sh[1] = bytes[position + 1];

                vals[numVals++] = ByteBuffer.wrap(sh).asShortBuffer().get();

                position += 3;
                System.err.println("got an expected short flag");
            } else if (type == 0) {
                System.err.println("end of data");
                break;
            } else {
                System.err
                        .println("got something unexpected in the data type
memory location: "
```

```
                                + bytes[position]);
                return;
            }
        }
        System.err.println("got short1: " + vals[0] + " short2: " + vals[1]
                + " short3: " + vals[2]);

        float conv1 = 0f, conv2 = 0f, conv3 = 0f;

        if (reqType == DataReqType.ACCELEROMETER) {
            conv1 = accelerometerConv(vals[0]);
            conv2 = accelerometerConv(vals[1]);
            conv3 = accelerometerConv(vals[2]);

        } else if (reqType == DataReqType.GYROSCOPE) {
            conv1 = gyroscopeConv(vals[0]);
            conv2 = gyroscopeConv(vals[1]);
            conv3 = gyroscopeConv(vals[2]);
        }

        System.err.println("got val1: " + conv1 + " val2: " + conv2 + " val3: "
                + conv3);

        Septuple<Time, Velocity, Velocity, Velocity, Acceleration, Acceleration, Acceleration> curData = null;
        if (reqType == DataReqType.GYROSCOPE) {
            curData = new Septuple<Time, Velocity, Velocity, Velocity, Acceleration, Acceleration, Acceleration>(
                    new Time(time), new Velocity(conv1), new Velocity(conv2),
                    new Velocity(conv3), null, null, null);
            data.add(curData);

        } else if (reqType == DataReqType.ACCELEROMETER) {
            curData = new Septuple<Time, Velocity, Velocity, Velocity, Acceleration, Acceleration, Acceleration>(
                    new Time(time), null, null, null, new Acceleration(conv1),
                    new Acceleration(conv2), new Acceleration(conv3));
            data.add(curData);
        }

        if (activeView == viewType.GRAPH) {
            graphPanel.addData(curData);
        } else if (activeView == viewType.TABLE) {
            tablePanel.addData(curData);
        } else if (activeView == viewType.SPECIFIC) {
            specificPanel.addData(curData);
        }
    }

    /**
     * gyroscopeConv converts the raw sensorData val to its correct
     * value.
     *
     * @param val
     *            The raw sensor data.
     * @return The converted sensor data.
     */
    private float gyroscopeConv(short val) {
        double gyroRate = val / gyro_sensitivity;

        return (float) (gyroRate * dps_to_radps);
    }

    /**
     * accelerometerConv converts the raw sensorData val to its correct
     * value.
     *
     * @param val
     *            The raw sensor data.
     * @return The converted sensor data.
```

```
         */
        private float accelerometerConv(short val) {
                return (float) (val * accel_sensitivity * g);
        }

        /*
         * (non-Javadoc)
         *
         * @see wsn.sensor.Sensor#getNewInstance()
         */
        @Override
        public Sensor getNewInstance() {
                return new MPU6050();
        }
}
```

```java
package wsn.sensor.sideview;

import java.awt.Dimension;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

import javax.swing.ButtonGroup;
import javax.swing.JPanel;
import javax.swing.JRadioButton;

import net.miginfocom.swing.MigLayout;
import wsn.sensor.MPU6050;
import wsn.sensor.Sensor;

/**
 * @author jshirley
 *
 */
public class MPU6050SideView extends JPanel implements SensorSideView {

        /**
         * The sensor that this sideView is using for a model.
         */
        private MPU6050 host;
        /**
         * The group of radio buttons including rdbtnAccelerometer and rdbtnGyroscope.
         */
        private ButtonGroup bg;
        /**
         * The ActionListener that is triggered when either rdbtnAccelerometer or
         * rdbtnGyroscope are clicked
         */
        private ActionListener buttonClicked;
        /**
         * The JRadioButton for specifying that Accelerometer data should be requested and
         * viewed.
         */
        private JRadioButton rdbtnAccelerometer;
        /**
         * The JRadioButton for specifying that Gyroscope data should be requested and
         * viewed.
         */
        private JRadioButton rdbtnGyroscope;

        /**
         * Creating a new MPU6050SideView lays out and initializes its components. It
         * also sets the Sensor that it is modeling to the Sensor parameter s.
         *
         * @param s
         *            The Sensor this SideView uses as a model.
         */
        public MPU6050SideView(Sensor s) {
                if(s instanceof MPU6050)
                {
                        this.host = (MPU6050) s;
                }

                bg = new ButtonGroup();
                buttonClicked = new ActionListener() {
                        @Override
                        public void actionPerformed(ActionEvent arg0) {
                                if(rdbtnAccelerometer.isSelected())
                                {
                                        host.setRequestType(MPU6050.DataReqType.ACCELEROMETER);
                                }
                                else
                                {
                                        host.setRequestType(MPU6050.DataReqType.GYROSCOPE);
                                }
                        }
                };

                this.setMinimumSize(new Dimension(300, 60));
                this.setMaximumSize(new Dimension(300, 60));
                this.setPreferredSize(new Dimension(300, 60));
                setLayout(new MigLayout("", "[grow]", "[][]"));

                rdbtnAccelerometer = new JRadioButton("Accelerometer");
                rdbtnAccelerometer.addActionListener(buttonClicked);
                add(rdbtnAccelerometer, "cell 0 0");
                rdbtnAccelerometer.setSelected(true);

                rdbtnGyroscope = new JRadioButton("Gyroscope");
                rdbtnGyroscope.addActionListener(buttonClicked);
                add(rdbtnGyroscope, "cell 0 1");
                rdbtnGyroscope.setSelected(false);

                bg.add(rdbtnAccelerometer);
                bg.add(rdbtnGyroscope);
        }

        /* (non-Javadoc)
         * @see wsn.sensor.sideview.SensorSideView#setSensor(wsn.sensor.Sensor)
         */
        @Override
        public void setSensor(Sensor s) {
                if(s instanceof MPU6050)
                {
                        this.host = (MPU6050) s;
                }
        }

        /* (non-Javadoc)
         * @see wsn.sensor.sideview.SensorSideView#getSensor()
         */
        @Override
        public Sensor getSensor() {
                return this.host;
        }

        /* (non-Javadoc)
         * @see wsn.sensor.sideview.SensorSideView#getPanel()
         */
        @Override
        public JPanel getPanel() {
                return this;
        }

}
```

```java
package wsn.sensor.view;

import java.util.Observable;

import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JTable;

import wsn.sensor.Sensor;

/**
 * @deprecated
 * @author jshirley
 *
 */
public class MPU6050TableView implements SensorView
{
    private JPanel panel;
    private JTable table;
    private String[] colHeaders = {"time","value"};

    public MPU6050TableView(Sensor s)
    {
        panel = new JPanel();
        table = new JTable(null, colHeaders);

        JScrollPane spanel = new JScrollPane(table);
        table.setFillsViewportHeight(true);
        panel.add(spanel);
    }

    @Override
    public JPanel getPanel()
    {
        return panel;
    }

        @Override
        public void setSensor(Sensor s) {
                // TODO Auto-generated method stub
        }

        @Override
        public Sensor getSensor() {
                // TODO Auto-generated method stub
                return null;
        }

        @Override
        public void addData(Object o) {
                // TODO Auto-generated method stub

        }

        @Override
        public void update(Observable o, Object arg) {
                // TODO Auto-generated method stub

        }

}
```

```java
package wsn.network;

import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.nio.ByteBuffer;
import java.nio.ShortBuffer;
import java.util.Arrays;
import java.util.Observable;
import java.util.Timer;
import java.util.TimerTask;
import java.util.logging.Level;
import java.util.logging.Logger;

import javax.swing.SwingWorker;

import wsn.Globals;

/**
 * The NetworkListener manages the connection to the Poly_Sense Server. It is
 * also used for requesting and sending data into the sensor network.
 *
 * @author jshirley
 *
 */
public class NetworkListener extends Observable {

        /**
         * Represents the number of times to try to read the Network for a response
         * packet.
         */
        private static final int EXPECT_PACKET_TRYS = 50;

        /**
         * The size in bytes of the packet used for communication between the
         * frontend and the Poly_Sense Server.
         */
        public static final int PACKET_SIZE = 18;
        /**
         * The size in bytes of the data in the packet.
         */
        public static final int DATA_SIZE = 15;

        /**
         * The Network being actively monitored.
         */
        private Network net;
        /**
         * The single instance of this Class.
         */
        private static NetworkListener instance = null;

        /**
         * The InputStream used for reading data from the active Network's socket.
         */
        private InputStream netReader;
        /**
         * The OutputStream used for writing data to the active Network's socket.
         */
        private OutputStream netWriter;

        /**
         * Represents whether the connection to the Poly_Sense Server is active.
         */
        private boolean isActive = false;

        /**
         * Simple timer for regularly checking the state of the network - the timer
         * that dispatches ping requests.
         */
        private Timer refresh;

        /**
         * Byte buffer that represents one packet, only one declared since in the
         * systems currently implementation the data requests occur sequentially -
         * only one packet at a time will be recieved and processed.
         */
        private byte[] packet = new byte[PACKET_SIZE];

        /**
         * @author jshirley
         *
         */
        public enum Events {
                CONNECTION_VALID, CONNECTION_INVALID
        }

        /**
         * Creating a new NetworkListener results in ping requests occurring every
         * 10s if data requests are not active.
         */
        private NetworkListener() {
                refresh = new Timer();
                // send a ping packet every 10s
                refresh.schedule(new TimerTask() {

                        @Override
                        public void run() {
                                if (net != null && Globals.getInstance().getActiveRun() != null) {
                                        if (!Globals.getInstance().getActiveRun().isRunning()) {
                                                sendPacket(Network.flags.PING.getValue(), (short)
0, "");
                                        }
                                }
                        }
                }, 0, 10000);
        }

        /**
         * @return The single instance of this Class.
         */
        public static NetworkListener getInstance() {
                if (instance == null) {
                        instance = new NetworkListener();
                }
                return instance;
        }

        /**
         * @return The Network this NetworkListener is monitoring.
         */
        public Network getNetwork() {
                return net;
        }

        /**
         * setNetwork changes the network to monitor to the Network net parameter.
         * Reopens the netReader and netWriter fields to access the Socket specified
         * in the net parameter. Also checks whether the Network is active at this
         * point.
         *
         * @param net
         *            Represents the Network for this NetworkListener to monitor.
         */
        public void setNetwork(Network net) {
                this.net = net;

                try {
                        netReader = net.getConnection().getInputStream();
                } catch (IOException e) {
                        e.printStackTrace();
                }
                try {
                        netWriter = net.getConnection().getOutputStream();
                } catch (IOException e) {
                        e.printStackTrace();
                }

                if (net.getConnection().isConnected()) {
                        fireEvent(Events.CONNECTION_VALID);
                        isActive = true;
                } else {
                        fireEvent(Events.CONNECTION_INVALID);
                        isActive = false;
                }
        }

        /**
         * @pre The caller has a lock on the Network.
         *
         *      sendPacket sends a packet to the short nodeId parameter with a flag
         *      based on the flag parameter and data based on the data parameter. In
         *      Poly_Sense's current implementation it is required for the user to
         *      obtain a lock on the Network before sending a packet and only
         *      releasing it once they have either received the packet or the
         *      request has timed out.
         *
         * @param flag
         *            Represents the flag for the packet to contain.
         * @param nodeId
         *            Represents the address for the packet to contain.
         * @param data
         *            Represents the data for the packet to contain.
         */
        public void sendPacket(final char flag, final short nodeId,
                        final String data) {

                SwingWorker<Boolean, Object> sw = new SwingWorker<Boolean, Object>() {

                        @Override
                        protected synchronized Boolean doInBackground() throws Exception {
                                char[] str = data.toCharArray();
                                if (str.length > DATA_SIZE) {
                                        System.err.println("Data element too long!");
                                        return false;
                                }

                                Arrays.fill(packet, (byte) 0);

                                packet[0] = (byte) flag;

                                // network/java order to host order
                                packet[1] = (byte) (nodeId & 0xff);
                                packet[2] = (byte) ((nodeId >> 8) & 0xff);

                                for (int i = 0; i < str.length; i++) {
                                        packet[i + 3] = (byte) str[i];
                                }

                                try {
                                        netWriter.write(packet, 0, PACKET_SIZE);
                                        netWriter.flush();
                                } catch (IOException ex) {
                                        System.err.println("disconnected from base station");
                                        isActive = false;
                                        net = null;
                                        fireEvent(Events.CONNECTION_INVALID);
                                }
                                fireEvent(Events.CONNECTION_VALID);
                                isActive = true;
                                return true;
                        }

                };
                sw.execute();
        }

        /**
         * @pre The caller has a lock on the Network.
         * 
         *      expectPacket should be called after the caller has called sendPacket
         *      and is expecting a response from the network. In Poly_Sense's
         *      current implementation it is required for the user to obtain a lock
         *      on the Network before sending a packet and only releasing it once
         *      they have either received the packet or the request has timed out.
         * 
         * @param flag
         *            Represents the type of packet that is expected.
         * @return True if the packet received contained the specified flag. False
         *         otherwise.
         */
        public boolean expectPacket(final char flag) {
                for (int i = 0; i < EXPECT_PACKET_TRYS; i++) {
                        int read = 0;
                        try {
                                if (net != null) {
                                        int bytesReady = net.getConnection().getInputStream()
                                                        .available();
                                        if (bytesReady >= PACKET_SIZE) {
                                                try {
                                                        read = netReader.read(packet, 0,
PACKET_SIZE);
                                                } catch (IOException ex) {
                                                        fireEvent(Events.CONNECTION_INVALID);
                                                        isActive = false;
                                                        Logger.getLogger
(NetworkListener.class.getName())
                                                                        .log(Level.SEVERE, null,
ex);
                                                }
                                                if (read == PACKET_SIZE) {
                                                        if (packet[0] == flag) {
                                                                fireEvent(Events.CONNECTION_VALID);
                                                                isActive = true;
                                                                return true;
                                                        }
                                                        System.err
                                                                        .println("got a different
packet than expected: ");
                                                        for (int j = 0; j < PACKET_SIZE; j++) {
                                                                System.err.print(((char) packet[j])
+ " ");
                                                        }
                                                        System.err.println();
                                                        return false;
                                                } else if (read < PACKET_SIZE) {
                                                        System.err.println("got less than a packet:
"
                                                                        + read);
                                                }
                                        }
                                } else {
                                        fireEvent(Events.CONNECTION_INVALID);
                                        isActive = false;
                                }
                        } catch (IOException e) {
                                e.printStackTrace();
                        }

                        try {
```

```java
                                // wait 100ms
                                Thread.sleep(100);
                        } catch (InterruptedException e) {
                                e.printStackTrace();
                        }
                }
                return false;
        }

        /**
         * @return The latest packet received from the Poly_Sense Server.
         */
        public byte[] getPacket() {
                return packet;
        }

        /**
         * @return The flag from the latest packet received from the Poly_Sense
         *         Server.
         */
        public char getPacketFlag() {
                return (char) packet[0];
        }

        /**
         * @return The nodeId (in network order/java order) from the latest packet
         *         received from the Poly_Sense Server.
         */
        public short getPacketNodeId() {
                byte[] bytes = { 0, 0 };
                bytes[0] = packet[1];
                bytes[1] = packet[2];

                // host order to network/java order
                ByteBuffer bb = ByteBuffer.wrap(Arrays.copyOfRange(bytes, 0, 2));
                ShortBuffer sb = bb.asShortBuffer();

                return sb.get();
        }

        /**
         * @return The data from the latest packet received from the Poly_Sense
         *         Server.
         */
        public byte[] getPacketData() {
                byte[] bytes = new byte[DATA_SIZE];

                for (int i = 0; i < DATA_SIZE; i++) {
                        bytes[i] = packet[i + 3];
                }
                return bytes;
        }

        /**
         * @return True is the Network the this NetworkListener is monitoring is
         *         active. False otherwise.
         */
        public boolean isActive() {
                return isActive;
        }

        /**
         * @param e
         *            The Event for NetworkListener to fire.
         */
        public void fireEvent(NetworkListener.Events e) {
                setChanged();
                notifyObservers(e);
        }
}
```

```java
package wsn.network;

import java.io.IOException;
import java.net.InetSocketAddress;
import java.net.Socket;

import javax.swing.JOptionPane;

/**
 * This class represents a connection to a remote host (for this project it is
 * always a Poly_Sense Server).
 *
 * @author jshirley
 *
 */
public class Network {
        /**
         * This is the TCP connection that the Network is using to communicate with
         * the Poly_Sense Server.
         */
        private Socket connection;

        /**
         * @author jshirley
         *
         */
        public enum flags {
                SER_DATA_REQ('1'), SENSOR_CONFIG_REQ('2'), SENSOR_DATA('3'), DEBUG('4'), PING(
                                '5');

                private final char val;

                flags(char val) {
                        this.val = val;
                }

                public char getValue() {
                        return val;
                }
        };

        /**
         * Creating a new Network will result in a Socket being opened for the ip
         * address addr on the int port port. If the Socket cannot be opened a
         * dialog appears.
         *
         * @param addr
         *            the ip address
         * @param port
         *            the port
         */
        public Network(String addr, int port) {
                connection = new Socket();
                try {
                        connection.connect(new InetSocketAddress(addr, port), 5000);
                } catch (IOException e) {
                        JOptionPane.showMessageDialog(null,
                                        "Connection failed for: " + addr + ":" + port
                                                        + "\nReason: " + e.getLocalizedMessage());
                        System.err.println("Connection failed for: " + addr + ":" + port);
                        connection = null;

                        e.printStackTrace();
                }
        }

        /**
         * @return The Socket connection to the Poly_Sense Server.
         */
        public Socket getConnection() {
                return connection;
        }
}
```

```java
package wsn.popups;

/**
 * A utility class for easily launching different types of popups.
 *
 * @author jshirley
 *
 */
public class Popups {

        /**
         * showPreferences launches a new PreferencesDialog
         */
        public static void showPreferences()
        {
                new PreferencesDialog().setVisible(true);
        }
        /**
         * @deprecated
         */
        public static void showHelp()
        {
                //TODO
        }
        /**
         * @deprecated
         */
        public static void showExit()
        {
                //TODO
        }
        /**
         * showExport launches a new ExportDialog
         */
        public static void showExport()
        {
                new ExportDialog().setVisible(true);
        }
        /**
         * showImport launches a new ImportDialog
         */
        public static void showImport()
        {
                new ImportDialog();
        }
}
```

```java
package wsn.popups;

import java.awt.FlowLayout;
import java.awt.Toolkit;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.prefs.BackingStoreException;

import javax.swing.Icon;
import javax.swing.JButton;
import javax.swing.JDialog;
import javax.swing.JFormattedTextField;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JTabbedPane;
import javax.swing.SwingConstants;

import net.miginfocom.swing.MigLayout;
import wsn.Globals;

/**
 * A popup that displays various Preferences a user can modify. After the user
 * closes the dialog, their choices will be saved.
 * 
 * @author jshirley
 * 
 */
public class PreferencesDialog extends JDialog {

        /**
         * The TextField that contains the current default network ip address and
         * port for the Poly_Sense Server
         */
        private JFormattedTextField formattedDefaultNetwork;

        /**
         * Creating a new PreferencesDialog lays out its components and initializes
         * them.
         */
        public PreferencesDialog() {
                setResizable(false);
                setTitle("Preferences");
                setIconImage(Toolkit.getDefaultToolkit().getImage(
                                PreferencesDialog.class.getResource("/wireless.png")));
                setBounds(100, 100, 450, 300);
                getContentPane().setLayout(new MigLayout("", "[grow]", "[grow][30]"));

                JTabbedPane tabbedPreferences = new JTabbedPane(SwingConstants.TOP);
                getContentPane().add(tabbedPreferences, "cell 0 0,grow");

                JPanel panelGeneral = new JPanel();
                tabbedPreferences.addTab("General", (Icon) null, panelGeneral, null);
                panelGeneral.setLayout(new MigLayout("", "[][grow]", "[]"));

                JLabel lblDefaultNetwork = new JLabel("Default Network: ");
                panelGeneral.add(lblDefaultNetwork, "cell 0 0,alignx trailing");

                formattedDefaultNetwork = new JFormattedTextField();
                formattedDefaultNetwork
                                .setText(Globals.getInstance().getUserPreferences()
                                                .get("default_network", "localhost:55555"));
                panelGeneral.add(formattedDefaultNetwork, "cell 1 0,growx");

                JPanel panel = new JPanel();
                getContentPane().add(panel, "cell 0 1,growx,aligny bottom");
                panel.setLayout(new FlowLayout(FlowLayout.RIGHT));

                JButton btnOk = new JButton("OK");
                btnOk.addActionListener(new ActionListener() {
                        @Override
```

```
            public void actionPerformed(ActionEvent arg0) {
                Globals.getInstance()
                        .getUserPreferences()
                        .put("default_network",
                                formattedDefaultNetwork.getText());
                try {
                    Globals.getInstance().getUserPreferences().sync();
                } catch (BackingStoreException e) {
                    e.printStackTrace();
                }
            }
        });
        btnOk.setActionCommand("OK");
        panel.add(btnOk);

        JButton btnCancel = new JButton("Cancel");
        btnCancel.setActionCommand("Cancel");
        panel.add(btnCancel);

    }

}
```

```java
package wsn.panels;

import java.awt.Dimension;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.Timer;
import java.util.TimerTask;

import javax.swing.DefaultComboBoxModel;
import javax.swing.ImageIcon;
import javax.swing.JButton;
import javax.swing.JComboBox;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JTextField;

import net.miginfocom.swing.MigLayout;
import wsn.Globals;
import wsn.sensor.Sensor;
import wsn.sensor.Sensor.viewType;

/**
 * This panel presents a single Nodes common information. This includes: node
 * address, current sensor type, current active view, and the status string (if
 * implemented).
 *
 * @author jshirley
 *
 */
public class PrefixActiveNodePanel extends JPanel {
        /**
         * This field displays the node address for this panel.
         */
        private JTextField fieldNode;
        /**
         * This field displays the node/sensor pairing's status.
         */
        private JTextField fieldStatus;
        /**
         * This timer requests updates from the sensor parameter's status string.
         */
        private Timer refresh;

        /**
         * This is the Sensor model that is being represented by this panel.
         */
        private Sensor sensor;
        /**
         * This is the node address that is being represented by this panel.
         */
        private Short node_id;
        /**
         * This combobox contains all of the Sensor Classes that were loaded at
         * runtime. It can be used to reconfigure the node.
         */
        private JComboBox comboType;
        /**
         * This combobox contains all of the viewType's for the current
         * configuration. It can be used to change what the current view is.
         */
        private JComboBox comboView;

        /**
         * Creating a new PrefixActiveNodePanel causing it to layout its components
         * and start a timer which updates the panels fieldStatus member every 5
         * seconds to match the Sensor whose status it is representing.
         *
         * @param sen
         *            The sensor that this panel is representing.
         * @param id
         *            The node address this panel is representing.
         */
        public PrefixActiveNodePanel(Sensor sen, Short id) {
                this.sensor = sen;
                this.node_id = id;

                refresh = new Timer();
                // get status string every 5s
                refresh.schedule(new TimerTask() {

                        @Override
                        public void run() {
                                if (fieldStatus != null && sensor != null) {
                                        fieldStatus.setText(sensor.getStatusString());
                                }
                        }
                }, 0, 5000);

                setLayout(new MigLayout("", "[][grow][]", "[][][][]"));
                this.setMinimumSize(new Dimension(300, 120));
                this.setMaximumSize(new Dimension(300, 120));
                this.setPreferredSize(new Dimension(300, 120));

                JLabel lblNode = new JLabel("Node:");
                add(lblNode, "cell 0 0,alignx trailing");

                fieldNode = new JTextField();
                fieldNode.setEnabled(false);
                fieldNode.setText(id.toString());
                add(fieldNode, "cell 1 0,growx");
                fieldNode.setColumns(10);

                JButton btnRemoveNode = new JButton("");
                btnRemoveNode.addActionListener(new ActionListener() {
                        @Override
                        public void actionPerformed(ActionEvent arg0) {
                                Globals.getInstance().removeNode(node_id);
                        }
                });

                ImageIcon btnIcon = new ImageIcon(
                                PrefixActiveNodePanel.class.getResource("/remove_x128.png"));
                btnRemoveNode.setIcon(new ImageIcon(btnIcon.getImage()
                                .getScaledInstance(10, 10, java.awt.Image.SCALE_SMOOTH)));
                add(btnRemoveNode, "cell 2 0,wmax 30");

                JLabel lblType = new JLabel("Type:");
                add(lblType, "cell 0 1,alignx trailing,hmax 30");

                comboType = new JComboBox(new DefaultComboBoxModel(Globals
                                .getInstance().getSensorNames().toArray()));
                comboType.setSelectedItem(sensor.getSensorName());
                add(comboType, "cell 1 1,growx,hmax 30");

                JButton btnCommitType = new JButton("");
                btnCommitType.addActionListener(new ActionListener() {
                        @Override
                        public void actionPerformed(ActionEvent e) {
                                Globals.getInstance().removeNode(node_id);
                                Globals.getInstance().addNode(node_id,
                                                comboType.getSelectedItem().toString());
                        }
                });
                btnIcon = new ImageIcon(
                                PrefixActiveNodePanel.class.getResource("/add_x128.png"));
                btnCommitType.setIcon(new ImageIcon(btnIcon.getImage()
                                .getScaledInstance(10, 10, java.awt.Image.SCALE_SMOOTH)));
                add(btnCommitType, "cell 2 1,wmax 30");

                JLabel lblView = new JLabel("View:");
                add(lblView, "cell 0 2,alignx trailing,hmax 30");

                comboView = new JComboBox(new DefaultComboBoxModel(sensor
                                .getViewTypes().toArray()));
                add(comboView, "cell 1 2,growx,hmax 30");

                JButton btnSetActiveView = new JButton("");
                if (Globals.getInstance().getActiveRun() == null) {
                        btnSetActiveView.setEnabled(false);
                }
                btnSetActiveView.addActionListener(new ActionListener() {
                        @Override
                        public void actionPerformed(ActionEvent e) {
                                sensor.setActiveView((viewType) comboView.getSelectedItem());
                                Globals.getInstance().setActiveSensor(sensor);
                        }
                });
                btnIcon = new ImageIcon(
                                PrefixActiveNodePanel.class.getResource("/rightarrow_x128.png"));
                btnSetActiveView.setIcon(new ImageIcon(btnIcon.getImage()
                                .getScaledInstance(10, 10, java.awt.Image.SCALE_SMOOTH)));
                add(btnSetActiveView, "cell 2 2,wmax 30");

                JLabel lblStatus = new JLabel("Status:");
                add(lblStatus, "cell 0 3,alignx trailing,hmax 30");

                fieldStatus = new JTextField();
                fieldStatus.setEnabled(false);
                fieldStatus.setText(sensor.getStatusString());
                add(fieldStatus, "cell 1 3,growx,hmax 30");
                fieldStatus.setColumns(10);

        }

}
```

```java
package wsn.sensor.types;

/**
 * A simple wrapper class that creates and object from a float which represents
 * a pressure value.
 *
 * @author jshirley
 *
 */
public class Pressure {
        private float pressure;

        public Pressure(float p) {
                this.pressure = p;
        }

        public float getPressure() {
                return pressure;
        }

        public void setPressure(float pressure) {
                this.pressure = pressure;
        }
}
```

```java
package wsn.run;

import java.util.Observable;
import java.util.concurrent.ScheduledThreadPoolExecutor;
import java.util.concurrent.TimeUnit;

import wsn.Globals;

/**
 * Run manages the time associated with a Sensor data request session. It
 * determines when data requests start and when they end.
 *
 * @author jshirley
 *
 */
public class Run extends Observable {
        /**
         * The thread which updates this Run's current time.
         */
        private ScheduledThreadPoolExecutor timeThread;
        /**
         * Represents whether this run is increasing in time.
         */
        private boolean isRunning = false;
        /**
         * Represents the total number of objects of the Run class.
         */
        private static int numRunsCreated = 0;

        /**
         * Represents the amount of time this run has been "running" for in tenths
         * of seconds. This excludes time while paused.
         */
        private long time = 0;

        /**
         * Creating a new Run increases the numRunsCreated field.
         */
        public Run() {
                numRunsCreated++;
        }

        /**
         * start creates a new Thread (ScheduledThreadPoolExecutor) to update the
         * running time and notify any observers of this Run. This is also what
         * starts requesting data from nodes.
         */
        public void start() {
                // thread to update time, you have to make a new one everytime? Would be
                // nice if you could reuse object...
                timeThread = new ScheduledThreadPoolExecutor(1);
                timeThread.scheduleAtFixedRate(new Runnable() {

                        @Override
                        public void run() {
                                time++;
                                setChanged();
                                notifyObservers();
                        }
                }, 0, 100, TimeUnit.MILLISECONDS);
                Globals.getInstance().startNodeRequests();

                isRunning = true;
        }

        /**
         * pause shuts down the timeThread and stops requesting data from nodes.
         */
        public void pause() {
                timeThread.shutdownNow();
                isRunning = false;

                Globals.getInstance().stopNodeRequests();
        }

        /**
         * stop behaves the same as pause expect it resets the running time of this
         * Run to 0.
         */
        public void stop() {
                time = 0;
                pause();
        }

        /**
         * @return True if the timeThread is running. False otherwise.
         */
        public boolean isRunning() {
                return isRunning;
        }

        /**
         * @return A string representation of the current run time of this Run. It
         *         is formatted as [SECONDS].[MILLISECONDS]s.
         */
        public String getFormattedTime() {
                return (time / 10) + "." + (time % 10) + "s";
        }

        /**
         * @return A string representation of this Run's name. Currently this only
         *         returns a name based on the total number of Run objects created.
         */
        public String getRunName() {
                return "Run: " + numRunsCreated;
        }

        /**
         * @return The current running time of this Run.
         */
        public long getTime() {
                return time;
        }
}
```

```java
package wsn.panels;

import java.util.Observable;
import java.util.Observer;

import javax.swing.JPanel;
import javax.swing.JScrollPane;

import wsn.Globals;
import wsn.run.Run;

/**
 * A single RunViewPanel houses the activeSensor's activeView.
 *
 * @author jshirley
 *
 */
public class RunViewPanel extends JScrollPane implements Observer {
        /**
         * This panel represents a view for a Run.
         */
        private JPanel panel = null;
        /**
         * This is the Run model that this panel represents.
         */
        private Run run = null;

        /**
         * Creating a new RunViewPanel sets its Run to model as the parameter Run r.
         * It also begins watching Globals for Events.
         *
         * @param r
         */
        public RunViewPanel(Run r) {
                this.run = r;
                this.setName(run.getRunName());

                panel = new JPanel();
                setViewportView(panel);

                Globals.getInstance().addObserver(this);
        }

        /*
         * update is Called when Globals fires and Event. If the active Sensor
         * changed, it changes its view to that that Sensor's activeView.
         *
         * @see java.util.Observer#update(java.util.Observable, java.lang.Object)
         */
        @Override
        public void update(Observable o, Object arg) {
                if (o instanceof Globals) {
                        if (arg == Globals.Events.ACTIVE_SENSOR_CHANGED) {
                                if (Globals.getInstance().getActiveRun() == run) {
                                        panel = Globals.getInstance().getActiveSensor()
                                                        .getActiveView();
                                        panel.revalidate();
                                        setViewportView(panel);
                                        revalidate();
                                        repaint();
                                }
                        }
                }

        }
}
```

```java
package wsn.sensor;

import java.util.ArrayList;
import java.util.Observable;

import javax.swing.JPanel;

import wsn.network.NetworkListener;

/**
 * Sensor defines the minimum methods required for integrating a new sensor into
 * the Poly_Sense Monitor.
 *
 * @author jshirley
 *
 */
public abstract class Sensor extends Observable {
        /**
         * @return A new instance of this Sensor.
         */
        public abstract Sensor getNewInstance();

        /**
         * @return The String representation of this Sensor.
         */
        public abstract String getSensorName();

        /**
         * @return The String representation of this Sensor's status to be displayed
         *         in the side panel.
         */
        public abstract String getStatusString();

        /**
         * @return The viewTypes allowed for this Sensor.
         */
        public abstract ArrayList<viewType> getViewTypes();

        /**
         * Sets the viewType that this Sensor will be displayed to viewType
         * parameter v.
         *
         * @param v
         *            Represents the viewType that this Sensor will display if it is
         *            the actively viewed Sensor
         */
        public abstract void setActiveView(viewType v);

        /**
         * @return The activeView that this Sensor will display.
         */
        public abstract JPanel getActiveView();

        /**
         * @return The sidePanel that this Sensor will display.
         */
        public abstract JPanel getSidePanel();

        /**
         * requestData should use the sendPacket method of the NetworkListener
         * paramater n to request data from the node specified at short parameter
         * addr.
         *
         * @param n
         *            The NetworkListener object to use to request data.
         * @param addr
         *            The address of the node that this Sensor is on.
         */
        public abstract void requestData(NetworkListener n, short addr);

        /**
         * @return A String representation of all of the data that this Sensor has
         *         gathered.
         */
        public abstract String getBackingData();

        /**
         * setData should initialize this Sensor to the state specified by the
         * String parameter data.
         * 
         * @param data
         *            Represents a state this Sensor was in.
         */
        public abstract void setData(String data);

        /**
         * addData should interpret the byte array parameter bytes as data for this
         * Sensor and store it.
         * 
         * @param bytes
         *            Contains the data element of a packet.
         */
        public abstract void addData(byte[] bytes);

        /**
         * @author jshirley
         * 
         */
        public enum viewType {
                GRAPH, TABLE, SPECIFIC, OTHER;
        }

        /**
         * @author jshirley
         * 
         */
        public enum sensorStates {
                UNITIALIZED, CONNECTED, DISCONNECTED;
        }
}
```

```java
package wsn.sensor.sideview;

import javax.swing.JPanel;

import wsn.sensor.Sensor;

/**
 * @author jshirley
 *
 */
public interface SensorSideView{
	/**
	 * @param s The Sensor that this SensorSideView uses as a model.
	 */
	public void setSensor(Sensor s);
	/**
	 * @return The Sensor that this SensorSideView uses as a model.
	 */
	public Sensor getSensor();
	/**
	 * @return The panel that this SensorSideView represents. This is the panel that will be placed in the side panel.
	 */
	public JPanel getPanel();
}
```

```java
package wsn.sensor.view;

import java.util.Observer;

import javax.swing.JPanel;

import wsn.sensor.Sensor;

/**
 * @author jshirley
 *
 *         This interface defines the basic methods needed to display sensor
 *         data in the Poly_Sense monitor.
 */
public interface SensorView extends Observer {
        /**
         * @return The panel that this SensorView represents.
         */
        public JPanel getPanel();

        /**
         * @param s
         *            The Sensor that this panel uses as a model.
         */
        public void setSensor(Sensor s);

        /**
         * @return The Sensor that this panel uses as a model.
         */
        public Sensor getSensor();

        /**
         * @param o
         *            Data to add to this SensorView.
         */
        public void addData(Object o);
}
```

```java
package wsn.util;

/**
 * A simple wrapper class that pairs 7 objects.
 *
 * @author jshirley
 *
 * @param <T>
 * @param <U>
 * @param <V>
 * @param <W>
 * @param <X>
 * @param <Y>
 * @param <Z>
 */
public class Septuple<T, U, V, W, X, Y, Z> implements Comparable<Object> {

        public final T t;
        public final U u;
        public final V v;
        public final W w;
        public final X x;
        public final Y y;
        public final Z z;

        public Septuple(T t, U u, V v, W w, X x, Y y, Z z) {
                this.t = t;
                this.u = u;
                this.v = v;
                this.w = w;
                this.x = x;
                this.y = y;
                this.z = z;
        }

        @Override
        public int compareTo(Object o) {
                return toString().compareTo(o.toString());
        }

}
```

```java
package wsn.sensor.types;

/**
 * A simple wrapper class that creates and object from a float which represents
 * a temperature value.
 *
 * @author jshirley
 */
public class Temperature {
        private float temperature;

        public Temperature(float t) {
                this.temperature = t;
        }

        public float getTemperature() {
                return temperature;
        }

        public void setTemperature(float temperature) {
                this.temperature = temperature;
        }
}
```

```java
package wsn.sensor.types;

/**
 * A simple wrapper class that creates and object from a long which represents a
 * time value.
 *
 * @author jshirley
 */
public class Time {
        long time;

        public Time(long t) {
                this.time = t;
        }

        public long getTime() {
                return time;
        }

        public void setTime(long time) {
                this.time = time;
        }

}
```

```java
package wsn.util;

/**
 *
 * A simple wrapper class that pairs 3 objects.
 *
 * @author jshirley
 *
 * @param <X>
 * @param <Y>
 * @param <Z>
 */
public class Triple<X, Y, Z> implements Comparable<Object> {
        public final X x;
        public final Y y;
        public final Z z;

        public Triple(X x, Y y, Z z) {
                this.x = x;
                this.y = y;
                this.z = z;
        }

        @Override
        public int compareTo(Object o) {
                return toString().compareTo(o.toString());
        }
}
```

```java
package wsn.util;

/**
 * A simple wrapper class that pairs 2 objects.
 *
 * @author jshirley
 *
 * @param <X>
 * @param <Y>
 */
public class Tuple<X, Y> implements Comparable<Object>{
        public final X x;
        public final Y y;

        public Tuple(X x, Y y) {
                this.x = x;
                this.y = y;
        }

        @Override
        public int compareTo(Object o) {
                return toString().compareTo(o.toString());
        }
}
```

```java
package wsn.sensor.types;

/**
 * A simple wrapper class that creates and object from a float which represents
 * a velocity value.
 *
 * @author jshirley
 */
public class Velocity {
        float velocity;

        public float getVelocity() {
                return velocity;
        }

        public void setVelocity(float velocity) {
                this.velocity = velocity;
        }

        public Velocity(float vel) {
                this.velocity = vel;
        }
}
```

```java
package wsn;

import java.awt.Color;
import java.awt.Component;
import java.awt.Dimension;
import java.awt.EventQueue;
import java.awt.Toolkit;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.Observable;
import java.util.Observer;

import javax.swing.Box;
import javax.swing.DefaultComboBoxModel;
import javax.swing.JButton;
import javax.swing.JCheckBoxMenuItem;
import javax.swing.JComboBox;
import javax.swing.JFormattedTextField;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JMenu;
import javax.swing.JMenuBar;
import javax.swing.JMenuItem;
import javax.swing.JPanel;
import javax.swing.JSeparator;
import javax.swing.JTabbedPane;
import javax.swing.JTextField;
import javax.swing.JToolBar;
import javax.swing.SwingConstants;
import javax.swing.border.BevelBorder;
import javax.swing.event.DocumentEvent;
import javax.swing.event.DocumentListener;

import net.miginfocom.swing.MigLayout;
import wsn.network.Network;
import wsn.network.NetworkListener;
import wsn.panels.ActiveNodesPanel;
import wsn.panels.RunViewPanel;
import wsn.popups.Popups;
import wsn.run.Run;
import wsn.sensor.Sensor;
import wsn.util.Tuple;

/**
 * The main frame of the Poly_Sense Monitor.
 *
 * @author jshirley
 *
 */
public class WSN implements Observer {

        /**
         * Root JFrame.
         */
        private JFrame wsn;
        /**
         * Network ip and port field.
         */
        private JFormattedTextField formattedNetwork;
        /**
         * Network status.
         */
        private JButton btnStatus;

        /**
         * Single instance of this class.
         */
        private static WSN instance = null;
        /**
         * Button for adding a node address and sensor combination.
         */
        private JButton btnAddNode;
        /**
         * Node address field.
         */
        private JTextField fldId;
        /**
         * TabbedPane for containing each run and its view.
         */
        private JTabbedPane tabbedRunView;
        /**
         * Button for adding a new Run.
         */
        private JButton btnNewRun;
        /**
         * Button for starting a Run.
         */
        private JButton btnStart;
        /**
         * Button for pausing a Run.
         */
        private JButton btnPause;
        /**
         * Button for stopping a Run.
         */
        private JButton btnStop;
        /**
         * Button for deleting a Run.
         */
        private JButton btnDelete;
        /**
         * Label which shows the current time of the current Run.
         */
        private JLabel lblCurTime;

        /**
         * @param args
         *            Represents the command line arguments - ignored.
         */
        public static void main(String[] args) {
                EventQueue.invokeLater(new Runnable() {
                        @Override
                        public void run() {
                                try {
                                        WSN window = WSN.getInstance();
                                        window.wsn.setVisible(true);
                                } catch (Exception e) {
                                        e.printStackTrace();
                                }
                        }
                });
        }

        /**
         * Creating a WSN initializes all of its components and starts listening for
         * Globals and NetworkListener Events.
         */
        private WSN() {
                initialize();

                Globals.getInstance().addObserver(this);
                NetworkListener.getInstance().addObserver(this);
        }

        /**
         * @return The single instance of WSN.
         */
        public static WSN getInstance() {
                if (instance == null) {
                        instance = new WSN();
                }
                return instance;
        }

        /**
         * Initialize the contents of the frame.
         */
        private void initialize() {
                wsn = new JFrame();
                wsn.setTitle("Poly_Sense Monitor");
                wsn.setIconImage(Toolkit.getDefaultToolkit().getImage(
                                WSN.class.getResource("/wireless.png")));
                wsn.setSize(new Dimension(800, 600));
                wsn.setMinimumSize(new Dimension(1024, 600));

                wsn.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
                wsn.getContentPane().setLayout(
                                new MigLayout("", "[300,grow][grow]", "[][grow][40]"));

                JToolBar toolBar = new JToolBar();
                toolBar.setFloatable(false);
                wsn.getContentPane().add(toolBar, "cell 0 0 2 1,grow");

                JLabel lblBaseStationAddress = new JLabel("Base Station Address: ");
                toolBar.add(lblBaseStationAddress);

                formattedNetwork = new JFormattedTextField();

                formattedNetwork.setText(Globals.getInstance().getUserPreferences()
                                .get("default_network", "localhost:55555"));
                toolBar.add(formattedNetwork);

                JButton btnConnect = new JButton("Connect");
                btnConnect.addActionListener(new ActionListener() {
                        @Override
                        public void actionPerformed(ActionEvent arg0) {
                                String s = formattedNetwork.getText();
                                int portIndex = s.lastIndexOf(':');
                                String addr = s.substring(0, portIndex);
                                String port = s.substring(portIndex + 1, s.length());

                                Network n = new Network(addr, Integer.parseInt(port));

                                if (n.getConnection() != null) {
                                        NetworkListener.getInstance().setNetwork(n);
                                }
                        }
                });
                toolBar.add(btnConnect);

                Component horizontalStrut = Box.createHorizontalStrut(20);
                toolBar.add(horizontalStrut);

                JLabel lblBaseStationStatus = new JLabel("Base Station Status: ");
                toolBar.add(lblBaseStationStatus);

                btnStatus = new JButton("Disconnected");
                btnStatus.setBackground(Color.RED);
                btnStatus.addActionListener(new ActionListener() {
                        @Override
                        public void actionPerformed(ActionEvent arg0) {
                        }
                });
                btnStatus.setEnabled(false);
                toolBar.add(btnStatus);

                ActiveNodesPanel panelActiveNodes = new ActiveNodesPanel();
                wsn.getContentPane().add(panelActiveNodes, "cell 0 1,grow");

                tabbedRunView = new JTabbedPane(SwingConstants.TOP);
                tabbedRunView.setBorder(new BevelBorder(BevelBorder.LOWERED, null,
                                null, null, null));
                wsn.getContentPane().add(tabbedRunView, "cell 1 1,grow");

                JPanel panelAddNode = new JPanel();
                panelAddNode.setBorder(new BevelBorder(BevelBorder.LOWERED, null, null,
                                null, null));
                wsn.getContentPane().add(panelAddNode, "cell 0 2,grow");
                panelAddNode.setLayout(new MigLayout("", "[][][][][][]", "[]"));

                JLabel lblId = new JLabel("id:");
                panelAddNode.add(lblId, "cell 0 0,width 20::"); // size > 20

                fldId = new JTextField();
                fldId.getDocument().addDocumentListener(new DocumentListener() {

                        @Override
                        public void removeUpdate(DocumentEvent arg0) {
                                checkAddNode();
                        }

                        @Override
                        public void insertUpdate(DocumentEvent arg0) {
                                checkAddNode();
                        }

                        @Override
                        public void changedUpdate(DocumentEvent arg0) {
                                checkAddNode();
                        }
                });
                panelAddNode.add(fldId, "cell 1 0,width 45:10000:10000"); // size =

                JLabel lblType = new JLabel("type:");
                panelAddNode.add(lblType, "cell 2 0,width 40::"); // size =

                final JComboBox comboBox = new JComboBox();
                comboBox.setModel(new DefaultComboBoxModel(Globals.getInstance()
                                .getSensorNames().toArray()));
                panelAddNode.add(comboBox, "cell 3 0,width 90:10000:10000");

                btnAddNode = new JButton("Add Node");
                btnAddNode.setEnabled(false);
                btnAddNode.addActionListener(new ActionListener() {
                        @Override
                        public void actionPerformed(ActionEvent arg0) {
                                Globals.getInstance().addNode(
                                                Short.parseShort(fldId.getText()),
                                                (String) comboBox.getSelectedItem());
                                fldId.setText("");
                                checkAddNode();
                        }
                });
                panelAddNode.add(btnAddNode, "cell 4 0,width 105:10000:10000"); // size

=

                JPanel panelRunControls = new JPanel();
                panelRunControls.setBorder(new BevelBorder(BevelBorder.LOWERED, null,
                                null, null, null));
                wsn.getContentPane().add(panelRunControls, "cell 1 2,grow");
                panelRunControls.setLayout(new MigLayout("", "[][][][][][][][][]", "[]"));

                btnNewRun = new JButton("New Run");
                btnNewRun.addActionListener(new ActionListener() {
                        @Override
                        public void actionPerformed(ActionEvent arg0) {
                                Run run = new Run();
                                run.addObserver(WSN.getInstance());
                                Globals.getInstance().addRun(run);
```

```java
                                tabbedRunView.add(new RunViewPanel(run));

                                checkTabbedRuns();
                        }
                });
                panelRunControls.add(btnNewRun, "cell 0 0,width 100:10000:10000");

                btnStart = new JButton("Start");
                btnStart.addActionListener(new ActionListener() {
                        @Override
                        public void actionPerformed(ActionEvent arg0) {
                                Globals.getInstance().getActiveRun().start();

                                checkTabbedRuns();
                        }
                });
                btnStart.setEnabled(false);
                panelRunControls.add(btnStart, "cell 1 0,width 50:10000:10000");

                btnPause = new JButton("Pause");
                btnPause.addActionListener(new ActionListener() {
                        @Override
                        public void actionPerformed(ActionEvent e) {
                                Globals.getInstance().getActiveRun().pause();

                                checkTabbedRuns();
                        }
                });
                btnPause.setEnabled(false);
                panelRunControls.add(btnPause, "cell 2 0,width 50:10000:10000");

                btnStop = new JButton("Stop");
                btnStop.addActionListener(new ActionListener() {
                        @Override
                        public void actionPerformed(ActionEvent e) {
                                Globals.getInstance().getActiveRun().stop();

                                checkTabbedRuns();
                        }
                });
                btnStop.setEnabled(false);
                panelRunControls.add(btnStop, "cell 3 0,width 50:10000:10000");

                btnDelete = new JButton("Delete");
                btnDelete.addActionListener(new ActionListener() {
                        @Override
                        public void actionPerformed(ActionEvent e) {
                                Globals.getInstance().getActiveRun().stop();
                                Globals.getInstance().removeActiveRun();

                                checkTabbedRuns();
                        }
                });
                btnDelete.setEnabled(false);
                panelRunControls.add(btnDelete, "cell 4 0,width 50:10000:10000");

                JLabel lblRunTime = new JLabel("Run Time:");
                panelRunControls.add(lblRunTime, "cell 6 0,width 70::");

                lblCurTime = new JLabel("000:00.0");
                panelRunControls.add(lblCurTime, "cell 7 0,width 70::");

                JMenuBar menuBar = new JMenuBar();
                wsn.setJMenuBar(menuBar);

                JMenu mnFile = new JMenu("File");
                menuBar.add(mnFile);

                JMenuItem mntmExport = new JMenuItem("Export");
                mntmExport.addActionListener(new ActionListener() {
                        @Override
                        public void actionPerformed(ActionEvent arg0) {
                                Popups.showExport();
                        }
                });
                mnFile.add(mntmExport);

                JMenuItem mntmImport = new JMenuItem("Import");
                mntmImport.addActionListener(new ActionListener() {
                        @Override
                        public void actionPerformed(ActionEvent arg0) {
                                Popups.showImport();
                        }
                });
                mnFile.add(mntmImport);

                JSeparator separator = new JSeparator();
                mnFile.add(separator);

                JMenuItem mntmPreferences = new JMenuItem("Preferences");
                mntmPreferences.addActionListener(new ActionListener() {
                        @Override
                        public void actionPerformed(ActionEvent arg0) {
                                Popups.showPreferences();
                        }
                });
                mnFile.add(mntmPreferences);

                JSeparator separator_1 = new JSeparator();
                mnFile.add(separator_1);

                JMenuItem mntmQuit = new JMenuItem("Quit");
                mnFile.add(mntmQuit);

                JMenu mnEdit = new JMenu("Edit");
                menuBar.add(mnEdit);

                JCheckBoxMenuItem chckbxmntmEnableLocalServer = new JCheckBoxMenuItem(
                                "Enable Local Server");
                mnEdit.add(chckbxmntmEnableLocalServer);

                JCheckBoxMenuItem chckbxmntmShowConsole = new JCheckBoxMenuItem(
                                "Show Console");
                mnEdit.add(chckbxmntmShowConsole);

                JMenu mnHelp = new JMenu("Help");
                menuBar.add(mnHelp);

                JMenuItem mntmHelp = new JMenuItem("Help");
                mnHelp.add(mntmHelp);

                JSeparator separator_2 = new JSeparator();
                mnHelp.add(separator_2);

                JMenuItem mntmAbout = new JMenuItem("About");
                mnHelp.add(mntmAbout);

        }

        /*
         * Updates the UI components of WSN whenever Globals, NetworkListener, or a
         * Run fires and Event.
         *
         * @see java.util.Observer#update(java.util.Observable, java.lang.Object)
         */
        @Override
        public void update(Observable o, Object arg) {
                if (o instanceof Globals) {
                        if (arg == Globals.Events.RUN_REMOVED) {
                                int prevIndx = tabbedRunView.getSelectedIndex();
```

```java
				tabbedRunView.remove(prevIndx);
				checkTabbedRuns();
			} else if (arg == Globals.Events.RUN_ADDED) {
				checkTabbedRuns();
			}

		} else if (o instanceof NetworkListener) {
			if (arg == NetworkListener.Events.CONNECTION_VALID) {
				btnStatus.setText("Connected");
				btnStatus.setBackground(Color.GREEN);
				checkAddNode();
			} else if (arg == NetworkListener.Events.CONNECTION_INVALID) {
				btnStatus.setText("Disconnected");
				btnStatus.setBackground(Color.RED);
				checkAddNode();
			}
		} else if (o instanceof Run) {
			if (o == Globals.getInstance().getActiveRun()) {
				lblCurTime.setText(((Run) o).getFormattedTime());
			}
		}

	}

	/**
	 * checkNodeField Checks where the "Add Node" button should be enabled or
	 * not. For it to be enabled a valid node address must be entered and there
	 * must be a connection to the base station.
	 */
	private void checkAddNode() {
		if (NetworkListener.getInstance().isActive()) {
			short id = 0;
			try {
				id = Short.parseShort(fldId.getText());
			} catch (NumberFormatException e) {
				btnAddNode.setEnabled(false);
			}
			if (id > 0) {
				btnAddNode.setEnabled(true);
				for (Tuple<Short, Sensor> t : Globals.getInstance().getNodes()) {
					if (t.x == id) {
						btnAddNode.setEnabled(false);
					}
				}

			} else {
				btnAddNode.setEnabled(false);
			}
		} else {
			btnAddNode.setEnabled(false);
		}
	}

	/**
	 * checkTabbedRuns updates the Run control buttons based on the number of
	 * Runs and the status of the viewed Run.
	 */
	private void checkTabbedRuns() {
		if (Globals.getInstance().getActiveRun() != null) {
			if (Globals.getInstance().getRuns().size() < Globals.MAX_RUNS) {
				btnNewRun.setEnabled(true);
			} else {
				btnNewRun.setEnabled(false);
			}
			btnDelete.setEnabled(true);
			if (Globals.getInstance().getActiveRun().isRunning()) {
				btnStop.setEnabled(true);
				btnPause.setEnabled(true);
				btnStart.setEnabled(false);
			} else {
```

```
                                btnStop.setEnabled(false);
                                btnPause.setEnabled(false);
                                btnStart.setEnabled(true);
                        }
                } else {
                        btnNewRun.setEnabled(true);
                        btnDelete.setEnabled(false);
                        btnStop.setEnabled(false);
                        btnPause.setEnabled(false);
                        btnStart.setEnabled(false);
                }
        }
}
```

Poly_Sense Server Source Code

```cpp
/*
 * main.cpp
 *
 *  Created on: May 25, 2013
 *      Author: jshirley
 */
#include "server.hpp"
#include "utils.hpp"
int main(int argc, char** argv) {
        char * comport = "/dev/ttyUSB1"; // default com port
    if (argc > 1) {
        comport = argv[1];
    }
    // start server, listen for connections
    int32_t server_socket = tcp_recv_setup();

    debug("sizeof serial_packet: %d\n",sizeof(Serial_packet));

    while (1) {
        // wait for a client
        int client_socket = waitForClient(server_socket);
        debug("client connected\n");

        // start server states
        server(client_socket,comport);
    }
    return 0;
}
```

```
/*
 * server.cpp
 *
 *  Created on: May 25, 2013
 *      Author: jshirley
 */

#include "server.hpp"
#include "serial.hpp"
#include "utils.hpp"
#include "rs232.h"

#define NUM_COMPORT_IDS 30

// Mapping of RS232 com port number to com port path
char RS232_comports[30][16] =
{ "/dev/ttyS0", "/dev/ttyS1", "/dev/ttyS2", "/dev/ttyS3", "/dev/ttyS4",
                "/dev/ttyS5", "/dev/ttyS6", "/dev/ttyS7", "/dev/ttyS8", "/dev/ttyS9",
                "/dev/ttyS10", "/dev/ttyS11", "/dev/ttyS12", "/dev/ttyS13",
                "/dev/ttyS14", "/dev/ttyS15", "/dev/ttyUSB0", "/dev/ttyUSB1",
                "/dev/ttyUSB2", "/dev/ttyUSB3", "/dev/ttyUSB4", "/dev/ttyUSB5",
                "/dev/ttyAMA0", "/dev/ttyAMA1", "/dev/ttyACM0", "/dev/ttyACM1",
                "/dev/rfcomm0", "/dev/rfcomm1", "/dev/ircomm0", "/dev/ircomm1" };

Connection * connection;

bool comportOpen = false;
int comportId = 0;

int tcp_recv_setup()
{
        int server_socket = 0;
        struct sockaddr_in local; /* socket address for local side  */
        socklen_t len = sizeof(local); /* length of local address */

        server_socket = socket(AF_INET, SOCK_STREAM, 0);
        if (server_socket < 0)
        {
                perror("socket call");
                exit(-1);
        }

        local.sin_family = AF_INET; //internet family
        local.sin_addr.s_addr = INADDR_ANY; //wild card machine address
        local.sin_port = htons(55555);

        // try to bind port
        if (bind(server_socket, (struct sockaddr *) &local, sizeof(local)) < 0)
        {
                perror("bind call");
                exit(-1);
        }
        if (getsockname(server_socket, (struct sockaddr*) &local, &len) < 0)
        {
                perror("getsockname call");
                exit(-1);
        }
        printf("socket has port %d \n", ntohs(local.sin_port));
        return server_socket;
}

int waitForClient(int socket)
{
        int client_socket = 0;

        if (listen(socket, 5) < 0)
        {
                perror("listen call");
                exit(-1);
        }
        fcntl(socket, F_SETFL, O_NONBLOCK);

        while (1)
        {
                // check for new clients
                client_socket = 0;
                if ((client_socket = accept(socket, (struct sockaddr*) 0,
                                (socklen_t *) 0)) < 0)
                {
                        if ((errno != EAGAIN) && (errno != EWOULDBLOCK))
                        {
                                perror("accept call");
                                exit(-1);
                        }
                }
                // we have a new client
                if (client_socket > 0)
                {
                        break;
                }
                sleep(1); // wait 1s
        }
        return client_socket;
}

void packetPrint(Serial_packet* packet)
{
        debug("got a whole packet.\nflag: %c, node: %hi, data: ", packet->flag,
                        packet->node_id);
        for (int i = 0; i < sizeof(packet->data); i++)
        {
                debug("%c ", packet->data[i]);
        }
        debug("\n");
}

#define MAX_READ_TRYS 50

bool readSerialPort(int comportId, Serial_packet * packet)
{
        uint8_t readAmt = 0;
        int trys = 0;

        // continously poll comport, building packet as bytes arrive
        while (readAmt < sizeof(Serial_packet))
        {
                readAmt += RS232_PollComport(comportId,
                                (unsigned char*) (((uint8_t *)packet) + readAmt),
                                sizeof(Serial_packet) - readAmt);
                debug("readAmt: %d\n", readAmt);

                // timeout
                if (trys > MAX_READ_TRYS)
                {
                        debug("comport not responding.\n");
                        return false; // don't disconnect from comport just from this...
                }
                trys++;
                usleep(100000);
        }
        return true;
}

bool writeAndReadPacket(int comportId, int socket, Serial_packet *packet)
{
        if ((packet->flag == SENSOR_CONFIG_REQ) || (packet->flag == SENSOR_DATA))
        {
                // send to network
                if (RS232_SendBuf(comportId, (unsigned char *) packet,
```

```c
                                sizeof(Serial_packet)) < 0)
                {
                        debug("write error. closing comport.\n");
                        RS232_CloseComport(comportId);
                        return false;
                }

                // data recv
                if (readSerialPort(comportId, packet))
                {
                        // fix 8bit mc issue
                        packet->node_id = (packet->node_id >> 8) | (packet->node_id << 8);

                        // send to frontend
                        if (send(socket, packet, sizeof(Serial_packet), 0) < 0)
                        {
                                debug("client disconnected\n");
                                return false;
                        }
                }
                return true; // don't close comport just cause readSerialPort timed out
        }
        else if (packet->flag == PING)
        {
                debug("ping\n");
                return true;
        }
        return true;
}

bool handleClosedComportForPacket(int comportId, int socket,
                Serial_packet *packet)
{
        if (packet->flag == SENSOR_CONFIG_REQ)
        {
                debug(
                                "client sent CONFIG req with no comport active! sleeping for a 700ms to simulate network latency\n");
                // TODO need to tell the client that the comport is not set up properly --- for now just return packet
                // Will leave this, good for testing.
                usleep(700000);


                if (send(socket, packet, sizeof(Serial_packet), 0) < 0)
                {
                        debug("client disconnected\n");
                }
                return true;
        }
        else if (packet->flag == SENSOR_DATA)
        {
                debug(
                                "client sent DATA req with no comport active! sleeping for a 700ms to simulate network latency\n");
                // TODO need to tell the client that the comport is not set up properly --- for now just return packet
                // Will leave this, good for testing.
                usleep(700000);

                // fix 8bit mc issue
                packet->node_id = htons(packet->node_id);

                // fake a packet
                packet->data[0] = 's';
                packet->data[1] = 1;
                packet->data[2] = 2;
                packet->data[3] = 's';
                packet->data[4] = 3;
                packet->data[5] = 4;
                packet->data[6] = 's';
                packet->data[7] = 5;
                packet->data[8] = 6;

                packetPrint(packet);

                if (send(socket, packet, sizeof(Serial_packet), 0) < 0)
                {
                        debug("client disconnected\n");
                }
                return true;
        }
        // debug info
        else if (packet->flag == DEBUG)
        {
                debug("Frontend: %s\n", packet->data);
                return true;
        }
        else if (packet->flag == PING)
        {
                debug("ping\n");
                return true;
        }
        // possibly data? but not needed
        else
        {
                debug("bad flag! disconnecting for now.\n");
                return false;
        }
}

uint8_t checkCommand(int socket)
{
        struct timeval t;
        t.tv_sec = 0;
        t.tv_usec = 0;

        fd_set fds;
        FD_ZERO(&fds);
        FD_SET(socket, &fds);
        debug("Starting blocking select for client cmd.\n");
        select(socket + 1, &fds, NULL, NULL, NULL);

        // will only return when data on socket
        if (FD_ISSET(socket, &fds))
        {
                debug("got data from the client!\n");
                return true;
        }
        return false;
}

bool processCommand(int comportId, int socket)
{
        Serial_packet packet;
        int readAmt = recv(socket, &packet, sizeof(Serial_packet), 0);
        if (readAmt < 0)
        {
                debug("client disconnected\n");
                return false;
        }

        // got a whole packet
        if (readAmt == sizeof(Serial_packet))
        {
                debug("packet from client: %c %hi %s\n", packet.flag, packet.node_id,
                                packet.data);

                // serial dependent flags allowed only if comport is available
                if (comportOpen)
                {
                        // sensor setup OR data req only require to be forwarded
                        return writeAndReadPacket(comportId, socket, &packet);
                }
                // comport is DOWN
                else
                {
                        return handleClosedComportForPacket(comportId, socket, &packet);
                }
        }
        else if (readAmt == 1)
        {
                debug("ping.\n");
                return true;
        }
        else
        {
                debug("not sure what we got from the client, disconnecting\n");
                return false;
        }

}

int getComportIdFor(char * file)
{
        int i;
        for (i = 0; i < NUM_COMPORT_IDS; i++)
        {
                if (strcmp(file, RS232_comports[i]) == 0)
                {
                        return i;
                }
        }
        debug("Couldn't find comportId for %s\n", file);
        return -1;
}

void server(int socket, char * comport)
{
        comportId = getComportIdFor(comport);
        if (RS232_OpenComport(comportId, 9600) == 1)
        {
                comportOpen = false;
                debug("comport open failed\n");
        }
        else
        {
                comportOpen = true;
        }

        Serial_packet packet;
        while (1)
        {

                if (checkCommand(socket))
                {
                        if (!processCommand(comportId, socket))
                        {
                                // processCommand returns false if the socket gets closed
                                break;
                        }
                }
        }
        debug("client must have left.\n");
        if (comportOpen)
        {
                RS232_CloseComport(comportId);
                comportOpen = false;
        }
}
```

```
/*
 * server.hpp
 *
 *  Created on: May 25, 2013
 *      Author: jshirley
 */

#ifndef SERVER_HPP_
#define SERVER_HPP_

// c++
#include <cstdlib>
#include <cstdio>
#include <iostream>
#include <set>

// c
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/wait.h>
#include <sys/uio.h>
#include <sys/time.h>
#include <fcntl.h>
#include <string.h>
#include <strings.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <errno.h>
#include <sys/timeb.h>


int tcp_recv_setup();
int waitForClient(int socket);
void server(int socket,char * comport);


enum flags {
    SER_DATA_REQ = '1',
    SENSOR_CONFIG_REQ = '2',
    SENSOR_DATA = '3',
    DEBUG = '4',
    PING = '5',
    SENSOR_CONFIG_ERR = '6',

};

typedef struct connection Connection;

// connection contains the necessary information for
// communication over a TCP socket.
struct connection {
    int32_t sk_num;
    struct sockaddr_in remote;
    uint32_t len;
};

typedef struct serial_packet Serial_packet;

// serial_packet contains the information to be
// transfered to the base station and Poly_Sense
// Monitor.
struct __attribute__((__packed__)) serial_packet {
    uint8_t flag;
    uint16_t node_id;
    uint8_t data[15];
};


#endif /* SERVER_HPP_ */
```

```cpp
/*
 * utils.hpp
 *
 *  Created on: May 25, 2013
 *      Author: jshirley
 */

#ifndef UTILS_HPP_
#define UTILS_HPP_

#include <stdio.h>

#define debug(...) fprintf(stderr, __VA_ARGS__)
#define debug // disables printing

#endif /* UTILS_HPP_ */
```

Poly_Sense Node and Base station Source Code

```
//**************************************************************************************
/** \file BMP085_driver.h
 *   This is the header for a driver class used to configure and acquire data from the
 *   Bosch BMP085 sensor
 *
 *  Revisions:
 *    \li 04-05-2013 HV Created file and version 1.0 of driver
 *
 *  License:
 *    This file is copyright 2013 by HA Vierra and released under the Lesser GNU
 *    Public License, version 2. It intended for educational use only, but its use
 *    is not limited thereto. */
/*    THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
 *    AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 *    IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 *    ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
 *    LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUEN-
 *    TIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 *    OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
 *    CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
 *    OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
 *    OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. */
//**************************************************************************************

// This define prevents this .H file from being included multiple times in a .CPP file
#ifndef _BMP085_DRIVER_H_
#define _BMP085_DRIVER_H_

#include <stdlib.h>                                                     // Include standard library header files
#include <avr/io.h>
#include "emstream.h"                       // Header for serial ports and devices
#include "FreeRTOS.h"                       // Header for the FreeRTOS RTOS
#include "queue.h"                          // Header for FreeRTOS queues
#include "rs232int.h"                       // Include header for serial port class
#include "frt_text_queue.h"
#include "shares.h"
#include "i2c_master.h"

//---------------------------------------------Register Addresses--------------------------------------------//
#define BMP085_ADDRESS  0x77 // 7-bit Slave address

#define BMP085_OUT_XLSB 0xF8 //
#define BMP085_OUT_L    0xF7 //
#define BMP085_OUT_H    0xF6 //
#define BMP085_CTRL         0xF4 //
#define BMP085_RESET    0xE0 //
#define BMP085_WIA              0xD0 //
#define BMP085_CAL0             0xAA //
#define BMP085_CAL1             0xAB //
#define BMP085_CAL2             0xAC //
#define BMP085_CAL3             0xAD //
#define BMP085_CAL4             0xAE //
#define BMP085_CAL5             0xAF //
#define BMP085_CAL6             0xB0 //
#define BMP085_CAL7             0xB1 //
#define BMP085_CAL8             0xB2 //
#define BMP085_CAL9             0xB3 //
#define BMP085_CAL10    0xB4 //
#define BMP085_CAL11    0xB5 //
#define BMP085_CAL12    0xB6 //
#define BMP085_CAL13    0xB7 //
#define BMP085_CAL14    0xB8 //
#define BMP085_CAL15    0xB9 //
#define BMP085_CAL16    0xBA //
#define BMP085_CAL17    0xBB //
#define BMP085_CAL18    0xBC //
#define BMP085_CAL19    0xBD //
#define BMP085_CAL20    0xBE //
```

```
#define BMP085_CAL21    0xBF //

//-------------------------------------------------------------------------------------
/** \brief This class makes a BMP085 pressure sensor driver object.
 */
class BMP085_driver : public i2c_master
{
    protected:
                emstream* ptr_to_serial;
    public:
                BMP085_driver(emstream*);

                //---------------------------------------Method Prototypes----------------------------------------//
                void Setup(void);
                void Test_I2C(void);
                bool Check_Registers(void);
                void Read_Cal_Data();
                int32_t Read_Temp();//returns Temp
                int32_t Read_Pres();// returns pressure

                //------------------------------------------ Globals----------------------------------------------//
                int16_t AC1;
                int16_t AC2;
                int16_t AC3;
                uint16_t AC4;
                uint16_t AC5;
                uint16_t AC6;
                int16_t B1;
                int16_t B2;
                int16_t MB;
                int16_t MC;
                int16_t MD;

                uint8_t AC1_L;
                uint8_t AC1_H;
                uint8_t AC2_L;
                uint8_t AC2_H;
                uint8_t AC3_L;
                uint8_t AC3_H;
                uint8_t AC4_L;
                uint8_t AC4_H;
                uint8_t AC5_L;
                uint8_t AC5_H;
                uint8_t AC6_L;
                uint8_t AC6_H;
                uint8_t B1_L;
                uint8_t B1_H;
                uint8_t B2_L;
                uint8_t B2_H;
                uint8_t MB_L;
                uint8_t MB_H;
                uint8_t MC_L;
                uint8_t MC_H;
                uint8_t MD_L;
                uint8_t MD_H;

                int32_t X1;
                int32_t X2;
                int32_t X3;
                int32_t B3;
                uint32_t B4;
                int32_t B5;
                int32_t B6;
                int32_t B7;

                int32_t pressure_pa;
                int32_t temp_raw;
};
```

```
#endif // _BMP085_DRIVER_H_
```

```
//**************************************************************************************
/** \file BMP085_driver.cpp
 *   This is a sensor driver class used to configure and acquire data from the
 *   Bosch BMP085 sensor
 *
 *  Revisions:
 *    \li 04-05-2013 HV Created file and version 1.0 of driver
 *
 *  License:
 *    This file is copyright 2013 by HA Vierra and released under the Lesser GNU
 *    Public License, version 2. It intended for educational use only, but its use
 *    is not limited thereto. */
/*    THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
 *    AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 *    IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 *    ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
 *    LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUEN-
 *    TIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 *    OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
 *    CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
 *    OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
 *    OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. */
//**************************************************************************************

#include "BMP085_driver.h"      // Header for this class
#include "shares.h"
#include <avr/interrupt.h>
#include <util/delay.h>
#include <math.h>

//-------------------------------------------------------------------------------------
/** This constructor creates a BMP085 sensor driver object which reads data via I2C.
 *  @param p_debug_port A serial port, often RS-232, for debugging text (default: NULL)
 */
BMP085_driver::BMP085_driver (emstream* p_debug_port)
    :i2c_master(p_debug_port)
{
        //--------setup the BMP085---------//
        *p_serial<<endl<<
        "//------------------------BMP085 Setup---------------------------//"<<endl;
        Test_I2C();
        Read_Cal_Data();
}

/** This function tests the I2C bus for the BMP085's address
 */
void BMP085_driver::Test_I2C()
{
        uint8_t Data = 0x00;
        I2C_read(BMP085_ADDRESS, BMP085_WIA, &Data);

        if(Data == 0x55)
        {
                *p_serial<<"\nI2C Read Test Passed, BMP085 Address: "<<hex<< Data<<endl;
        }
        else
        {
                *p_serial<<"ERROR: I2C Read Test Failed"<<endl;
                //while(1){}
        }
}
/** This function reads calibration data from the BMP085, and stores it for
 *  later use in data conversion
 */
void BMP085_driver::Read_Cal_Data()
{
        *p_serial<<"Calibration Data Read..."<<endl;
        I2C_read(BMP085_ADDRESS, BMP085_CAL0, &AC1_H);
        I2C_read(BMP085_ADDRESS, BMP085_CAL1, &AC1_L);
        I2C_read(BMP085_ADDRESS, BMP085_CAL2, &AC2_H);
        I2C_read(BMP085_ADDRESS, BMP085_CAL3, &AC2_L);
        I2C_read(BMP085_ADDRESS, BMP085_CAL4, &AC3_H);
        I2C_read(BMP085_ADDRESS, BMP085_CAL5, &AC3_L);
        I2C_read(BMP085_ADDRESS, BMP085_CAL6, &AC4_H);
        I2C_read(BMP085_ADDRESS, BMP085_CAL7, &AC4_L);
        I2C_read(BMP085_ADDRESS, BMP085_CAL8, &AC5_H);
        I2C_read(BMP085_ADDRESS, BMP085_CAL9, &AC5_L);
        I2C_read(BMP085_ADDRESS, BMP085_CAL10, &AC6_H);
        I2C_read(BMP085_ADDRESS, BMP085_CAL11, &AC6_L);
        I2C_read(BMP085_ADDRESS, BMP085_CAL12, &B1_H);
        I2C_read(BMP085_ADDRESS, BMP085_CAL13, &B1_L);
        I2C_read(BMP085_ADDRESS, BMP085_CAL14, &B2_H);
        I2C_read(BMP085_ADDRESS, BMP085_CAL15, &B2_L);
        I2C_read(BMP085_ADDRESS, BMP085_CAL16, &MB_H);
        I2C_read(BMP085_ADDRESS, BMP085_CAL17, &MB_L);
        I2C_read(BMP085_ADDRESS, BMP085_CAL18, &MC_H);
        I2C_read(BMP085_ADDRESS, BMP085_CAL19, &MC_L);
        I2C_read(BMP085_ADDRESS, BMP085_CAL20, &MD_H);
        I2C_read(BMP085_ADDRESS, BMP085_CAL21, &MD_L);

        AC1 = (int16_t)((AC1_H<<8)|AC1_L);
        AC2 = (int16_t)((AC2_H<<8)|AC2_L);
        AC3 = (int16_t)((AC3_H<<8)|AC3_L);
        AC4 = (uint16_t)((AC4_H<<8)|AC4_L);
        AC5 = (uint16_t)((AC5_H<<8)|AC5_L);
        AC6 = (uint16_t)((AC6_H<<8)|AC6_L);
        B1  = (int16_t)((B1_H<<8)|B1_L);
        B2  = (int16_t)((B2_H<<8)|B2_L);
        MB  = (int16_t)((MB_H<<8)|MB_L);
        MC  = (int16_t)((MC_H<<8)|MC_L);
        MD  = (int16_t)((MD_H<<8)|MD_L);

        *p_serial<<"Calibration Data Read Complete"<<endl<<endl;
}

/** This function reads the temperature data from the BMP085 and stores it.
 *  The temperature data is also used to determine the atmospheric pressure.
 */
int32_t BMP085_driver::Read_Temp()
{
        uint8_t temp_h = 0;
        uint8_t temp_l = 0;
        int32_t UT = 0;

        //Request a temperature read
        I2C_write(BMP085_ADDRESS, BMP085_CTRL, 0x2E);
        _delay_ms(5);//wait at least 4.5 seconds for conversion

        //Read temperature registers
        I2C_read(BMP085_ADDRESS, BMP085_OUT_H, &temp_h);
        I2C_read(BMP085_ADDRESS, BMP085_OUT_L, &temp_l);

        //Convert to celcius
        UT = (int32_t)(((uint16_t)temp_h<<8)|temp_l);
        X1 = ((int32_t)(UT - AC6)*(int32_t)AC5)>>15;
        X2 = ((int32_t)MC<<11)/(X1+MD);
        B5 = X1 + X2;

        temp_raw = (int32_t)(B5 + 8)>>4;

        return(temp_raw);//returns temp
}

/** This function reads the pressure data from the BMP085 and stores it.
 *  It uses the calibration data, and temperature to calculate the atmospheric pressure
 *  in Pascals.
 */
int32_t BMP085_driver::Read_Pres()
{
        uint8_t pres_h = 0;
```

```c
        uint8_t pres_m = 0;
        uint8_t pres_l = 0;
        int32_t UP = 0;
        int32_t p = 0;

        //Request a pressure read
        I2C_write(BMP085_ADDRESS, BMP085_CTRL, 0x34);
        _delay_ms(2);//wait for conversion to complete

        //Read Pressure
        I2C_read(BMP085_ADDRESS,  BMP085_OUT_H, &pres_h);
        I2C_read(BMP085_ADDRESS,  BMP085_OUT_L, &pres_m);
        I2C_read(BMP085_ADDRESS,  BMP085_OUT_XLSB, &pres_l);

        Read_Temp();

        UP = (int32_t)((((uint32_t)pres_h<<16)|((uint32_t)pres_m<<8)|pres_l)>>8);
        B6 = B5 - 4000;
        X1 = ((B2*(B6*B6)>>12)>>11);
        X2 = (AC2*B6)>>11;
        X3 = X1 + X2;
        B3 = ((((int32_t)AC1*4+X3)<<0)+2)>>2;
        X1 = (AC3 *B6)>>13;
        X2 = (B1 * ((B6 * B6)>>12))>>16;
        X3 = ((X1 + X2) + 2)>>2;
        B4 = (AC4 * (uint32_t)(X3 + 32768))>>15;
        B7 = ((uint32_t)(UP-B3)*(5000>>0));
        if(B7 < (int32_t)0x80000000)
                p = (B7<<1)/B4;
        else
                p = (B7/B4)<<1;
        X1 = (p>>8) * (p>>8);
        X1 = (X1*3038)>>16;
        X2 = (-7357*p)>>16;

        pressure_pa = (p + ((X1 + X2 + 3791)>>4));
        return(pressure_pa);//returns pres in units of Pa
}
```

```
//*************************************************************************************
/** \file i2c_master.h
 *    This file contains a base class for classes that use the I2C (also known as TWI)
 *    interface on an AVR. The terms "I2C" (the two means squared) and "TWI" are
 *    essentially equivalent; Philips has trademarked the former, and Atmel doesn't pay
 *    them a license fee, so Atmel chips that meet exactly the same specification are
 *    not allowed to use the "I2C" name, even though everything works the same.
 *
 *    Note: The terms "master" and "slave" are standard terminology used in the
 *    electronics industry to describe interactions between electronic components only.
 *    The use of such terms in this documentation is made only for the purpose of
 *    usefully documenting electronic hardware and software, and such use must not be
 *    misconstrued as diminishing our revulsion at the socially diseased human behavior
 *    which is described using the same terms, nor implying any insensitivity toward
 *    people from any background who have been affected by such behavior.
 *
 *  Revised:
 *    \li 12-24-2012 JRR Original file, as a standalone HMC6352 compass driver
 *    \li 12-28-2012 JRR I2C driver split off into a base class for optimal reusability
 *    \li 2-11-2013 HV Added sensor_read() and sensor_write() methods and defined i2c
 *                  status codes
 *  License:
 *    This file is copyright 2012 by JR Ridgely and released under the Lesser GNU
 *    Public License, version 2. It is intended for educational use only, but its use
 *    is not limited thereto. */
/*    THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
 *    AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 *    IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 *    ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
 *    LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUEN-
 *    TIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 *    OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
 *    CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
 *    OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
 *    OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. */
//*************************************************************************************

// This define prevents this file from being included more than once in a *.cpp file
#ifndef _I2C_MASTER_H_
#define _I2C_MASTER_H_

#include <stdlib.h>                         // Standard C/C++ library stuff

#include "emstream.h"                       // Header for base serial devices


/// This is the desired bit rate for the I2C interface in bits per second.
#define I2C_BITRATE         600000L//800000L//100000L

/// This value is put in the TWBR register to set the desired bitrate.
const uint8_t I2C_TWBR_VALUE = (((F_CPU / I2C_BITRATE) - 16) / 2);
//-----------------Status Codes for Master Transmitter Mode-----------------//
const uint8_t START_TX = 0X08;
const uint8_t REPEAT_START_TX = 0X10;
const uint8_t SLA_W_TX_ACK_RX = 0X18;
const uint8_t SLA_W_TX_NOT_ACK_RX = 0X20;
const uint8_t DATA_TX_ACK_RX = 0X28;
const uint8_t DATA_TX_NOT_ACK_RX = 0X30;
const uint8_t ARB_LOST = 0X38;

//-----------------Status Codes for Master Receiver Mode-----------------//
const uint8_t SLA_R_TX_ACK_RX = 0X40;
const uint8_t SLA_R_TX_NOT_ACK_RX = 0X48;
const uint8_t DATA_RX_ACK_RX = 0X50;
const uint8_t DATA_RX_NOT_ACK_RX = 0X58;




//-------------------------------------------------------------------------------------
```

```
/** \brief This class is a simple driver for an I2C (also known as TWI) bus on an AVR
 *  processor.
 *  \details It encapsulates basic I2C functionality such as the ability to send and
 *  receive bytes through the TWI bus. Currently only operation of the AVR as an I2C
 *  bus master is supported; this is what's needed for the AVR to interface with most
 *  I2C based sensors.
 */

class i2c_master
{
protected:
        /// This is a pointer to a serial port object which is used for debugging the code.
        emstream* p_serial;

public:
        // This constructor sets up the driver
        i2c_master (emstream* = NULL);

        // This destructor doesn't exist...psych

        // This method causes a start condition on the TWI bus
        void start (void);

        // This method causes a repeated start on the TWI bus
        void repeated_start (void);

        /** This method causes a stop condition on the I2C bus. It's inline because
         *  causing a stop condition is a one-liner (in C++ and even in assembly).
        */
        void stop (void){TWCR = (1 << TWINT) | (1 << TWSTO) | (1 << TWEN);};

        // This method sends a byte out the TWI bus
        bool send (uint8_t, uint8_t);
        // This method receives a byte from the TWI bus
        uint8_t receive (bool);

        //Methods making sensor read/write easy
        void I2C_read (uint8_t , uint8_t , uint8_t*);
        void I2C_write(uint8_t , uint8_t , uint8_t);

        //Method to print error messages
        void print_error_message(uint8_t error_code, uint8_t expected_code);
};

#endif // _I2C_MASTER_H_
```

```
//**************************************************************************************
/** \file i2c_master.cpp
 *    This file contains a base class for classes that use the I2C (also known as TWI)
 *    interface on an AVR. The terms "I2C" (the two means squared) and "TWI" are
 *    essentially equivalent; Philips has trademarked the former, and Atmel doesn't pay
 *    them a license fee, so Atmel chips that meet exactly the same specification are
 *    not allowed to use the "I2C" name, even though everything works the same.
 *
 *    Note: The terms "master" and "slave" are standard terminology used in the
 *    electronics industry to describe interactions between electronic components only.
 *    The use of such terms in this documentation is made only for the purpose of
 *    usefully documenting electronic hardware and software, and such use must not be
 *    misconstrued as diminishing our revulsion at the socially diseased human behavior
 *    which is described using the same terms, nor implying any insensitivity toward
 *    people from any background who have been affected by such behavior.
 *
 *  Revised:
 *    \li 12-24-2012 JRR Original file, as a standalone HMC6352 compass driver
 *    \li 12-28-2012 JRR I2C driver split off into a base class for optimal reusability
 *    \li 2-11-2013 HV Added sensor_read() and sensor_write() methods and defined i2c
 *                   status codes
 *    \li 4-14-2013 HV Added print_error_message()
 *
 *  License:
 *    This file is copyright 2012 by JR Ridgely and released under the Lesser GNU
 *    Public License, version 2. It is intended for educational use only, but its use
 *    is not limited thereto. */
/*    THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
 *    AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 *    IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 *    ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
 *    LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUEN-
 *    TIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 *    OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
 *    CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
 *    OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
 *    OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. */
//**************************************************************************************

#include "FreeRTOS.h"                       // Main header for FreeRTOS
#include "task.h"                           // Needed for the vTaskDelay() function
#include "i2c_master.h"                     // Header for this class
#include <util/delay.h>
#include "shares.h"

#define USE_TIMEOUT_TICKS 1

//shared_data<bool>* I2C_error;
const uint8_t DELAY = 5;
const uint8_t TIMOUT_TICKS = 250;
//--------------------------------------------------------------------------------------
/** This constructor creates an I2C driver object.
 *  @param p_debug_port A serial port, often RS-232, for debugging text (default: NULL)
 */

i2c_master::i2c_master (emstream* p_debug_port)
{
        p_serial = p_debug_port;                // Set the debugging serial port pointer

        TWBR = I2C_TWBR_VALUE;                  // Set the bit rate for the I2C port
}


//--------------------------------------------------------------------------------------
/** This method causes a start condition on the I2C bus. In hardware, a start condition
 *  means that the SDA line is dropped while the SCL line stays high. This gets the
 *  attention of all the other devices on the bus so that they will listen for their
 *  addresses.
 */

```

```
void i2c_master::start (void)
{
        // Cause the start condition to happen
        TWCR = (1 << TWINT) | (1 << TWSTA) | (1 << TWEN);

        // Wait for the TWINT bit to indicate that the start condition has been completed
        if(USE_TIMEOUT_TICKS)
        {
                for (uint8_t tntr = 0; !(TWCR & (1 << TWINT)); tntr++)
                {
                        if (tntr > TIMOUT_TICKS)
                        {
                                DBG (p_serial, PMS ("I2C start timeout") << endl);
                                break;
                        }
                }
        }
        else
                while(!(TWCR & (1 << TWINT)));

        // Check that the start condition was transmitted OK
        if ((TWSR & 0xF8) != START_TX)
        {
                print_error_message((uint8_t)TWSR, (uint8_t)0x08);
                //DBG (p_serial, PMS ("I2C start: 0x") << hex << TWSR << PMS (" not 0x08")
                        // << dec << endl);
        }
}

//-------------------------------------------------------------------------------------
/** This method causes a repeated start condition on the I2C bus. This is similar to
 *  a regular start condition, except that a different return code is expected if
 *  things go as they should.
 */

void i2c_master::repeated_start (void)
{
        // Cause the start condition to happen
        TWCR = (1 << TWINT) | (1 << TWSTA) | (1 << TWEN);

        // Wait for the TWINT bit to indicate that the start condition has been completed
        if(USE_TIMEOUT_TICKS)
        {
                for (uint8_t tntr = 0; !(TWCR & (1 << TWINT)); tntr++)
                {
                        if (tntr > TIMOUT_TICKS)
                        {
                                DBG (p_serial, PMS ("I2C re-start timeout") << endl);
                                //I2C_error->put(true);
                                break;

                        }
                }
        }
        else
                while(!(TWCR & (1 << TWINT)));

        // Check that the start condition was transmitted OK
        if ((TWSR & 0xF8) != REPEAT_START_TX)
        {
                print_error_message((uint8_t)TWSR, (uint8_t)0x10);
                //DBG (p_serial, PMS ("I2C re-start: 0x") << hex << TWSR << PMS (" not 0x10")
                         //<< dec << endl);
        }
}


//-------------------------------------------------------------------------------------
/** This method performs an I2C send to transfer a byte to a remote device. The
 *  expected response code varies depending on what is being sent at what time; some
 *  examples of expected responses are as follows:
 *  \li \c 0x18 - When one has sent SLA+W, a slave address for a write command, and a
 *                good ACK has been received
 *  \li \c 0x40 - When one has sent SLA+R, a slave address for a read command, and a
 *                good ACK has been received
 *  \li \c 0x28 - When one has transmitted a data byte and received a good ACK
 *  @param byte_to_send The byte which is being sent to the remote device
 *  @param expected_response A response byte which the I2C port in the AVR will give
 *                           if the transfer goes correctly
 *  @return True if the transmission was successful and false if not
 */

bool i2c_master::send (uint8_t byte_to_send, uint8_t expected_response)
{
        TWDR = byte_to_send;
        TWCR = (1 << TWINT) | (1 << TWEN);
        if(USE_TIMEOUT_TICKS)
        {
                for (uint8_t tntr = 0; !(TWCR & (1 << TWINT)); tntr++)
                {
                        if (tntr > TIMOUT_TICKS)
                        {
                                DBG (p_serial, PMS ("I2C send timeout") << endl);
                                return (false);

                        }
                }
        }
        else
                while(!(TWCR & (1 << TWINT)));
        // Check that the address thingy was transmitted OK
        if ((TWSR & 0xF8) != expected_response)
        {
                print_error_message((uint8_t)TWSR, (uint8_t)expected_response);
                //DBG (p_serial, PMS ("I2C send: 0x") << hex << TWSR << PMS (" not 0x")
                //       << expected_response << dec << endl);
                return (false);
        }
        return (true);
}


//-------------------------------------------------------------------------------------
/** This method receives a byte from the I2C bus. Other code must have already run the
 *  \c start() command and sent and address byte which got the MPU6050's attention.
 *  @param ack True if we end our data request with ACK, telling the slave that we
 *             want more data after this; false if we end our data request with NACK,
 *             telling the slave that we don't want more data after this
 *  @return The byte which was received from the remote device
 */

uint8_t i2c_master::receive (bool ack)
{
        uint8_t expected_response;              // Code we expect from the AVR's I2C port

        if (ack)  // If we expect more data after this, send an ACK after we get the data
        {
                TWCR = (1 << TWINT) | (1 << TWEN) | (1 << TWEA);
                expected_response = DATA_RX_ACK_RX;
        }
        else      // We're not going to ask for more data; send a NACK when we're done
        {
                TWCR = (1 << TWINT) | (1 << TWEN);
                expected_response = DATA_RX_NOT_ACK_RX;
        }

        if(USE_TIMEOUT_TICKS)
        {
                for (uint8_t tntr = 0; !(TWCR & (1 << TWINT)); tntr++)
                {
                        if (tntr > TIMOUT_TICKS)
                        {
                                DBG (p_serial, PMS ("I2C receive timeout") << endl);
                                //I2C_error->put(true);
                                break;
                        }
                }
        }
        else
                while(!(TWCR & (1 << TWINT)));

        // Check that the address thingy was transmitted OK
        if ((TWSR & 0XF8) != expected_response)
        {
                print_error_message((uint8_t)TWSR, (uint8_t)expected_response);
                //DBG (p_serial, PMS ("I2C receive: 0x") << hex << TWSR << PMS (" not 0x")
                //       << expected_response << dec << endl);
        }

        return (TWDR);
}


/** This method utilizes other methods within i2c_master to provide an easy single byte read function
 *  \c start() command and sent and address byte which got the MPU6050's attention.
 *  \c send() command to send the slave address  then register address on SDA line
 *  \c repeated_start() command to repeat the start condition, trasitioning between a write and read
 *  \c receive() command to read data present on the SDA line
 *  \c stop() command to send stop condition to the MPU6050
 *
 *  @param slave_address_7b 7 bit sensor address, for the MPU6050 it's 0x68
 *  @param register_address 8 bit register address to be read from
 *  @param data pointer to an 8 bit variable to store the read data
 */
void i2c_master::I2C_read(uint8_t slave_address_7b, uint8_t register_address, uint8_t* data)
{
        //if(I2C_error->get())
        //      return;
    //vTaskDelay(1);
        _delay_us(DELAY);
    uint8_t slave_address_R = ((slave_address_7b<<1)|1); // add the 'R' bit to read
    uint8_t slave_address_W = (slave_address_7b<<1); // add the 'W' bit to write

    start ();                              // Send an I2C start condition
        //_delay_us(DELAY);
    send (slave_address_W, SLA_W_TX_ACK_RX);                      // Now send the write address thingy
        //_delay_us(DELAY);
    send (register_address, DATA_TX_ACK_RX);                     // who am I address
        //_delay_us(DELAY);
    repeated_start ();
        //_delay_us(DELAY);
    send (slave_address_R, SLA_R_TX_ACK_RX);
        //_delay_us(DELAY);
    *data = receive (false);
        //_delay_us(DELAY);
    stop ();
        _delay_us(DELAY);
}

/** This method utilizes other methods within i2c_master to provide an easy single byte write function
 *  \c start() command and sent and address byte which got the MPU6050's attention.
 *  \c send() command to send the slave address  then register address on SDA line
 *  \c repeated_start() command to repeat the start condition, trasitioning between a write and read
 *  \c receive() command to read data present on the SDA line
 *  \c stop() command to send stop condition to the MPU6050
 *
 *  @param slave_address_7b 7 bit sensor address, for the MPU6050 it's 0x68
 *  @param register_address 8 bit register address to be read from
 *  @param data pointer to an 8 bit value to transmit
 */
void i2c_master::I2C_write(uint8_t slave_address_7b, uint8_t register_address, uint8_t data)
{
        //if(I2C_error->get())
        //      return;
        //portTickType previousTicks = xTaskGetTickCount ();
        //delay_from_to (previousTicks, configMS_TO_TICKS (200));
        //vTaskDelay(1);
    _delay_us(DELAY);
    uint8_t slave_address_W = (slave_address_7b<<1); // add the 'W' bit to write
    start ();
        //_delay_us(DELAY);
    //DBG (p_serial, PMS ("Got past start()") << endl);// Send an I2C start condition
    send (slave_address_W, SLA_W_TX_ACK_RX);                      // Now send the write address thingy
        //_delay_us(DELAY);
    //DBG (p_serial, PMS ("Sent slave address") << endl);
    send (register_address, DATA_TX_ACK_RX);
        //_delay_us(DELAY);
    //DBG (p_serial, PMS ("sent reg address") << endl);
    send (data, DATA_TX_ACK_RX);
        //_delay_us(DELAY);
    //DBG (p_serial, PMS ("sent data") << endl);
    stop ();
        _delay_us(DELAY);
    //DBG (p_serial, PMS ("stopped") << endl);
}

/** This method prints any error status thrown during an I2C transaction
 *
 *  @param error_code 8-bit AVR code corresponding to a particular I@C status
 *  @param expected_code 8-bit AVR code corresponding to a particular I@C status
 */
void i2c_master::print_error_message(uint8_t error_code, uint8_t expected_code)
{
        uint8_t i = 0;
        uint8_t num_error_codes = 2;
        uint8_t current_error_code;
        for(i = 0; i < num_error_codes; i++)
        {
                if(i == 0)
                {
                        current_error_code = error_code;
                        *p_serial<<"I2C Error. Status Recieved: "<<endl;
                }
                else
                {
                        current_error_code = expected_code;
                        *p_serial<<endl<<"Expected Status: "<<endl;
                }

                switch(current_error_code)
                {
                        case START_TX:
                                *p_serial<<"A START condition has been transmitted"<<endl;
                                break;
                        case REPEAT_START_TX:
                                *p_serial<<"A repeated START condition has been transmitted"<<endl;
                                break;
                        case SLA_W_TX_ACK_RX:
                                *p_serial<<"SLA+W has been transmitted; ACK has been received"<<endl;
                                break;
                        case SLA_W_TX_NOT_ACK_RX:
                                *p_serial<<"SLA+W has been transmitted; NOT ACK has been received"<<endl;
                                break;
                        case DATA_TX_ACK_RX:
                                *p_serial<<"Data byte has been transmitted; ACK has been received"<<endl;
                                break;
                        case DATA_TX_NOT_ACK_RX:
                                *p_serial<<"Data byte has been transmitted; NOT ACK has been received"<<endl;
                                break;
                        case ARB_LOST:
                                *p_serial<<"Arbitration lost in SLA+W or data bytes"<<endl;
                                break;
                        case SLA_R_TX_ACK_RX:
                                *p_serial<<"SLA+R has been transmitted; ACK has been received"<<endl;
                                break;
                        case SLA_R_TX_NOT_ACK_RX:
                                *p_serial<<"SLA+R has been transmitted; NOT ACK has been received"<<endl;
                                break;
                        case DATA_RX_ACK_RX:
                                *p_serial<<"Data byte has been received; ACK has been returned"<<endl;
                                break;
                        case DATA_RX_NOT_ACK_RX:
                                *p_serial<<"Data byte has been received; NOT ACK has been returned"<<endl;
                                break;
                        default:
                                *p_serial<<"Unknown Status"<<endl;
                                break;
                }
        }
        *p_serial<<endl;
        stop();
        _delay_ms(5);

        //I2C_error->put(1);
}


```

```
#--------------------------------------------------------------------------------------
# File:    Makefile for an AVR project
#          The makefile is the standard way to control the compilation and linking of
#          C/C++ files into an executable file. This makefile is also used to control
#          the downloading of the executable file to the target processor and the
#          generation of documentation for the project.
#
# Version:  4-11-2004 JRR Original file
#           6-19-2006 JRR Modified to use AVR-JTAG-ICE for debugging
#          11-21-2008 JRR Added memory locations and removed extras for bootloader
#          11-26-2008 JRR Cleaned up; changed method of choosing programming method
#          11-14-2009 JRR Added make support to put library files into subdirectory
#           9-28-2012 JRR Restructured to work with FreeRTOS subdirectory
#           4-20-2013 HAV Removed all contents unused by Poly_Sense and updated with
#                         new targets
#           5-03-2013 JLS Added defines for make node and make base commands
#
# Relies   The avr-gcc compiler and avr-libc library
# on:      The avrdude downloader, if downloading through an ISP port
#          AVR-Insight or DDD and avarice, if debugging with the JTAG port
#          Doxygen, for automatic documentation generation
#
# Copyright 2006-2012 by JR Ridgely.  This makefile is intended for use in educational
# courses only, but its use is not restricted thereto. It is released under the terms
# of the Lesser GNU Public License with no warranty whatsoever, not even an implied
# warranty of merchantability or fitness for any particular purpose. Anyone who uses
# this file agrees to take all responsibility for any and all consequences of that use.
#--------------------------------------------------------------------------------------

# The name of the program you're building, usually the file which contains main().
# The name without its extension (.c or .cpp or whatever) must be given here.
TARGET = WSN_main

# A list of the source (.c, .cc, .cpp) files in the project, including $(TARGET). Files
# in library subdirectories do not go in this list; they're automatically in LIB_OBJS
SRC = $(TARGET).cpp i2c_master.cpp MPU6050_driver.cpp task_data_acquisition.cpp \
                    BMP085_driver.cpp task_com_port.cpp task_bitcloud_node.cpp \
                                        task_bitcloud_base_station.cpp


LDFLAGS += -Wl,--gc-sections
LDFLAGS += -mmcu=atmega128rfa1

DEFINES += \
  -DPHY_ATMEGA128RFA1 \
  -DHAL_ATMEGA128RFA1 \
  -DPLATFORM_RCB128RFA1 \
#  -DF_CPU=8000000

# Clock frequency of the CPU, in Hz. This number should be an unsigned long integer.
# For example, 16 MHz would be represented as 16000000UL.
F_CPU = 16000000UL

# These codes are used to switch on debugging modes if they're being used. Several can
# be placed on the same line together to activate multiple debugging tricks at once.
# -DSERIAL_DEBUG       For general debugging through a serial device
# -DTRANSITION_TRACE   For printing state transition traces on a serial device
# -DTASK_PROFILE       For doing profiling, measurement of how long tasks take to run
# -DUSE_HEX_DUMPS      Include functions for printing hex-formatted memory dumps
OTHERS = -DSERIAL_DEBUG

# If the code -DTASK_SETUP_AND_LOOP is specified, ME405/FreeRTOS tasks classes will be
# required to provide methods setup() and loop(). Otherwise, they must only provide a
# a method called run() which is called just once by the scheduler.
OTHERS +=

# Other codes, used for turning on special features in a given project, can be put here
# -DSWOOP_BOARD        Tells the nRF24L01 radio driver to set up for the Swoop 1 board
# -DME405_BOARD_V05    Sets up radio driver for old ME405 board with 1 motor driver
# -DME405_BOARD_V06    Sets up radio driver for new ME405 board with 2 motor drivers
# -DME405_BREADBOARD   Sets up radio driver for ATmegaXX 40-pin on breadboard
# -DPOLYDAQ_BOARD      Sets up radio and other stuff for a PolyDAQ board
OTHERS +=

# This define is used to choose the type of programmer from the following options:
# bsd        - Parallel port in-system (ISP) programmer using SPI interface on AVR
# jtagice    - Serial or USB interface JTAG-ICE mk I clone from ETT or Olimex
# bootloader - Resident program in the AVR which downloads through USB/serial port
PROG = usbtiny

# These defines specify the ports to which the downloader device is connected.
# PPORT is for "bsd" on a parallel port, lpt1 on Windows or /dev/parport0 on Linux.
# JPORT is for "jtagice" on a serial port such as com1 or /dev/ttyS0, or usb-serial
#       such as com4 or /dev/ttyUSB1, or aliased serial port such as /dev/avrjtag
# BPORT is for "bootloader", the USB/serial port program downloader on the AVR
# The usbtiny programmer doesn't need a port specification; it has a USB identifier
PPORT = /dev/parport0
JPORT = /dev/ttyUSB1
BPORT = /dev/ttyUSB0

#-------------------------------------------------------------------------------------
# This section specifies the type of CPU; uncomment one line for your processor. To add
# a new chip to the file, put its designation here and also set fuse bytes below.
MCU = atmega128rfa1

#####################################################################################
################ End of the stuff the user is expected to need to change #############

# This is the name of the library file which will hold object code which has been
# compiled from all the source files in the library subdirectories
LIB_NAME = WSN.a

# A list of directories in which source files (*.cpp, *.c) and headers (.h) for the
# library are kept
LIB_DIRS = lib/freertos lib/frtcpp lib/misc lib/radio lib/sensors lib/sd_card \
           lib/serial lib/wsn

#-------------------------------------------------------------------------------------
# In this section, default settings for fuse bytes are given for each processor which
# this makefile supports. New chip specifications can be added to this file as needed.

# ATmega128RFa1 being tested on the SparkFun board
EFUSE = 0xFE
HFUSE = 0xD1
LFUSE = 0xEF


#-------------------------------------------------------------------------------------
# Tell the compiler how hard to try to optimize the code. Optimization levels are:
# -O0  Don't try to optimize anything (even leaves empty delay loops in)
# -O1  Some optimizations; code usually smaller and faster than O0
# -O2  Pretty high level of optimization; often good compromise of speed and size
# -O3  Tries really hard to make code run fast, even if code size gets pretty big
# -Os  Tries to make code size small. Sometimes -O1 makes it smaller, though(?!)
OPTIM = -O2

# Warnings which need to be given
C_WARNINGS = -Wall -Wextra -Wshadow -Wpointer-arith -Wbad-function-cast -Wcast-align \
             -Wsign-compare -Wstrict-prototypes -Wmissing-prototypes -Wunused \
             -Wmissing-declarations -Waggregate-return

CPP_WARNINGS = -Wall -Wextra -Wshadow -Wpointer-arith -Wcast-align -Wsign-compare \
               -Wmissing-declarations -Wunused

C_FLAGS = -D GCC_MEGA_AVR -D F_CPU=$(F_CPU) -D _GNU_SOURCE -DPHY_ATMEGA128RFA1 -DPLATFORM_RCB128RFA1 \
          -fsigned-char -funsigned-bitfields -fpack-struct -fshort-enums \
          -std=gnu99 -g $(OPTIM) -mmcu=$(MCU) $(OTHERS) $(C_WARNINGS) \
          $(patsubst %,-I%,$(LIB_DIRS))
```

```
CPP_FLAGS = -D GCC_MEGA_AVR -D F_CPU=$(F_CPU) -D _GNU_SOURCE \
            -fsigned-char -funsigned-bitfields -fshort-enums \
            -g $(OPTIM) -mmcu=$(MCU) $(OTHERS) $(CPP_WARNINGS) \
            $(patsubst %,-I%,$(LIB_DIRS))

# This section makes a list of object files from the source files in the SRC list,
# separating the C++ source files, the C source files, and assembly source files
OBJS = $(patsubst %.cpp, %.o, $(filter %.cpp, $(SRC))) \
       $(patsubst %.c, %.o, $(filter %.c, $(SRC))) \
       $(ASRC:.S=.o)

# This section makes a list of object files from the source files in subdirectories
# in the LIB_DIRS list, separating the C++, C, and assembly source files
LIB_SRC =  $(foreach A_DIR, $(LIB_DIRS), $(wildcard $(A_DIR)/*.cpp)) \
           $(foreach A_DIR, $(LIB_DIRS), $(wildcard $(A_DIR)/*.cc)) \
           $(foreach A_DIR, $(LIB_DIRS), $(wildcard $(A_DIR)/*.c)) \
           $(foreach A_DIR, $(LIB_DIRS), $(wildcard $(A_DIR)/*.S))

LIB_OBJS = $(patsubst %.cpp, %.o, $(filter %.cpp, $(LIB_SRC))) \
           $(patsubst %.cc, %.o, $(filter %.cc, $(LIB_SRC))) \
           $(patsubst %.c, %.o, $(filter %.c, $(LIB_SRC))) \
           $(LIB_ASRC:.S=.o)

#-------------------------------------------------------------------------------------
# Inference rules show how to process each kind of file.

# How to compile a .c file into a .o file
.c.o:
        @echo $<
        @avr-gcc $(DEFINES) -c $(C_FLAGS) $< -o $@

# How to compile a .cc file into a .o file
.cc.o:
        @echo $<
        @avr-gcc $(DEFINES) -c $(CPP_FLAGS) $< -o $@

# How to compile a .cpp file into a .o file
.cpp.o:
        @echo $<
        @avr-gcc $(DEFINES) -c $(CPP_FLAGS) $< -o $@

#-------------------------------------------------------------------------------------
# Make the main target of this project.  This target is invoked when the user types
# 'make' as opposed to 'make <target>.'  This must be the first target in Makefile.

all:  $(TARGET).hex

node: DEFINES += -DNODE

base: DEFINES += -DBASE

node base: install

#-------------------------------------------------------------------------------------
# This rule creates a .hex format downloadable file. A raw binary file which can be
# used by some bootloaders can be created; a listing file is also created.

$(TARGET).hex:  $(TARGET).elf
        @avr-objdump -h -S $(TARGET).elf > $(TARGET).lst
        @avr-objcopy -j .text -j .data -O ihex $(TARGET).elf $(TARGET).hex
        @avr-size $(TARGET).elf

#-------------------------------------------------------------------------------------
# This rule controls the linking of the target program from object files. The target
# is saved as an ELF debuggable binary.

$(TARGET).elf:  library $(OBJS)
        avr-gcc $(OBJS) $(LIB_NAME) -g -lm -mmcu=$(MCU) -o $(TARGET).elf
```

```
#---------------------------------------------------------------------------------------
# This is a dummy target that doesn't do anything. It's included because the author
# belongs to a faculty labor union and has been instilled with reverence for laziness.

nothing:

#---------------------------------------------------------------------------------------
# 'make install' will make the project, then download the program using whichever
# method has been selected -- ISP cable, JTAG-ICE module, or USB/serial bootloader

install:  $(TARGET).hex
  ifeq ($(PROG), usbtiny)
        avrdude -p $(MCU) -c usbtiny -V -B 1 -Uflash:w:$(TARGET).hex
  else
        @echo "ERROR: No programmer" $(PROG) "in the Makefile"
  endif

#---------------------------------------------------------------------------------------
# 'make fuses' will set up the processor's fuse bits in a "standard" mode. Standard is
# a setup in which there is no bootloader but the ISP and JTAG interfaces are enabled.

fuses: nothing
  ifeq ($(PROG), usbtiny)
        avrdude -p $(MCU) -c usbtiny -q -V noreset -Ulfuse:w:$(LFUSE):m
        avrdude -p $(MCU) -c usbtiny -q -V noreset -Uhfuse:w:$(HFUSE):m
        avrdude -p $(MCU) -c usbtiny -q -V noreset -Uefuse:w:$(EFUSE):m
  else
        @echo "ERROR: Only bsd or USBtiny set to program fuse bytes in this Makefile"
  endif

#---------------------------------------------------------------------------------------
# 'make readfuses' will see what the fuses currently hold

readfuses: nothing
  ifeq ($(PROG), bsd)
        @echo "ERROR: Not yet programmed to read fuses with bsd/ISP cable"
  else ifeq ($(PROG), jtagice)
        @avarice -e -j $(JPORT) --read-fuses
  else ifeq ($(PROG), bootloader)
        @echo "ERROR: Not yet programmed to read fuses via bootloader"
  else
        @echo "ERROR: No known device specified to read fuses"
  endif

#------------------------------------------------------------------------------
# 'make reset' will read a byte of lock bits, ignore it, and reset the chip

reset:
  ifeq ($(PROG), usbtiny)
        avrdude -p $(MCU) -c usbtiny -q -V -Ulfuse:r:/dev/null:r
  else
        @echo "ERROR: make reset only works with parallel ISP cable"
  endif

#---------------------------------------------------------------------------------------
# 'make doc' will use Doxygen to create documentation for the project. 'make libdoc'
# will do the same for the subdirectories which include ME405 library files.

.PHONY: doc libdoc
doc:
        @doxygen doxygen.conf

libdoc:
        @doxygen doxy_lib.conf

#---------------------------------------------------------------------------------------
# 'make clean' will erase the compiled files, listing files, etc. so you can restart
# the building process from a clean slate. It's also useful before committing files to
# a CVS or SVN or Git repository.

```

```
clean:
        rm -f $(LIB_NAME) *.o *.hex *.lst *.elf *~
        rm -fr doc
        for subdir in $(LIB_DIRS); do \
                rm -f $$subdir/*.o; \
                rm -f $$subdir/*.lst; \
                rm -f $$subdir/*~; \
        done

#--------------------------------------------------------------------------------------
# 'make library' will build the library file, using an automatically generated list of
# all the C, C++, and assembly source files in the library directories in LIB_DIRS

library: $(LIB_OBJS)
        @avr-ar -r $(LIB_NAME) $(LIB_OBJS)

#--------------------------------------------------------------------------------------
# 'make help' will show a list of things this makefile can do

help:
        @echo 'make          - Build program file ready to download'
        @echo 'make install  - Build program and download with parallel ISP cable'
        @echo 'make reset    - Reset processor with parallel cable RESET line'
        @echo 'make doc      - Generate documentation with Doxygen'
        @echo 'make clean    - Remove compiled files from all directories'
        @echo ' '
        @echo 'Notes: 1. Other less commonly used targets are in the Makefile'
        @echo '       2. You can combine targets, as in "make clean all"'
```

```
//**************************************************************************************
/** \file MPU6050_driver.h
 *   This file contains driver for the Invensense MPU6050 Accelerometer/Gyroscope
 *
 *  Revisions:
 *    \li 02-11-2013 HV Created file and version 1.0 of driver
 *    \li 03-01-2013 HV Made child class of IMUfilter; created conversion methods
 *    \li 06-03-2013 HV Removed all conversion code
 *
 *  License:
 *    This file is copyright 2013 by HA Vierra and released under the Lesser GNU
 *    Public License, version 2. It intended for educational use only, but its use
 *    is not limited thereto. */
/*    THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
 *    AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 *    IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 *    ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
 *    LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUEN-
 *    TIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 *    OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
 *    CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
 *    OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
 *    OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. */
//**************************************************************************************

// This define prevents this .H file from being included multiple times in a .CPP file
#ifndef _MPU6050_DRIVER_H_
#define _MPU6050_DRIVER_H_

//-------------------------------------Libraries-------------------------------------//
#include <stdlib.h>                                                     // Include standard library header files
#include <avr/io.h>
#include "emstream.h"                       // Header for serial ports and devices
#include "FreeRTOS.h"                       // Header for the FreeRTOS RTOS
#include "queue.h"                          // Header for FreeRTOS queues
#include "rs232int.h"                       // Include header for serial port class
#include "frt_text_queue.h"
#include "shares.h"
#include "i2c_master.h"

//--------------------------------Register Addresses---------------------------------//
#define MPU6050_ADDRESS 0x68 // Address with end write bit; 7-bit: 0x68
#define MPU6050_RA_XG_OFFS_TC 0x00 //[7] PWR_MODE, [6:1] XG_OFFS_TC, [0] OTP_BNK_VLD
#define MPU6050_RA_YG_OFFS_TC 0x01 //[7] PWR_MODE, [6:1] YG_OFFS_TC, [0] OTP_BNK_VLD
#define MPU6050_RA_ZG_OFFS_TC 0x02 //[7] PWR_MODE, [6:1] ZG_OFFS_TC, [0] OTP_BNK_VLD
#define MPU6050_RA_X_FINE_GAIN 0x03 //[7:0] X_FINE_GAIN
#define MPU6050_RA_Y_FINE_GAIN 0x04 //[7:0] Y_FINE_GAIN
#define MPU6050_RA_Z_FINE_GAIN 0x05 //[7:0] Z_FINE_GAIN
#define MPU6050_RA_XA_OFFS_H 0x06 //[15:0] XA_OFFS
#define MPU6050_RA_XA_OFFS_L_TC 0x07
#define MPU6050_RA_YA_OFFS_H 0x08 //[15:0] YA_OFFS
#define MPU6050_RA_YA_OFFS_L_TC 0x09
#define MPU6050_RA_ZA_OFFS_H 0x0A //[15:0] ZA_OFFS
#define MPU6050_RA_ZA_OFFS_L_TC 0x0B
#define MPU6050_RA_XG_OFFS_USRH 0x13 //[15:0] XG_OFFS_USR
#define MPU6050_RA_XG_OFFS_USRL 0x14
#define MPU6050_RA_YG_OFFS_USRH 0x15 //[15:0] YG_OFFS_USR
#define MPU6050_RA_YG_OFFS_USRL 0x16
#define MPU6050_RA_ZG_OFFS_USRH 0x17 //[15:0] ZG_OFFS_USR
#define MPU6050_RA_ZG_OFFS_USRL 0x18
#define MPU6050_RA_SMPLRT_DIV 0x19
#define MPU6050_RA_CONFIG 0x1A
#define MPU6050_RA_GYRO_CONFIG 0x1B
#define MPU6050_RA_ACCEL_CONFIG 0x1C
#define MPU6050_RA_FF_THR 0x1D
#define MPU6050_RA_FF_DUR 0x1E
#define MPU6050_RA_MOT_THR 0x1F
#define MPU6050_RA_MOT_DUR 0x20
#define MPU6050_RA_ZRMOT_THR 0x21
```

```
#define MPU6050_RA_ZRMOT_DUR 0x22
#define MPU6050_RA_FIFO_EN 0x23
#define MPU6050_RA_I2C_MST_CTRL 0x24
#define MPU6050_RA_I2C_SLV0_ADDR 0x25
#define MPU6050_RA_I2C_SLV0_REG 0x26
#define MPU6050_RA_I2C_SLV0_CTRL 0x27
#define MPU6050_RA_I2C_SLV1_ADDR 0x28
#define MPU6050_RA_I2C_SLV1_REG 0x29
#define MPU6050_RA_I2C_SLV1_CTRL 0x2A
#define MPU6050_RA_I2C_SLV2_ADDR 0x2B
#define MPU6050_RA_I2C_SLV2_REG 0x2C
#define MPU6050_RA_I2C_SLV2_CTRL 0x2D
#define MPU6050_RA_I2C_SLV3_ADDR 0x2E
#define MPU6050_RA_I2C_SLV3_REG 0x2F
#define MPU6050_RA_I2C_SLV3_CTRL 0x30
#define MPU6050_RA_I2C_SLV4_ADDR 0x31
#define MPU6050_RA_I2C_SLV4_REG 0x32
#define MPU6050_RA_I2C_SLV4_DO 0x33
#define MPU6050_RA_I2C_SLV4_CTRL 0x34
#define MPU6050_RA_I2C_SLV4_DI 0x35
#define MPU6050_RA_I2C_MST_STATUS 0x36
#define MPU6050_RA_INT_PIN_CFG 0x37
#define MPU6050_RA_INT_ENABLE 0x38
#define MPU6050_RA_DMP_INT_STATUS 0x39
#define MPU6050_RA_INT_STATUS 0x3A
#define MPU6050_RA_ACCEL_XOUT_H 0x3B
#define MPU6050_RA_ACCEL_XOUT_L 0x3C
#define MPU6050_RA_ACCEL_YOUT_H 0x3D
#define MPU6050_RA_ACCEL_YOUT_L 0x3E
#define MPU6050_RA_ACCEL_ZOUT_H 0x3F
#define MPU6050_RA_ACCEL_ZOUT_L 0x40
#define MPU6050_RA_TEMP_OUT_H 0x41
#define MPU6050_RA_TEMP_OUT_L 0x42
#define MPU6050_RA_GYRO_XOUT_H 0x43
#define MPU6050_RA_GYRO_XOUT_L 0x44
#define MPU6050_RA_GYRO_YOUT_H 0x45
#define MPU6050_RA_GYRO_YOUT_L 0x46
#define MPU6050_RA_GYRO_ZOUT_H 0x47
#define MPU6050_RA_GYRO_ZOUT_L 0x48
#define MPU6050_RA_EXT_SENS_DATA_00 0x49
#define MPU6050_RA_EXT_SENS_DATA_01 0x4A
#define MPU6050_RA_EXT_SENS_DATA_02 0x4B
#define MPU6050_RA_EXT_SENS_DATA_03 0x4C
#define MPU6050_RA_EXT_SENS_DATA_04 0x4D
#define MPU6050_RA_EXT_SENS_DATA_05 0x4E
#define MPU6050_RA_EXT_SENS_DATA_06 0x4F
#define MPU6050_RA_EXT_SENS_DATA_07 0x50
#define MPU6050_RA_EXT_SENS_DATA_08 0x51
#define MPU6050_RA_EXT_SENS_DATA_09 0x52
#define MPU6050_RA_EXT_SENS_DATA_10 0x53
#define MPU6050_RA_EXT_SENS_DATA_11 0x54
#define MPU6050_RA_EXT_SENS_DATA_12 0x55
#define MPU6050_RA_EXT_SENS_DATA_13 0x56
#define MPU6050_RA_EXT_SENS_DATA_14 0x57
#define MPU6050_RA_EXT_SENS_DATA_15 0x58
#define MPU6050_RA_EXT_SENS_DATA_16 0x59
#define MPU6050_RA_EXT_SENS_DATA_17 0x5A
#define MPU6050_RA_EXT_SENS_DATA_18 0x5B
#define MPU6050_RA_EXT_SENS_DATA_19 0x5C
#define MPU6050_RA_EXT_SENS_DATA_20 0x5D
#define MPU6050_RA_EXT_SENS_DATA_21 0x5E
#define MPU6050_RA_EXT_SENS_DATA_22 0x5F
#define MPU6050_RA_EXT_SENS_DATA_23 0x60
#define MPU6050_RA_MOT_DETECT_STATUS 0x61
#define MPU6050_RA_I2C_SLV0_DO 0x63
#define MPU6050_RA_I2C_SLV1_DO 0x64
#define MPU6050_RA_I2C_SLV2_DO 0x65
#define MPU6050_RA_I2C_SLV3_DO 0x66
#define MPU6050_RA_I2C_MST_DELAY_CTRL 0x67
#define MPU6050_RA_SIGNAL_PATH_RESET 0x68
```

```
#define MPU6050_RA_MOT_DETECT_CTRL 0x69
#define MPU6050_RA_USER_CTRL 0x6A
#define MPU6050_RA_PWR_MGMT_1 0x6B
#define MPU6050_RA_PWR_MGMT_2 0x6C
#define MPU6050_RA_BANK_SEL 0x6D
#define MPU6050_RA_MEM_START_ADDR 0x6E
#define MPU6050_RA_MEM_R_W 0x6F
#define MPU6050_RA_DMP_CFG_1 0x70
#define MPU6050_RA_DMP_CFG_2 0x71
#define MPU6050_RA_FIFO_COUNTH 0x72
#define MPU6050_RA_FIFO_COUNTL 0x73
#define MPU6050_RA_FIFO_R_W 0x74
#define MPU6050_RA_WHO_AM_I 0x75

//-------------------------------------------------------------------------------------
/** \brief This class makes an MPU6050 Accelerometer and Gyroscope sensor object.
 *  \details This is a MPU6050 class which programs and gets readings from an MPU6050
 *           Accelerometer/Gyroscope sensor.
 */

class MPU6050_driver : public i2c_master
{
    protected:
                emstream* ptr_to_serial;

    public:
                MPU6050_driver(emstream*);

//--------------------------------Method Prototypes----------------------------------//
                void Setup(void);
                void Test_I2C(void);
                bool Check_Registers(void);
                void Get_Accel_Values(void);
                void Get_Gyro_Rates(void);
//-------------------------------------Globals---------------------------------------//
                uint8_t ACCEL_XOUT_L;
                uint8_t ACCEL_XOUT_H;
                uint8_t ACCEL_YOUT_L;
                uint8_t ACCEL_YOUT_H;
                uint8_t ACCEL_ZOUT_L;
                uint8_t ACCEL_ZOUT_H;
                int16_t ACCEL_XOUT;
                int16_t ACCEL_YOUT;
                int16_t ACCEL_ZOUT;
                int16_t ACCEL_XOUT_OFFSET;
                int16_t ACCEL_YOUT_OFFSET;
                int16_t ACCEL_ZOUT_OFFSET;
                uint8_t GYRO_XOUT_L;
                uint8_t GYRO_XOUT_H;
                uint8_t GYRO_YOUT_L;
                uint8_t GYRO_YOUT_H;
                uint8_t GYRO_ZOUT_L;
                uint8_t GYRO_ZOUT_H;
                int16_t GYRO_XOUT;
                int16_t GYRO_YOUT;
                int16_t GYRO_ZOUT;
                int16_t GYRO_XOUT_OFFSET;
                int16_t GYRO_YOUT_OFFSET;
                int16_t GYRO_ZOUT_OFFSET;
};

#endif // _MPU6050_DRIVER_H_
```

```cpp
//**************************************************************************************
/** \file MPU6050_driver.cpp
 *   This file contains driver class for the Invensense MPU6050 Accelerometer/Gyroscope
 *
 *  Revisions:
 *    \li 02-11-2013 HV Created file and version 1.0 of driver
 *    \li 03-01-2013 HV Made child class of IMUfilter; created conversion methods
 *    \li 06-03-2013 HV Removed all conversion code
 *
 *  License:
 *    This file is copyright 2013 by HA Vierra and released under the Lesser GNU
 *    Public License, version 2. It intended for educational use only, but its use
 *    is not limited thereto. */
/*    THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
 *    AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 *    IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 *    ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
 *    LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUEN-
 *    TIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 *    OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
 *    CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
 *    OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
 *    OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. */
//**************************************************************************************

#include "MPU6050_driver.h"                        // Header for this class
#include "shares.h"
#include <avr/interrupt.h>
#include <util/delay.h>
#include <math.h>
#include <stdio.h>

//-------------------------------------------------------------------------------------
/** This constructor creates an I2C MPU6050 sensor driver object.
 *  @param p_debug_port A serial port, often RS-232, for debugging text (default: NULL)
 */
MPU6050_driver::MPU6050_driver (emstream* p_debug_port)
    :i2c_master(p_debug_port)
{
        //--------setup the MPU6050---------//
        *p_serial<<endl<<
        "//------------------------MPU6050 Setup---------------------------//"<<endl;
        //sensor_config_error = false;
        Test_I2C();
        Setup();
        Check_Registers();
        *p_serial<<
        "//--------------------MPU6050 Setup Complete----------------------//"<<endl<<endl;
        //hard-coded offsets
        ACCEL_XOUT_OFFSET = -948;
        ACCEL_YOUT_OFFSET = -154;
        ACCEL_ZOUT_OFFSET = 15761;
        GYRO_XOUT_OFFSET  = -44;
        GYRO_YOUT_OFFSET  = 9;
        GYRO_ZOUT_OFFSET  = -48;
}

//-------------------------------------------------------------------------------------
/** This function writes several commands to the internal configuration registers
 *  within the MPU6050. Here, the sensitivity, data rate, filtering cuttoff, etc. can
 *  be changed.
 */
void MPU6050_driver::Setup()
{
        //--------------Config Registers for MPU6050--------------------//
        //Sets sample rate to 1000/(1+1) = 500Hz // 0x09=>100Hz
        I2C_write(MPU6050_ADDRESS, MPU6050_RA_SMPLRT_DIV,0x09);//0x09);//0x01);
        //Disable FSync, 5Hz DLPF for accel
        I2C_write(MPU6050_ADDRESS, MPU6050_RA_CONFIG, 0x06);
        //Disable gyro self tests, scale of 1000 degrees/s; 65.5 LSB/deg/s
        I2C_write(MPU6050_ADDRESS, MPU6050_RA_GYRO_CONFIG, 0x10);
        //Disable accel self tests, scale of +-2g, no DHPF; 0.06 mg/LSB
        I2C_write(MPU6050_ADDRESS, MPU6050_RA_ACCEL_CONFIG, 0x00);

        I2C_write(MPU6050_ADDRESS, MPU6050_RA_SIGNAL_PATH_RESET, 0x00);
        //Motion detection control
        I2C_write(MPU6050_ADDRESS, MPU6050_RA_MOT_DETECT_CTRL, 0x00);
        //Disables FIFO, AUX I2C, FIFO and I2C reset bits to 0
        I2C_write(MPU6050_ADDRESS, MPU6050_RA_USER_CTRL, 0x00);
        //Sets clock source to gyro reference w/ PLL
        I2C_write(MPU6050_ADDRESS, MPU6050_RA_PWR_MGMT_1, 0x00);
        //Controls frequency of wakeups in accel low power mode plus the standby modes
        I2C_write(MPU6050_ADDRESS, MPU6050_RA_PWR_MGMT_2, 0x00);

        *p_serial<<("MPU6050 Setup Complete")<<endl;
}

//--------------------------------------------------------------------------------
/** This function tests the I2C bus for the MPU6050's address
 */
void MPU6050_driver::Test_I2C()
{
        uint8_t Data = 0x00;
        I2C_read(MPU6050_ADDRESS, MPU6050_RA_WHO_AM_I, &Data);

        if(Data == 0x68)
        {
                *p_serial<<("I2C Read Test Passed, MPU6050 Address: 0x")<<hex<<Data<<endl;
        }
        else
        {
                *p_serial<<("ERROR: I2C Read Test Failed.")<<endl;
                //while(1){};
        }
}

//--------------------------------------------------------------------------------
/** This function reads each of the configuration registers which are set in Setup()
 *  and compares their read values with the expected values to check for write errors.
 */
bool MPU6050_driver::Check_Registers()
{
        uint8_t Data = 0x00;
        uint8_t Failed = 0;

        I2C_read(MPU6050_ADDRESS, MPU6050_RA_SMPLRT_DIV, &Data);
        if(Data != 0x09) { *p_serial
        <<("Register check 1 failed, value should be 0x09, was: 0x")
        <<hex<<Data<<endl; Failed = 1; }
        I2C_read(MPU6050_ADDRESS, MPU6050_RA_CONFIG, &Data);
        if(Data != 0x06) { *p_serial
        <<("Register check 2 failed, value should be 0x06, was: 0x")
        <<hex<<Data<<endl; Failed = 1; }
        I2C_read(MPU6050_ADDRESS, MPU6050_RA_GYRO_CONFIG, &Data);
        if(Data != 0x10) { *p_serial
        <<("Register check 3 failed, value should be 0x10, was: 0x")
        <<hex<<Data<<endl; Failed = 1; }
        I2C_read(MPU6050_ADDRESS, MPU6050_RA_ACCEL_CONFIG, &Data);
        if(Data != 0x00) { *p_serial
        <<("Register check 4 failed, value should be 0x00, was: 0x")
        <<hex<<Data<<endl; Failed = 1; }
        I2C_read(MPU6050_ADDRESS, MPU6050_RA_INT_ENABLE, &Data);
        if(Data != 0x00) { *p_serial
        <<("Register check 5 failed, value should be 0x01, was: 0x")
        <<hex<<Data<<endl; Failed = 1; }

        I2C_read(MPU6050_ADDRESS, MPU6050_RA_SIGNAL_PATH_RESET, &Data);
        if(Data != 0x00) { *p_serial
        <<("Register check 6 failed, value should be 0x00, was: 0x")
        <<hex<<Data<<endl; Failed = 1; }
        I2C_read(MPU6050_ADDRESS, MPU6050_RA_MOT_DETECT_CTRL, &Data);
        if(Data != 0x00) { *p_serial
        <<("Register check 7 failed, value should be 0x00, was: 0x")
        <<hex<<Data<<endl; Failed = 1; }
        I2C_read(MPU6050_ADDRESS, MPU6050_RA_USER_CTRL, &Data);
        if(Data != 0x00) { *p_serial
        <<("Register 8 check failed, value should be 0x00, was: 0x")
        <<hex<<Data<<endl; Failed = 1; }
        I2C_read(MPU6050_ADDRESS, MPU6050_RA_PWR_MGMT_1, &Data);
        if(Data != 0x00) { *p_serial
        <<("Register check 9 failed, value should be 0x02, was: 0x")
        <<hex<<Data<<endl; Failed = 1; }
        I2C_read(MPU6050_ADDRESS, MPU6050_RA_PWR_MGMT_2, &Data);
        if(Data != 0x00) { *p_serial
        <<("Register check 10 failed, value should be 0x00, was: 0x")
        <<hex<<Data<<endl; Failed = 1; }


        if (Failed == 0)
        {
                *p_serial<<("Register value check passed")<<endl;
                return true;
        }
        else
        {
                *p_serial<<("Register value check failed")<<endl;
                return false;
        }
}

//-------------------------------------------------------------------------------------
/** This function Gets raw accelerometer data, performs no processing
 */
void MPU6050_driver::Get_Accel_Values()
{
        uint16_t one_g = 0;
        if(ACCEL_ZOUT_OFFSET != 0)//if calibration is ran, set the 1g offset
                one_g = 16384;

        I2C_read(MPU6050_ADDRESS, MPU6050_RA_ACCEL_XOUT_H, &ACCEL_XOUT_H);
        I2C_read(MPU6050_ADDRESS, MPU6050_RA_ACCEL_XOUT_L, &ACCEL_XOUT_L);
        I2C_read(MPU6050_ADDRESS, MPU6050_RA_ACCEL_YOUT_H, &ACCEL_YOUT_H);
        I2C_read(MPU6050_ADDRESS, MPU6050_RA_ACCEL_YOUT_L, &ACCEL_YOUT_L);
        I2C_read(MPU6050_ADDRESS, MPU6050_RA_ACCEL_ZOUT_H, &ACCEL_ZOUT_H);
        I2C_read(MPU6050_ADDRESS, MPU6050_RA_ACCEL_ZOUT_L, &ACCEL_ZOUT_L);

        ACCEL_XOUT = (uint16_t)((ACCEL_XOUT_H<<8)|ACCEL_XOUT_L) - ACCEL_XOUT_OFFSET;
        ACCEL_YOUT = (uint16_t)((ACCEL_YOUT_H<<8)|ACCEL_YOUT_L) - ACCEL_YOUT_OFFSET;
        ACCEL_ZOUT = (uint16_t)((ACCEL_ZOUT_H<<8)|ACCEL_ZOUT_L) - ACCEL_ZOUT_OFFSET+one_g;
}

//-------------------------------------------------------------------------------------
/** This function Gets raw gyroscope data, performs no processing
 */
void MPU6050_driver::Get_Gyro_Rates()
{
        I2C_read(MPU6050_ADDRESS, MPU6050_RA_GYRO_XOUT_H, &GYRO_XOUT_H);
        I2C_read(MPU6050_ADDRESS, MPU6050_RA_GYRO_XOUT_L, &GYRO_XOUT_L);
        I2C_read(MPU6050_ADDRESS, MPU6050_RA_GYRO_YOUT_H, &GYRO_YOUT_H);
        I2C_read(MPU6050_ADDRESS, MPU6050_RA_GYRO_YOUT_L, &GYRO_YOUT_L);
        I2C_read(MPU6050_ADDRESS, MPU6050_RA_GYRO_ZOUT_H, &GYRO_ZOUT_H);
        I2C_read(MPU6050_ADDRESS, MPU6050_RA_GYRO_ZOUT_L, &GYRO_ZOUT_L);

        GYRO_XOUT = (uint16_t)((GYRO_XOUT_H<<8)|GYRO_XOUT_L) - GYRO_XOUT_OFFSET;
        GYRO_YOUT = (uint16_t)((GYRO_YOUT_H<<8)|GYRO_YOUT_L) - GYRO_YOUT_OFFSET;
        GYRO_ZOUT = (uint16_t)((GYRO_ZOUT_H<<8)|GYRO_ZOUT_L) - GYRO_ZOUT_OFFSET;
}
```

```
//*************************************************************************************
/** \file shares.h
 *    This file contains extern declarations for queues and other inter-task data
 *    communication objects used in the Poly_Sense base station and node configs
 *
 *  Revisions:
 *    \li 04-20-2013 HAV Original file created
 *  License:
 *              This file is copyright 2013 by HA Vierra and released under the Lesser GNU
 *              Public License, version 2. It intended for educational use only, but its use
 *              is not limited thereto. */
/*              THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
 *              AND     ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 *              IMPLIED         WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 *              ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
 *              LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUEN-
 *              TIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 *              OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
 *              CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
 *              OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
 *              OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. */
//*************************************************************************************

// This define prevents this .h file from being included multiple times in a .cpp file
#ifndef _SHARES_H_
#define _SHARES_H_

#include "frt_queue.h"                      // Header of wrapper for FreeRTOS queues
#include "frt_text_queue.h"                 // Header for a "<<" queue class
#include "frt_shared_data.h"                // Header for thread-safe shared data
#include "rs232int.h"                       // ME405/507 library for serial comm.

#define PACKET_DATA_SIZE_BYTES 15

#define DATA_TYPE_FLOAT 'f'
#define DATA_TYPE_INT 'i'
#define DATA_TYPE_SHORT 's'

#define GET_CHAR 2
#define SER_PORT 1


void print_packet(uint8_t type);


//Packet for computer-base station communication
typedef struct serial_packet serial_packet;
struct serial_packet
{
        uint8_t flag;
        uint16_t node_id;
        uint8_t data[PACKET_DATA_SIZE_BYTES];
};

// Flags
enum flags
{
        SER_DATA_REQ = '1',
        SENSOR_CONFIG_REQ = '2',
        SENSOR_DATA = '3',
        SENSOR_CONFIG_ERROR = '6',
        DEBUG = '4'
};

//Global Command Packet
extern serial_packet command_pack;

// Semaphore to signal a packet is ready to be sent to node
extern frt_queue<bool>* send_packet;

```

```
// Semaphore to signal a sensor read
extern frt_queue<bool>* get_sensor_data;

// Pointer to char array holding the sensor's name
extern char sensor_name[PACKET_DATA_SIZE_BYTES];

// Pointer to char array holding the previous sensor's name
extern char last_sensor_name[PACKET_DATA_SIZE_BYTES];

// Pointer to char array telling a sensor what data to transmit.
extern char sensor_command[PACKET_DATA_SIZE_BYTES];

// State variable telling whether device is sending or receiving
extern bool transmit; // true = TX, false = RX

// State variable telling whether node has been configured for a particular sensor yet
extern bool node_configured; // true = TX, false = RX

#endif // _SHARES_H_
```

```c
#ifdef BASE
//*************************************************************************************
/** \file task_bitcloud_base_station.h
 *    This file contains the code for a task class which controls the brightness of an
 *    LED using a voltage measured from the A/D as input. The fun part: the brightness
 *    that is being controlled can be on another AVR computer, with signals being sent
 *    and received via wireless transceivers.
 *
 *  Revisions:
 *    \li 03-07-2013 HAV Original file created
 *
 *  License:
 *    This file is copyright 2013 by HA Vierra and released under the Lesser GNU
 *    Public License, version 2. It intended for educational use only, but its use
 *    is not limited thereto. */
/*    THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
 *    AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 *    IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 *    ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
 *    LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUEN-
 *    TIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 *    OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
 *    CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
 *    OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
 *    OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. */
//*************************************************************************************

// This define prevents this .h file from being included multiple times in a .cpp file
#ifndef _TASK_BITCLOUD_BASE_STATION_H_
#define _TASK_BITCLOUD_BASE_STATION_H_

#include <stdlib.h>                                                     // Prototype declarations for I/O functions
#include <avr/io.h>                                                     // Header for special function registers
#include "FreeRTOS.h"                                           // Primary header for FreeRTOS
#include "task.h"                                                       // Header for FreeRTOS task functions
#include "queue.h"                                                      // FreeRTOS inter-task communication queues
#include "frt_task.h"                                           // base task class library
#include "time_stamp.h"                                         // Class to implement a microsecond timer
#include "frt_queue.h"                                          // Header of wrapper for FreeRTOS queues
#include "frt_shared_data.h"                            // Header for thread-safe shared data
#include "rs232int.h"                                           // library for serial comm.

#include "config.h"
#include "hal.h"
#include "phy.h"
#include "sys.h"
#include "nwk.h"
#include "halSleep.h"
#include <util/delay.h>

// Base station must be a coordinator, and have an app_addr of 0
#define APP_CAPTION     "Coordinator"
#define APP_NODE_TYPE   0
#define APP_COORDINATOR 1
#define APP_ADDR        0x0000


typedef enum AppState_t
{
  APP_STATE_INITIAL,
  APP_STATE_SEND,
  APP_STATE_WAIT_CONF,
  APP_STATE_SENDING_DONE,
  APP_STATE_PREPARE_TO_SLEEP,
  APP_STATE_SLEEP,
  APP_STATE_WAKEUP,
  APP_STATE_RECEIEVE,
  APP_STATE_WAITING,
} AppState_t;

//-------------------------------------------------------------------------------------
/** \brief This class is used to facilitate wireless communication with all paired nodes
 *         and handle synchronization with task_com_port.
 */
class task_bitcloud_base_station : public frt_task
{
public:
        emstream* ser_port;

        // This constructor creates a generic task of which many copies can be made
        task_bitcloud_base_station (const char*, unsigned portBASE_TYPE, size_t, emstream*);

        // This method is called by the RTOS once to run the task loop for ever and ever.
        void run (void);

        //Sets the network configutaions for network communication
        void NWK_Config(void);

        //Sends the data
        void sendFrame(void);

        void APP_TaskHandler(void);
};

#endif // _TASK_BITCLOUD_BASE_STATION_H_
#endif // ifdef BASE
```

```
#ifdef BASE
//*************************************************************************************
/** \file task_bitcloud_base_station.cpp
 *        This file contains code for a task class which uses the onboard RF transciever
 *                of the ATmega128RFA1 to wirelessly communicate as coordintor device with the
 *                nodes within range. It also utilizes queue flags to synchronize task run/block
 *        time with task_com_port.
 *
 *  Revisions:
 *    \li 03-07-2013 HAV Original file created
 *
 *  License:
 *    This file is copyright 2013 by HA Vierra and released under the Lesser GNU
 *    Public License, version 2. It intended for educational use only, but its use
 *    is not limited thereto. */
/*    THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
 *    AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 *    IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 *    ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
 *    LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUEN-
 *    TIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 *    OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
 *    CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
 *    OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
 *    OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. */
//*************************************************************************************

#include "frt_text_queue.h"                  // Header for text queue class
#include "task_bitcloud_base_station.h"     // Header for this task
#include "shares.h"                         // Shared inter-task communications
#include "semphr.h"

//global serial port pointer
emstream* p_ser;

//Application State
AppState_t appState;

//Data request struct
NWK_DataReq_t nwkDataReq;

//Prototype for the callback function indicating if the base station has received data
bool appDataInd(NWK_DataInd_t *ind);

//Prototype for the callback function indicating if the base station sent
//the data successfully
void appDataConf(NWK_DataReq_t *req);

//Variable that indicates if the base station is done transmitting/receving data
bool transmission_done;


/** This callback function indicates that the base station has received wireless data
 *  from the node. It then copies the data from *ind into command_pack and changes
 *  appState to APP_STATE_RECEIVE.
 *  @param *ind The pointer to the payload data.
 *  @return true
 */
bool appDataInd(NWK_DataInd_t *ind)
{
        memcpy(&command_pack,(ind->data),sizeof(command_pack));
        appState = APP_STATE_RECEIEVE;
        return true;
}

/** This callback function indicates the confirmation status of the sent data. It
 *  checks req->status and changes the state to APP_STATE_SENDING_DONE if the
 *  transmission is successful or changes the state to APP_STATE_SEND to resend the
 *  data.
 *  @param *req The pointer to the confirmation status.
 */
void appDataConf(NWK_DataReq_t *req)
{
        if(NWK_SUCCESS_STATUS == req->status)
                appState = APP_STATE_SENDING_DONE;
        else
        {
                if(command_pack.flag == DEBUG)
                {
                        switch(req->status)
                        {
                                case NWK_ERROR_STATUS:
                                {
                                   *p_ser<<"NWK_ERROR_STATUS"<<endl;
                                } break;
                                case NWK_OUT_OF_MEMORY_STATUS:
                                {
                                   *p_ser<<"NWK_OUT_OF_MEMORY_STATUS"<<endl;
                                } break;
                                case  NWK_NO_ACK_STATUS:
                                {
                                   *p_ser<<"NWK_NO_ACK_STATUS"<<endl;
                                } break;
                                case NWK_PHY_CHANNEL_ACCESS_FAILURE_STATUS:
                                {
                                   *p_ser<<"NWK_PHY_CHANNEL_ACCESS_FAILURE_STATUS"<<endl;
                                } break;
                                case NWK_PHY_NO_ACK_STATUS:
                                {
                                   *p_ser<<"NWK_PHY_NO_ACK_STATUS"<<endl;
                                } break;

                                default:
                                        break;
                        }
                }
                 appState = APP_STATE_SEND;
        }
}


/** This functions sets the necessary network configutaions for network communication.
 */
void task_bitcloud_base_station::NWK_Config(void)
{
        //Sets the node network address
        NWK_SetAddr(APP_ADDR);

        //Sets the network identifier
        NWK_SetPanId(APP_PANID);

        //Sets frequency band
        PHY_SetChannel(APP_CHANNEL);

        //Sets the transceiver state
        PHY_SetRxState(true);

        //Registers endpoint
        NWK_OpenEndpoint(APP_ENDPOINT, appDataInd);

   if(transmit)
      appState = APP_STATE_SEND;
   else
      appState = APP_STATE_WAITING;
}

/** This functions sends the command_pack to corresponding node indicated
 *  by command_pack.node_id. It also changes the appState to APP_STATE_WAIT_CONF.
 */
void task_bitcloud_base_station::sendFrame(void)
```

```
{
      nwkDataReq.dstAddr = command_pack.node_id;
      nwkDataReq.dstEndpoint = APP_ENDPOINT;
      nwkDataReq.srcEndpoint = APP_ENDPOINT;
      nwkDataReq.options = NWK_OPT_ACK_REQUEST;
      nwkDataReq.data = (uint8_t *)&command_pack;
      nwkDataReq.size = sizeof(command_pack);
      nwkDataReq.confirm = appDataConf;
      NWK_DataReq(&nwkDataReq);
          appState = APP_STATE_WAIT_CONF;
}

/** This functions controls the state machine by checking appState and executing
 *  appropriate functions within each state.
 */
 void task_bitcloud_base_station::APP_TaskHandler(void)
{
  switch (appState)
  {
    case  APP_STATE_INITIAL:
    {
                if(command_pack.flag == DEBUG)
                {
                        *ser_port<<"Initializing"<<endl;
                }
      NWK_Config();
    } break;

    case APP_STATE_SEND:
    {
                //Blink LED
                PORTF |= (1<<PF7);
                //_delay_ms(250);
                PORTF &= ~(1<<PF7);
                if(command_pack.flag == DEBUG)
                {
                        *ser_port<<"Sending Data"<<endl;
                }
      sendFrame();
    } break;

    case APP_STATE_SENDING_DONE:
    {
                if(command_pack.flag == DEBUG)
                {
                        *ser_port<<"Data Sent from base station"<<endl;
                }
                // Put in waiting state to recieve node packet
                appState = APP_STATE_WAITING;
    } break;

    case APP_STATE_RECEIEVE:
    {
                if(command_pack.flag == DEBUG)
                {
                        //Prints the flag, node id, data from command_pack to the serial port.
         *ser_port<<"Received Node Packet"<<endl<<"Packet Contents: "<<endl;
                        *ser_port<<bin<<command_pack.flag<<endl<<command_pack.node_id<<endl;
                        for(uint8_t i =0; i < sizeof(command_pack.data); i++)
                                *ser_port<<bin<<command_pack.data[i]<<endl;
                }
                else
                {
                 //Sends each byte of data from command_pack using putchar()
                        for(uint8_t i =0; i < sizeof(serial_packet); i++)
                                ser_port->putchar(((uint8_t *)&command_pack)[i]);
                }
                transmission_done = true;
        } break;

        case APP_STATE_WAITING:
    {

        } break;

        default:
    break;
  }
}

//---------------------------------------------------------------------------------------
/** This constructor creates a task which handles the wireless transmission from the
 *  base station to the node as well as data from the node to the base station.
 *  @param a_priority The priority at which this task will initially run (default: 0)
 *  @param a_stack_size The size of this task's stack in bytes
 *                      (default: configMINIMAL_STACK_SIZE)
 *  @param p_ser_dev Pointer to a serial device (port, radio, SD card, etc.) which can
 *                   be used by this task to communicate (default: NULL)
 */

task_bitcloud_base_station::task_bitcloud_base_station (const char* a_name,
                                                                 unsigned portBASE_TYPE a_priority,
                                                                 size_t a_stack_size,
                                                                 emstream* p_ser_dev
                                                                )
        : frt_task (a_name, a_priority, a_stack_size, p_ser_dev)
{
        //Application State
        appState = APP_STATE_INITIAL;


        ser_port = p_ser_dev;
        p_ser = p_ser_dev;
}

//---------------------------------------------------------------------------------------
/** This method is called once by the RTOS scheduler. Each time around the for (;;)
 *  loop, it waits for a send packet flag to arrive in a queue, then initializes the
 *  boolean flag signaling a complete transmission as false. Then it executes the
 *  bitcloud state machines which operate the RF transciever to send and receive
 *  wireless data.
 */
void task_bitcloud_base_station::run (void)
{
        transmission_done = false;
        SYS_Init();

        for (;;)
        {
                transmit = send_packet ->get();
                transmission_done = false;

                //Initializes appState to APP_STATE_INITIAL
                appState = APP_STATE_INITIAL;

      //While loop continues until the node is done transmitting/receiving data
                while(!transmission_done)
                {
                        SYS_TaskHandler();
                        APP_TaskHandler();
                }
        }
}
#endif //ifdef BASE
```

```
#ifdef NODE
//*************************************************************************************
/** \file task_bitcloud_node.h
 *        This file contains code for a task class which uses the onboard RF transciever
 *                of the ATmega128RFA1 to wirelessly communicate as coordintor device with the
 *                base station. It also utilizes queue flags to synchronize task run/block
 *        time with task_data_acquisition.
 *
 *  Revisions:
 *    \li 03-07-2013 HAV Original file created
 *
 *  License:
 *    This file is copyright 2013 by HA Vierra and released under the Lesser GNU
 *    Public License, version 2. It intended for educational use only, but its use
 *    is not limited thereto. */
/*    THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
 *    AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 *    IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 *    ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
 *    LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUEN-
 *    TIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 *    OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
 *    CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
 *    OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
 *    OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. */
//*************************************************************************************

// This define prevents this .h file from being included multiple times in a .cpp file
#ifndef _TASK_BITCLOUD_NODE_H_
#define _TASK_BITCLOUD_NODE_H_

#include <stdlib.h>                                                     // Prototype declarations for I/O functions
#include <avr/io.h>                                                     // Header for special function registers
#include "FreeRTOS.h"                                           // Primary header for FreeRTOS
#include "task.h"                                                       // Header for FreeRTOS task functions
#include "queue.h"                                                      // FreeRTOS inter-task communication queues
#include "frt_task.h"                                           // ME405/507 base task class
#include "time_stamp.h"                                         // Class to implement a microsecond timer
#include "frt_queue.h"                                          // Header of wrapper for FreeRTOS queues
#include "frt_shared_data.h"                            // Header for thread-safe shared data
#include "rs232int.h"                                           // ME405/507 library for serial comm.

#include "config.h"
#include "hal.h"
#include "phy.h"
#include "sys.h"
#include "nwk.h"
#include "halSleep.h"
#include <util/delay.h>

// A node device must be a router to communication exclusievely with the base station
// and an address between 0x0001 and 0x8000
#define APP_ADDR         0x0002
#define APP_CAPTION     "Router"
#define APP_NODE_TYPE   1
#define APP_ROUTER      1


typedef enum AppState_t
{
  APP_STATE_INITIAL,
  APP_STATE_SEND,
  APP_STATE_WAIT_CONF,
  APP_STATE_SENDING_DONE,
  APP_STATE_PREPARE_TO_SLEEP,
  APP_STATE_SLEEP,
  APP_STATE_WAKEUP,
  APP_STATE_RECEIEVE,
  APP_STATE_WAITING,
} AppState_t;

//-------------------------------------------------------------------------------------
/** \brief This class is used to facilitate wireless communication with the base station
 *         and handle synchronization with task_data_acquisition.
 */
class task_bitcloud_node : public frt_task
{
private:
protected:

public:
        emstream* ser_port;

        // This constructor creates a generic task of which many copies can be made
        task_bitcloud_node (const char*, unsigned portBASE_TYPE, size_t, emstream*);

        // This method is called by the RTOS once to run the task loop for ever and ever.
        void run (void);

        //Sets the network configutaions for network communication
        void NWK_Config(void);

        //Sends the data
        void sendFrame(void);

        void APP_TaskHandler(void);
};

#endif // _TASK_BITCLOUD_NODE_H_
#endif // ifdef NODE
```

```
#ifdef NODE
//**************************************************************************************
/** \file task_bitcloud_node.cpp
 *
 *
 *  Revisions:
 *    \li 03-07-2013 HAV Original file created
 *
 *  License:
 *    This file is copyright 2013 by HA Vierra and released under the Lesser GNU
 *    Public License, version 2. It intended for educational use only, but its use
 *    is not limited thereto. */
/*    THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
 *    AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 *    IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 *    ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
 *    LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUEN-
 *    TIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 *    OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
 *    CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
 *    OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
 *    OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. */
//**************************************************************************************

#include "frt_text_queue.h"                 // Header for text queue class
#include "task_bitcloud_node.h"          // Header for this task
#include "shares.h"                         // Shared inter-task communications
#include "semphr.h"

//global serial port pointer
emstream* p_ser;

//Application State
AppState_t appState;

//Data request struct
NWK_DataReq_t nwkDataReq;

//Prototype for the callback function indicating if the node has received data
bool appDataInd(NWK_DataInd_t *ind);

//Prototype for the callback function indicating if the node sent the data successfully
void appDataConf(NWK_DataReq_t *req);

//Variable that indicates if the node is done transmitting/receving data
bool transmission_done;

/** This callback function indicates that the node has received wireless data from
 *  the base station. It then copies the data from *ind into command_pack and changes
 *  appState to APP_STATE_RECEIVE.
 *  @param *ind The pointer to the payload data.
 *  @return true
 */
bool appDataInd(NWK_DataInd_t *ind)
{
        memcpy(&command_pack,(ind->data),sizeof(command_pack));
        appState = APP_STATE_RECEIEVE;
        return true;
}

/** This callback function indicates the confirmation status of the sent data. It
 *  checks req->status and changes the state to APP_STATE_SENDING_DONE if the
 *  transmission is successful or changes the state to APP_STATE_SEND to resend the
 *  data.
 *  @param *req The pointer to the confirmation status.
 */
void appDataConf(NWK_DataReq_t *req)
{
        if(NWK_SUCCESS_STATUS == req->status)
                appState = APP_STATE_SENDING_DONE;
        else
        {
                switch(req->status)
                {
                        case NWK_ERROR_STATUS:
                        {
                                *p_ser<<"NWK_ERROR_STATUS"<<endl;
                        } break;
                        case NWK_OUT_OF_MEMORY_STATUS:
                        {
                                *p_ser<<"NWK_OUT_OF_MEMORY_STATUS"<<endl;
                        } break;
                        case  NWK_NO_ACK_STATUS:
                        {
                                *p_ser<<"NWK_NO_ACK_STATUS"<<endl;
                        } break;
                        case NWK_PHY_CHANNEL_ACCESS_FAILURE_STATUS:
                        {
                                *p_ser<<"NWK_PHY_CHANNEL_ACCESS_FAILURE_STATUS"<<endl;
                        } break;
                        case NWK_PHY_NO_ACK_STATUS:
                        {
                                *p_ser<<"NWK_PHY_NO_ACK_STATUS"<<endl;
                        } break;

                        default:
                        break;
                }
                 appState = APP_STATE_SEND;
        }
}

/** This functions sets the necessary network configutaions for network communication.
 */
void task_bitcloud_node::NWK_Config(void)
{
        //Sets the node network address
        NWK_SetAddr(APP_ADDR);
        //Sets the network identifier
        NWK_SetPanId(APP_PANID);
        //Sets frequency band
        PHY_SetChannel(APP_CHANNEL);
        //Sets the transceiver state
        PHY_SetRxState(true);
        //Registers endpoint
        NWK_OpenEndpoint(APP_ENDPOINT, appDataInd);
   if(transmit)
      appState = APP_STATE_SEND;
   else
      appState = APP_STATE_WAITING;
}

/** This functions sends the command_pack to the base station.
 *  It also changes the appState to APP_STATE_WAIT_CONF.
 */
void task_bitcloud_node::sendFrame(void)
{
        nwkDataReq.dstAddr = 0;// Always send to base station from node
        nwkDataReq.dstEndpoint = APP_ENDPOINT;
        nwkDataReq.srcEndpoint = APP_ENDPOINT;
        nwkDataReq.options = NWK_OPT_ACK_REQUEST;
        nwkDataReq.data = (uint8_t *)&command_pack;
        nwkDataReq.size = sizeof(command_pack);
        nwkDataReq.confirm = appDataConf;
        NWK_DataReq(&nwkDataReq);
        appState = APP_STATE_WAIT_CONF;

        *ser_port<<"Sending Packet"<<endl<<"Packet Contents: "<<endl;
        print_packet(SER_PORT);
}
```

```

/** This functions controls the state machine by checking appState and executing
 *  appropriate functions within each state.
 */
 void task_bitcloud_node::APP_TaskHandler(void)
{
        switch (appState)
        {
                case  APP_STATE_INITIAL:
                {
                        NWK_Config();
                } break;

                case APP_STATE_SEND:
                {
                        //Blink LED
                        PORTF |= (1<<PF7);
                        //_delay_ms(250);
                        PORTF &= ~(1<<PF7);
                        sendFrame();
                } break;

                case APP_STATE_SENDING_DONE:
                {
                        *ser_port<<"Packet Sent"<<endl;
                        send_packet->put(false);
                        transmission_done = true;
                } break;

                case APP_STATE_RECEIEVE:
                {
                        // Check the packet's flag
                        switch(command_pack.flag)
                        {
                                case SENSOR_CONFIG_REQ:
                                {
                                   *ser_port<<"Received Base Config Packet"<<endl<<"Packet Contents:"<<endl;
                                        print_packet(SER_PORT);

                                        memcpy(sensor_name, command_pack.data, sizeof(command_pack.data));

                                        *ser_port<<"Sensor Name: "<<sensor_name<<endl<<endl;


                                        // Set flag that node is configured to read from a sensor
                                        node_configured = true;

                                        // Signal ready to config sensor driver
                                        get_sensor_data->put(false);


                                        //send_packet->put(true);
                                        transmission_done = true;
                                        break;
                                }
                                case SENSOR_DATA:
                                {
               *ser_port<<"Received Data Request Packet"
               <<endl<<"Packet Contents:"<<endl;
                                        print_packet(SER_PORT);

                                        // Check if sensor name has been configured
                                        if(node_configured)
                                        {
                                                memcpy(sensor_command, command_pack.data, sizeof(command_pack.data));
                                                // Signal ready to read sensor
                                                get_sensor_data->put(true);
                                                transmission_done = true;
                                        }
                                        else
                                        {
                                                *ser_port<<"ERROR:Node not configured, cannot read sensor data"<<endl;
                                                appState = APP_STATE_WAITING;
                                        }
                                        break;
                                }
                                case DEBUG:
                                        break;

                                default:
                                        break;
                        }
                        // //transmission_done = true;

                } break;

                case APP_STATE_WAITING:
                {

                } break;

                default:
                break;
        }
}

//-------------------------------------------------------------------------------------
/** This constructor creates a task which handles the wireless transmission from the
 *  node to the base station as well as data from the base station to the node.
 *  @param a_name A character string which will be the name of this task
 *  @param a_priority The priority at which this task will initially run (default: 0)
 *  @param a_stack_size The size of this task's stack in bytes
 *                      (default: configMINIMAL_STACK_SIZE)
 *  @param p_ser_dev Pointer to a serial device (port, radio, SD card, etc.) which can
 *                   be used by this task to communicate (default: NULL)
 */

task_bitcloud_node::task_bitcloud_node (const char* a_name,
                                                                 unsigned portBASE_TYPE a_priority,
                                                                 size_t a_stack_size,
                                                                 emstream* p_ser_dev
                                                                )
        : frt_task (a_name, a_priority, a_stack_size, p_ser_dev)
{
        //Application State
        appState = APP_STATE_INITIAL;
        node_configured = false;
        transmit = false;
        ser_port = p_ser_dev;
        p_ser = p_ser_dev;

}

//-------------------------------------------------------------------------------------
/** This method is called once by the RTOS scheduler. Each time around the for (;;)
 *  loop, it waits for a send packet flag to arrive in a queue, then initializes the
 *  boolean flag signaling a complete transmission as false. Then it executes the
 *  bitcloud state machines which operate the RF transciever to send and receive
 *  wireless data.
 */
void task_bitcloud_node::run (void)
{
        *ser_port<<"Starting Bitcloud Task: Node"<<endl;

        SYS_Init();

        for (;;)
        {
                // wait for queue and check for RX(false)/TX(true) mode
                transmit = send_packet->get();
                if(transmit)
                        *ser_port<<"Transmitting..."<<endl;
                else
                        *ser_port<<"Receiving..."<<endl;

                transmission_done = false;

      //Initializes appState to APP_STATE_INITIAL
                appState = APP_STATE_INITIAL;

      //While loop continues until the node is done transmitting/receiving data
                while(!transmission_done)
                {
                        SYS_TaskHandler();
                        APP_TaskHandler();
                }
        }
}
#endif //ifdef NODE
```

```
#ifdef BASE
//**************************************************************************************
/** \file task_com_port.h
 *    This is the header file for a task class which handles communciation with
 *        a host computer via UART/USB. This task is responsible for coordinating data
 *    packets between the host computer and task_bitcloud_node through the use of
 *    flags on the packet.
 *
 *  Revisions:
 *    \li 04-11-2013 HAV Original file created
 *
 *  License:
 *    This file is copyright 2013 by HA Vierra and released under the Lesser GNU
 *    Public License, version 2. It intended for educational use only, but its use
 *    is not limited thereto. */
/*    THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
 *    AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 *    IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 *    ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
 *    LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUEN-
 *    TIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 *    OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
 *    CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
 *    OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
 *    OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. */
//**************************************************************************************

// This define prevents this .h file from being included multiple times in a .cpp file
#ifndef _TASK_COM_PORT_H_
#define _TASK_COM_PORT_H_

#include <stdlib.h>                                                     // Prototype declarations for I/O functions
#include <avr/io.h>                                                     // Header for special function registers
#include "FreeRTOS.h"                                           // Primary header for FreeRTOS
#include "task.h"                                                       // Header for FreeRTOS task functions
#include "queue.h"                                                      // FreeRTOS inter-task communication queues
#include "frt_task.h"                                           // ME405/507 base task class
#include "time_stamp.h"                                         // Class to implement a microsecond timer
#include "frt_queue.h"                                          // Header of wrapper for FreeRTOS queues
#include "frt_shared_data.h"                            // Header for thread-safe shared data
#include "rs232int.h"                                           // ME405/507 library for serial comm.

//-------------------------------------------------------------------------------------
/** This constructor creates a task which handles the wired communication between the
 *  host and the base station.
 */
class task_com_port : public frt_task
{
protected:
        // No protected variables or methods for this class
        emstream* ser_port;
public:
        // This constructor creates a generic task of which many copies can be made
        task_com_port (const char*, unsigned portBASE_TYPE, size_t, emstream*);

        // This method is called by the RTOS once to run the task loop for ever and ever.
        void run (void);
};

#endif // _TASK_COM_PORT_H_
#endif // ifdef BASE
```

```
#ifdef BASE
//**************************************************************************************
/** \file task_com_port.cpp
 *    This file contains the code for a task class which handles communciation with
 *        a host computer via UART/USB. This task is responsible for coordinating data
 *    packets between the host computer and task_bitcloud_node through the use of
 *    flags on the packet.
 *  Revisions:
 *    \li 04-11-2013 HAV Original file created
 *
 *  License:
 *    This file is copyright 2013 by HA Vierra and released under the Lesser GNU
 *    Public License, version 2. It intended for educational use only, but its use
 *    is not limited thereto. */
/*    THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
 *    AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 *    IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 *    ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
 *    LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUEN-
 *    TIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 *    OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
 *    CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
 *    OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
 *    OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. */
//**************************************************************************************

#include "frt_text_queue.h"                  // Header for text queue class
#include "task_com_port.h"                   // Header for this task
#include "shares.h"                          // Shared inter-task communications

/** This constructor creates a task which handles the wired communication between the
 *  host and the base station.
 */
task_com_port::task_com_port (const char* a_name,
                                                                 unsigned portBASE_TYPE a_priority,
                                                                 size_t a_stack_size,
                                                                 emstream* p_ser_dev
                                                                )
        : frt_task (a_name, a_priority, a_stack_size, p_ser_dev)
{
        // Nothing is done in the body of this constructor. All the work is done in the
        // call to the frt_task constructor on the line just above this one
        ser_port = p_ser_dev;
}

//-------------------------------------------------------------------------------------
/** This method is called once by the RTOS scheduler. Each time around the for (;;)
 *  loop, it reads in a full data packet from the host computer. If the server data
 *  request flag is received, the base station sends back the same packet to the host.
 *  if the sensor data or sensor config flag is received, the base station relays the
 *  packet to the designated node.
 */
void task_com_port::run (void)
{
        //initialize all memory in packet to zero
        memset(command_pack.data, 0, sizeof(command_pack.data));

        //Temporary buffer for packet read-in
        uint8_t* packet_read_in;

        for (;;)
        {
                //Initialize pointer to base of command_pack struct
                packet_read_in = &(command_pack.flag);

                // Read an entire packet in
                for(uint16_t i = 0; i < sizeof(serial_packet); )
                {
                        if (ser_port->check_for_char ())//check for a charachter on the uart line
                        {
                                *packet_read_in = p_serial->getchar ();
                                packet_read_in++;
                                i++;
                        }
                }
                // Based on the flag, route the packet accordingly
                switch(command_pack.flag)
                {   // Signifies a data request from a configured node/sensor
                        case SER_DATA_REQ:
                        {
                                //For host computer
                                *ser_port<<(uint8_t *)&command_pack;
                        }       break;
                        // Signifies a configuration request for a node
                        case SENSOR_CONFIG_REQ:
                        {
                                send_packet -> put(true);
                        }       break;
            case SENSOR_DATA:
                        {
                                send_packet -> put(true);
                        }       break;

                        case DEBUG:
                        {  //For Testing
                                *ser_port<<command_pack.flag<<endl<<command_pack.node_id<<endl;
                                for(uint8_t i =0; i < sizeof(command_pack.data); i++)
                                        *ser_port<<command_pack.data[i]<<endl;
                                send_packet -> put(true);
                        }   break;
                        default:
                                *ser_port<<"FLAG ERROR. Packet Contents:"<<endl;
                                for(uint8_t i =0; i < sizeof(serial_packet); i++)
                                        ser_port->putchar(((uint8_t *)&command_pack)[i]);
                                break;
                }
        }
}
#endif // ifdef BASE
```

```cpp
#ifdef NODE
//**************************************************************************************
/** \file task_data_acquisition.h
 *    This file contains the code for a task class which handles communciation with
 *        various sensors. Based on the name string sent in the data packet to the node,
 *    a corresponding sensor driver object is initialized, ran, and its data is put into
 *    the packet to be sent.
 *
 *  Revisions:
 *    \li 03-07-2013 HAV Original file created
 *
 *  License:
 *    This file is copyright 2013 by HA Vierra and released under the Lesser GNU
 *    Public License, version 2. It intended for educational use only, but its use
 *    is not limited thereto. */
/*    THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
 *    AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 *    IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 *    ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
 *    LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUEN-
 *    TIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 *    OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
 *    CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
 *    OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
 *    OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. */
//**************************************************************************************

// This define prevents this .h file from being included multiple times in a .cpp file
#ifndef _TASK_DATA_ACQUISITION_H_
#define _TASK_DATA_ACQUISITION_H_

#include <stdlib.h>                                                     // Prototype declarations for I/O functions
#include <avr/io.h>                                                     // Header for special function registers

#include "FreeRTOS.h"                                           // Primary header for FreeRTOS
#include "task.h"                                                       // Header for FreeRTOS task functions
#include "queue.h"                                                      // FreeRTOS inter-task communication queues

#include "frt_task.h"                                           // ME405/507 base task class
#include "time_stamp.h"                                         // Class to implement a microsecond timer
#include "frt_queue.h"                                          // Header of wrapper for FreeRTOS queues
#include "frt_shared_data.h"                            // Header for thread-safe shared data

#include "rs232int.h"                                           // ME405/507 library for serial comm.
#include "MPU6050_driver.h"                                     //
#include "BMP085_driver.h"

//--------------------------------------------------------------------------------------
/** \brief This task acquired data from a sensor specified by a packet sent from a base
 *         station.
 */
class task_data_acquisition : public frt_task
{
private:

protected:
        emstream* ser_port;
public:
        // This constructor creates a generic task of which many copies can be made
        task_data_acquisition (const char*, unsigned portBASE_TYPE, size_t, emstream*);

        // This method is called by the RTOS once to run the task loop for ever and ever.
        void run (void);
};
```

```

#endif // _TASK_DATA_ACQUISITION_H_
#endif //ifdef NODE
```

```
#ifdef NODE
//*************************************************************************************
/** \file task_data_acquisition.cpp
 *    This file contains the code for a task class which handles communciation with
 *        various sensors. Based on the name string sent in the data packet to the node,
 *    a corresponding sensor driver object is initialized, ran, and its data is put into
 *    the packet to be sent.
 *
 *  Revisions:
 *    \li 03-07-2013 HAV Original file created
 *    \li 05-30-2013 Added dynamic driver allocation and data packer function
 *
 *  License:
 *    This file is copyright 2013 by HA Vierra and released under the Lesser GNU
 *    Public License, version 2. It intended for educational use only, but its use
 *    is not limited thereto. */
/*    THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
 *    AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 *    IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 *    ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
 *    LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUEN-
 *    TIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 *    OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
 *    CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
 *    OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
 *    OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. */
//*************************************************************************************

#include "frt_text_queue.h"                  // Header for text queue class
#include "task_data_acquisition.h"           // Header for this task
#include "shares.h"                          // Shared inter-task communications
#include "semphr.h"

// Function to pack data with data-type delimiter
void data_packer(void* destination, void* data1, char data_type1, void* data2,
     char data_type2, void* data3, char data_type3);

//------------------------------------------------------------------------------------
/** This constructor creates a task which uses sensor driver objects to acquire sensor
        data and pack it to be sent back to the base station.
 *  @param a_priority The priority at which this task will initially run (default: 0)
 *  @param a_stack_size The size of this task's stack in bytes
 *                      (default: configMINIMAL_STACK_SIZE)
 *  @param p_ser_dev Pointer to a serial device (port, radio, SD card, etc.) which can
 *                   be used by this task to communicate (default: NULL)
 */

task_data_acquisition::task_data_acquisition (const char* a_name,
                                                                 unsigned portBASE_TYPE a_priority,
                                                                 size_t a_stack_size,
                                                                 emstream* p_ser_dev
                                                                )
        : frt_task (a_name, a_priority, a_stack_size, p_ser_dev)
{
        // Nothing is done in the body of this constructor. All the work is done in the
        // call to the frt_task constructor on the line just above this one
        ser_port = p_ser_dev;
}

//------------------------------------------------------------------------------------
/** This method is called once by the RTOS scheduler. Each time around the for (;;)
 *  loop, it waits for a data queue holding a flag which signifies a setup command
 *  or a data request command from the base station. A setup command is responded with
 *  the deletion of any active driver, and the creation of a new sensor driver based on
 *  the sensor name string. A data request is handles differently for each sensor, but
 *  all data is packed into the data packet and a flag is sent via a queue to
 *  task_bitcloud_node signaling a transmit is ready.
 */
void task_data_acquisition::run (void)
{
        // Make a variable which will hold times to use for precise task scheduling
        portTickType previousTicks = xTaskGetTickCount ();

        // Sensor data buffers
        uint16_t sensor_data_s[3];
        int32_t  sensor_data_i[3];

        // Declare all Sensor objects here and initialize to zero
        MPU6050_driver* mpu6050_driver = 0;
        BMP085_driver* bmp085_driver = 0;

        for (;;)
        {       // Wait for flag signal from task_bitcloud_node
                if(!(get_sensor_data->get()))//true = config, false = sensor read
                {       //Check the driver address to see if it's in use, then delete it.
                        if(mpu6050_driver != 0)
                        {
                                vPortFree(mpu6050_driver);
                                mpu6050_driver = 0;
                                *ser_port<<"deleted MPU6050 driver"<<endl;
                        }
                        if(bmp085_driver != 0)
                        {
                                vPortFree(bmp085_driver);
                                mpu6050_driver = 0;
                                *ser_port<<"deleted BMP085 driver"<<endl;
                        }
                        else
                        {
                                *ser_port<<"First time config:"<<endl;
                        }
                        // Check the sensor name, and create a driver accordingly
                        if((strcmp(sensor_name, "MPU6050")) == 0)
                        {
                                // Allocate memory for new sensor driver
                                mpu6050_driver = (MPU6050_driver*)pvPortMalloc(sizeof(MPU6050_driver));
                                mpu6050_driver = new MPU6050_driver(ser_port);
                                *ser_port<<"Created an MPU6050_driver"<<endl;
                        }
                        else if((strcmp(sensor_name, "BMP085")) == 0)
                        {
                                // Allocate memory for new sensor driver
                                bmp085_driver = (BMP085_driver*)pvPortMalloc(sizeof(BMP085_driver));
                                bmp085_driver = new BMP085_driver(ser_port);
                                *ser_port<<"Created an BMP085_driver"<<endl;
                        }
                        else
                        {
                                *ser_port<<"Sensor Name Not Found. Config Failed"<<endl;
                        }
                }
                else // Read Sensor data
                {
                        if((strcmp(sensor_name, "MPU6050")) == 0)
                        {
                                switch(sensor_command[0])
                                {
                                        // Accelerometer
                                        case 'a':
                                                mpu6050_driver -> Get_Accel_Values();
                                                //Clear all data to zero first
                                                memset(command_pack.data, 0, sizeof(command_pack.data));
                                                sensor_data_s[0] = mpu6050_driver -> ACCEL_XOUT;
                                                sensor_data_s[1] = mpu6050_driver -> ACCEL_YOUT;
                                                sensor_data_s[2] = mpu6050_driver -> ACCEL_ZOUT;
                                                // Print out data
                                                //*ser_port<<dec<<"ACCEL_XOUT: "<<mpu6050_driver-
```

```
>ACCEL_XOUT<<
                        //endl<<"ACCEL_YOUT: "<<mpu6050_driver->ACCEL_YOUT<<endl<<
                        //"ACCEL_ZOUT: "<<mpu6050_driver->ACCEL_ZOUT<<endl<<endl;
                        break;
                    // Gyroscope
                    case 'g':
                        mpu6050_driver -> Get_Gyro_Rates();
                        //Clear all data to zero first
                        memset(command_pack.data, 0, sizeof(command_pack.data));
                        sensor_data_s[0] = mpu6050_driver -> GYRO_XOUT;
                        sensor_data_s[1] = mpu6050_driver -> GYRO_YOUT;
                        sensor_data_s[2] = mpu6050_driver -> GYRO_ZOUT;
                        // Print out data
                        //*ser_port<<dec<<"GYRO_XOUT: "<<mpu6050_driver->GYRO_XOUT<<
                        //endl<<"GYRO_YOUT: "<<mpu6050_driver->GYRO_YOUT<<endl<<
                        //"GYRO_ZOUT: "<<mpu6050_driver->GYRO_ZOUT<<endl<<endl;
                        break;
                    default:
                        break;
                }
                // pack up sensor data
                data_packer((void*)command_pack.data, (void*)sensor_data_s,
                            DATA_TYPE_SHORT, (void*)(sensor_data_s + 1),
                            DATA_TYPE_SHORT, (void*)(sensor_data_s + 2),
                            DATA_TYPE_SHORT);
            }
            else if((strcmp(sensor_name, "BMP085")) == 0)
            {
                bmp085_driver ->Read_Cal_Data();
                bmp085_driver ->Read_Pres();

                //Clear all data to zero first
                memset(command_pack.data, 0, sizeof(command_pack.data));
                sensor_data_i[0] = bmp085_driver -> pressure_pa;
                sensor_data_i[1] = bmp085_driver -> temp_raw;
                sensor_data_i[2] = 0;
                // Print sensor data
                //*ser_port<<dec<<"Pressure: "<<bmp085_driver -> pressure_pa<<endl
                //               <<"Temp: "<<bmp085_driver -> temp_raw<<endl;

                // pack up sensor data
                data_packer((void*)command_pack.data, (void*)sensor_data_i,
                            DATA_TYPE_INT, (void*)(sensor_data_i + 1),
                            DATA_TYPE_INT, (void*)(sensor_data_i + 2),
                            DATA_TYPE_INT);
            }
            else
            {
                *ser_port<<"ERROR: node configured, but sensor name incorrect"<<endl;
            }

        }
        // Start a data transmit with new packet
        //(same pack if config, sensor data if data requested)
        send_packet ->put(true);


        runs++;
        delay_from_to (previousTicks, configMS_TO_TICKS (100));
    }
}
```

```
/** This function copies data into the data packet to be sent to the base station
 *
 *  @param void* destination: pointer to base address of data packet to be packed
 *  @param void* data1, data2, data3: pointer to data variables
 *  @param char data_type1, data_type2, data_type3: string denoting data type
 */
void data_packer(void* destination, void* data1, char data_type1, void* data2,
         char data_type2, void* data3, char data_type3)
{
        uint8_t new_data_pack[PACKET_DATA_SIZE_BYTES] = {0};
        uint8_t* p_new_data_pack = new_data_pack;

        switch(data_type1)
        {
                case DATA_TYPE_FLOAT:
                        memcpy(p_new_data_pack, &data_type1, sizeof(data_type1));
                        p_new_data_pack += sizeof(data_type1);
                        memcpy(p_new_data_pack, data1, sizeof(float));
                        p_new_data_pack += sizeof(float);
                        break;
                case DATA_TYPE_INT:
                        memcpy(p_new_data_pack, &data_type1, sizeof(data_type1));
                        p_new_data_pack += sizeof(data_type1);
                        memcpy(p_new_data_pack, data1, sizeof(int32_t));
                        p_new_data_pack += sizeof(int32_t);
                        break;
                case DATA_TYPE_SHORT:
                        memcpy(p_new_data_pack, &data_type1, sizeof(data_type1));
                        p_new_data_pack += sizeof(data_type1);
                        memcpy(p_new_data_pack, data1, sizeof(uint16_t));
                        p_new_data_pack += sizeof(uint16_t);
                        break;
                default:
                        return;
                        break;
        }

        switch(data_type2)
        {
                case DATA_TYPE_FLOAT:
                        memcpy(p_new_data_pack, &data_type2, sizeof(data_type2));
                        p_new_data_pack += sizeof(data_type2);
                        memcpy(p_new_data_pack, data2, sizeof(float));
                        p_new_data_pack += sizeof(float);
                        break;
                case DATA_TYPE_INT:
                        memcpy(p_new_data_pack, &data_type2, sizeof(data_type2));
                        p_new_data_pack += sizeof(data_type2);
                        memcpy(p_new_data_pack, data2, sizeof(int32_t));
                        p_new_data_pack += sizeof(int32_t);
                        break;
                case DATA_TYPE_SHORT:
                        memcpy(p_new_data_pack, &data_type2, sizeof(data_type2));
                        p_new_data_pack += sizeof(data_type2);
                        memcpy(p_new_data_pack, data2, sizeof(uint16_t));
                        p_new_data_pack += sizeof(uint16_t);
                        break;
                default:
                        return;
                        break;
        }

        switch(data_type3)
        {
                case DATA_TYPE_FLOAT:
                        memcpy(p_new_data_pack, &data_type3, sizeof(data_type3));
                        p_new_data_pack += sizeof(data_type3);
                        memcpy(p_new_data_pack, data3, sizeof(float));
                        p_new_data_pack += sizeof(float);
                        break;
                case DATA_TYPE_INT:
                        memcpy(p_new_data_pack, &data_type3, sizeof(data_type3));
                        p_new_data_pack += sizeof(data_type3);
                        memcpy(p_new_data_pack, data3, sizeof(int32_t));
                        p_new_data_pack += sizeof(int32_t);
                        break;
                case DATA_TYPE_SHORT:
                        memcpy(p_new_data_pack, &data_type3, sizeof(data_type3));
                        p_new_data_pack += sizeof(data_type3);
                        memcpy(p_new_data_pack, data3, sizeof(uint16_t));
                        p_new_data_pack += sizeof(uint16_t);
                        break;
                default:
                        return;
                        break;
        }

        memcpy(destination, new_data_pack, sizeof(new_data_pack));
}
#endif //ifdef NODE
```

```
//**************************************************************************************
/** \file WSN_main.cpp
 *   This is the main function for the Poly_Sense node and base station program. Based
 *   on the make configuration, certain libraries and tasks pertaining exclusively to
 *   the target of interest (base station or node) are included. This function declares
 *   an instance of all global variables, initializes the tasks needed to operate, and
 *   starts the scheduler.
 *
 *  Revisions:
 *    \li 03-07-2013 HAV Original file created
 *    \li 06-05-2013 HAV Removed ADC test code and added global declarations
 *
 *  License:
 *    This file is copyright 2013 by HA Vierra and released under the Lesser GNU
 *    Public License, version 2. It intended for educational use only, but its use
 *    is not limited thereto. */
/*    THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
 *    AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 *    IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 *    ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
 *    LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUEN-
 *    TIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 *    OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
 *    CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
 *    OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
 *    OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. */
//**************************************************************************************

//------------------------------------Libraries---------------------------------------//
#include <stdlib.h>                         // Prototype declarations for I/O functions
#include <avr/io.h>                         // Port I/O for SFR's
#include <avr/wdt.h>                        // Watchdog timer header
#include <string.h>                         // Functions for C string handling
#include "FreeRTOS.h"                       // Primary header for FreeRTOS
#include "task.h"                           // Header for FreeRTOS task functions
#include "queue.h"                          // FreeRTOS inter-task communication queues
#include "croutine.h"                       // Header for co-routines and such
#include "rs232int.h"                       // library for serial comm.
#include "frt_task.h"                       // Header of wrapper for FreeRTOS tasks
#include "frt_text_queue.h"                 // Wrapper for FreeRTOS character queues
#include "frt_queue.h"                      // Header of wrapper for FreeRTOS queues
#include "frt_shared_data.h"                // Header for thread-safe shared data
#include "shares.h"                         // Global ('extern') declarations
#include <util/delay.h>
// Defined in makefile as for base station or node configuration
#ifdef NODE
        #include "task_bitcloud_node.h"
        #include "task_data_acquisition.h"
#elif BASE
        #include "task_bitcloud_base_station.h"
        #include "task_com_port.h"
#else
        #error "no node type spec"
#endif

//------------------------------------Defines----------------------------------------//
/*  Analog Switch
 *       - Active Low Enable: PD4
 *       - Select: PB7
 *         - 0: 5V operation
 *         - 1: 3.3V operation
 *
 *      Level Shifter
 *       - Active High Enable: PD5
 *
 *      Voltage Regulator
 *       - Active High Enable: PF6
**/
#define ANALOG_SWITCH_EN_PIN  4
#define ANALOG_SWITCH_SEL_PIN 7
```

```
#define LEVEL_SHIFTER_EN_PIN  5
#define VOLTAGE_REGULATOR_EN_PIN 6

#define ANALOG_SWITCH_SEL_5V  0
#define ANALOG_SWITCH_SEL_3V3 1
#define LEVEL_SHIFTER_EN_5V  1
#define LEVEL_SHIFTER_EN_3V3 0
#define VOLTAGE_REGULATOR_EN_5V 1
#define VOLTAGE_REGULATOR_EN_3V3 0

//-----------------------------------Prototypes---------------------------------------//
uint8_t initialize_polysense_mode(uint8_t operating_voltage);

//-------------------------------------Globals----------------------------------------//
//Create a data packet
serial_packet command_pack;
//Binary Semaphore/queue to signal packet ready and determine RX or TX
frt_queue<bool>* send_packet = new frt_queue<bool>(1);
// Semaphore/queue to signal a sensor read
frt_queue<bool>* get_sensor_data = new frt_queue<bool>(1);
rs232* ser_port = new rs232 (9600, 1);
// True => TX mode, False => RX mode
bool transmit = false;
char sensor_name[PACKET_DATA_SIZE_BYTES] = {0};
char sensor_command[PACKET_DATA_SIZE_BYTES] = {0};
bool sensor_config_error = false;
// State variable telling whether node has been configured for a particular sensor yet
bool node_configured = false; // true = TX, false = RX

//--------------------------------------Main------------------------------------------//
/** The main function sets up the RTOS.  Some test tasks are created. Then the
 *  scheduler is started up; the scheduler runs until power is turned off or there's a
 *  reset.
 *  @return This is a real-time microcontroller program which doesn't return. Ever.
 */
int main (void)
{
        // Disable the watchdog timer unless it's needed later. This is important because
        // sometimes the watchdog timer may have been left on...and it tends to stay on
        MCUSR = 0;
        wdt_disable ();

        // Initialize board to 3.3V operation for digital i2c sensors
        initialize_polysense_mode(3);

//----------------------------------Initialize Tasks---------------------------------//

        #ifdef NODE
                *ser_port << clrscr << PMS ("PolySense Node: Starting Program") << endl;
                send_packet->put(false);
                new task_bitcloud_node("Node", task_priority (2), 800, ser_port);
                new task_data_acquisition("Sense", task_priority (1), 1000, ser_port);
        #elif BASE
                new task_com_port("Host", task_priority (1), 800, ser_port);
                new task_bitcloud_base_station("Base", task_priority (2), 800, ser_port);
        #else
                #error "no node type spec"
        #endif

        // Start the Scheduler
        vTaskStartScheduler ();
}

//----------------------------Function Implementations-------------------------------//
/** This function prints out the contents of a data packet to the serial port of a host
 *
 *  @param uint8_t operating_voltage: configs Poly_Sense board for 5V or 3.3V operation
 */
uint8_t initialize_polysense_mode(uint8_t operating_voltage)
{
        if((operating_voltage != 3) && (operating_voltage != 5))
        {
                return 0;
        }
        // Set pinE0 as output and tie low for gnd reference
        DDRE |= (1<<0);
        PORTE &= ~(1<<0);

        //Set pins for analog switch, voltage regulator and level shifter as outputs
        DDRD |= ((1 << ANALOG_SWITCH_EN_PIN) | (1 << LEVEL_SHIFTER_EN_PIN));
        DDRF |= (1 << VOLTAGE_REGULATOR_EN_PIN);
        DDRB |= (1 << ANALOG_SWITCH_SEL_PIN);


        // Enable the active low analog switch. Used in 5V and 3.3V operation
        PORTD &= ~(1 << ANALOG_SWITCH_EN_PIN);

        DDRF |= (1 << 7);
        PORTF|= (1 << 7);
        switch(operating_voltage)
        {
                //3V3 operation
                case 3:
                        //Disable 5V regulator
                        PORTF &= ~(VOLTAGE_REGULATOR_EN_3V3 <<
VOLTAGE_REGULATOR_EN_PIN);
                        //Disable Level Shifter for 3V3 Operation
                        PORTD &= ~(LEVEL_SHIFTER_EN_3V3 << LEVEL_SHIFTER_EN_PIN);
                        // Set the analog switch to bypass the level shifter
                        PORTB |= (ANALOG_SWITCH_SEL_3V3 << ANALOG_SWITCH_SEL_PIN);
                        break;

                // 5V0 Operation
                case 5:
                        //Enable 5V regulator
                        PORTF |= (VOLTAGE_REGULATOR_EN_5V << VOLTAGE_REGULATOR_EN_PIN);
                        //Enable Level Shifter for 5V Operation
                        PORTD |= (LEVEL_SHIFTER_EN_5V << LEVEL_SHIFTER_EN_PIN);
                        // Set the analog switch to route through the level shifter
                        PORTB &= ~(ANALOG_SWITCH_SEL_5V << ANALOG_SWITCH_SEL_PIN);
                        break;

                default:
                        return 0;
        }
        return 1;
}


/** This function prints out the contents of a data packet to the serial port of a host
 *
 *  @param uint8_t type: value used to signify a character print or integer print
 */
void print_packet(uint8_t type)
{
        //rs232* ser_port = new rs232 (9600, 1);

        if(type == GET_CHAR)
        {
                for(uint8_t i =0; i < sizeof(serial_packet); i++)
                        ser_port->putchar(((uint8_t *)&command_pack)[i]);
        }
        else if(type ==SER_PORT)
        {
                for(uint8_t i =0; i < sizeof(command_pack); i++)
                        *ser_port<<((uint8_t *)(&command_pack))[i]<<endl;
        }
}
```

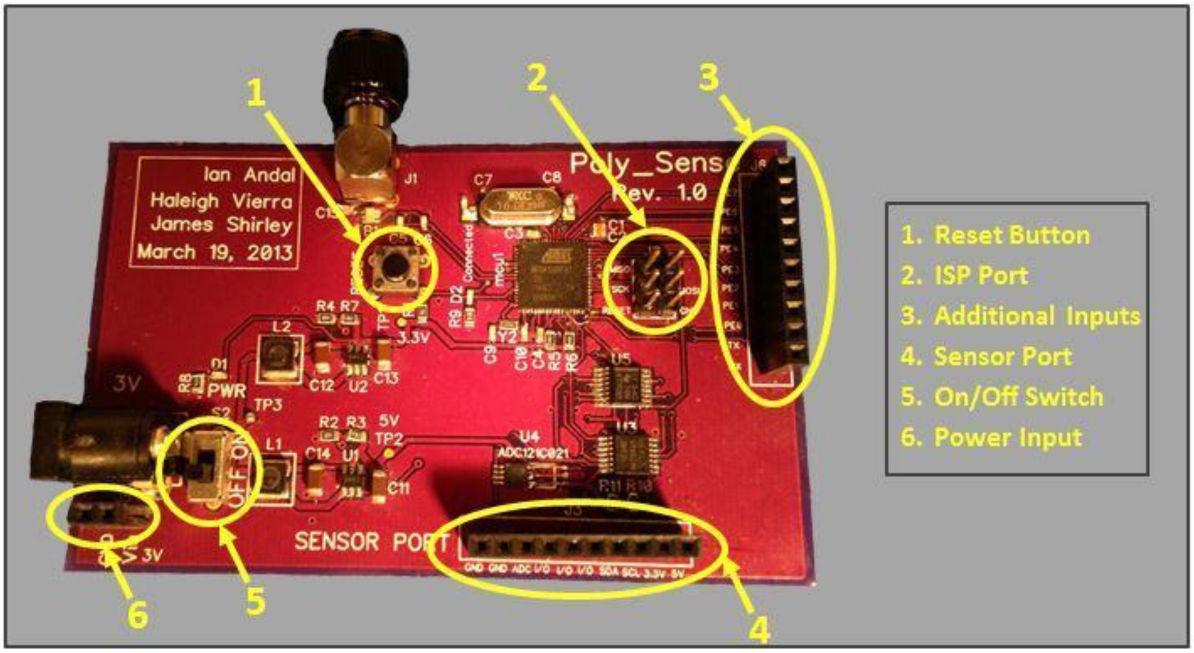
APPENDIX D: User Manual

Poly_Sense User Manual

California Polytechnic State University
San Luis Obispo
Spring 2013

Ian Andal, *Electrical Engineering*
James Shirley, *Computer Science*
Haleigh Vierra, *Electrical Engineering*

Features



Getting Started:

System Requirements

***Poly_Sense* Monitor**

Java 7

Tested on:

Windows 8 x86_64, Archlinux x86_64

4gb RAM, core i7 1.7ghz

***Poly_Sense* Server**

Tested on:

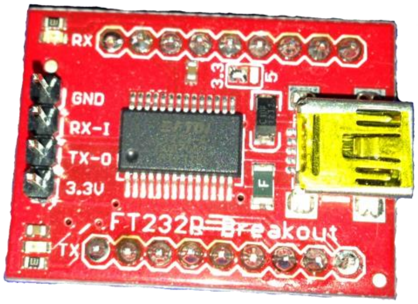
Windows 8 x86_64 (cygwin libraries), Archlinux x86_64

4gb RAM, core i7 1.7ghz

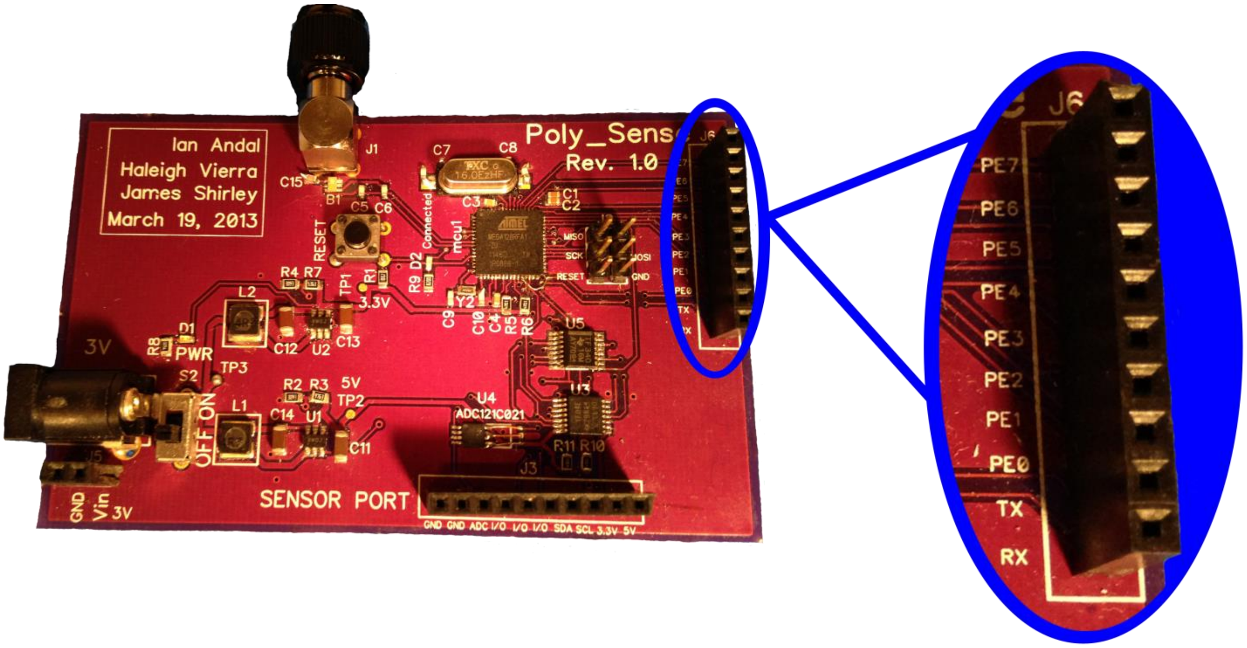
Setting up your hardware

Connecting the base station

Using the current *Poly_Sense* platform, connecting the base station requires the use of an external Serial to USB converter as seen in the image below.



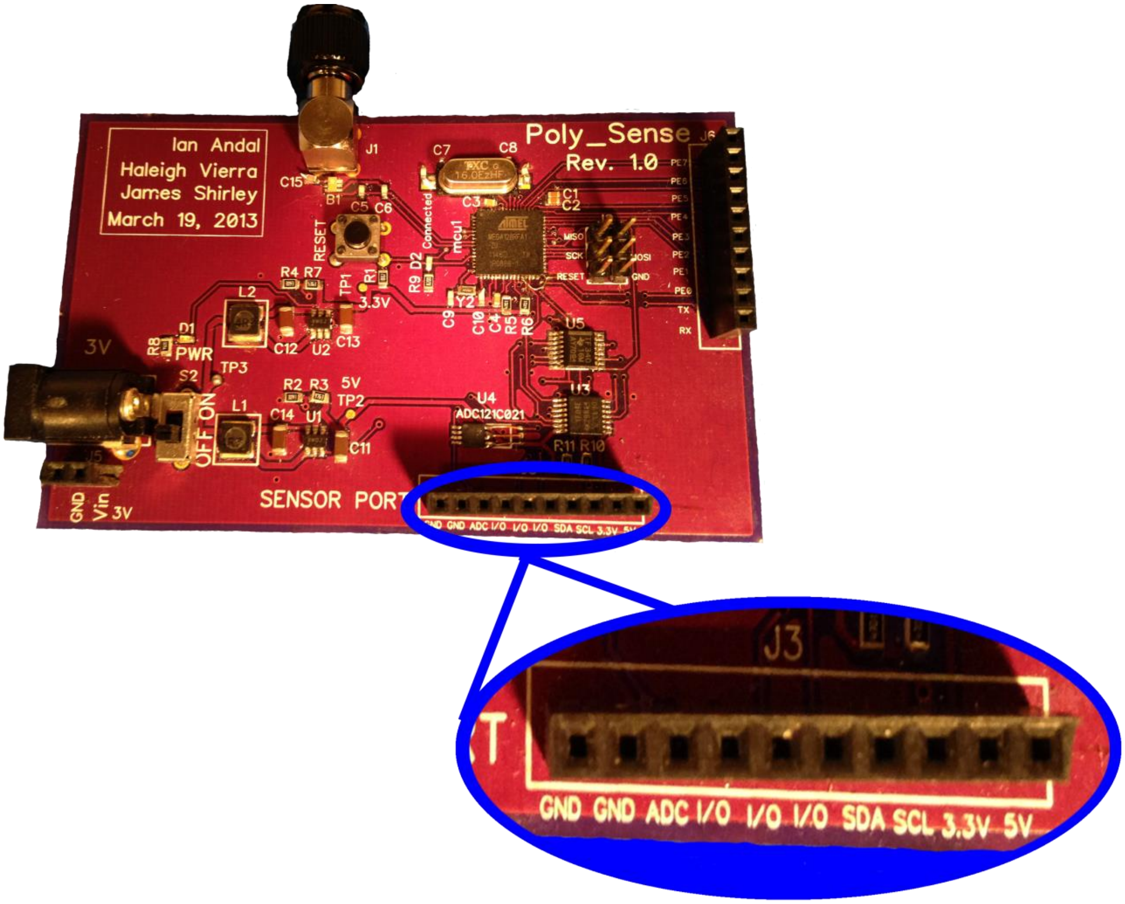
To use the Serial-USB converter, simply connect the RX pin on the converter to the TX pin of the base station, then connect the TX pin of the convert to the RX pin of the base station. Also, be sure to include a ground connection between the Serial to USB converter and the base station; any of the pins on the additional inputs port seen in the image below are configured to be tied to ground on startup.



Once the converter is connected to the base station, simply connect a USB cable from the convert to the host computer to complete the setup.

Adding a sensor to a node

To connect a sensor to a *Poly_Sense* node, you can simply connect to the necessary pins on the sensor port header as seen below:



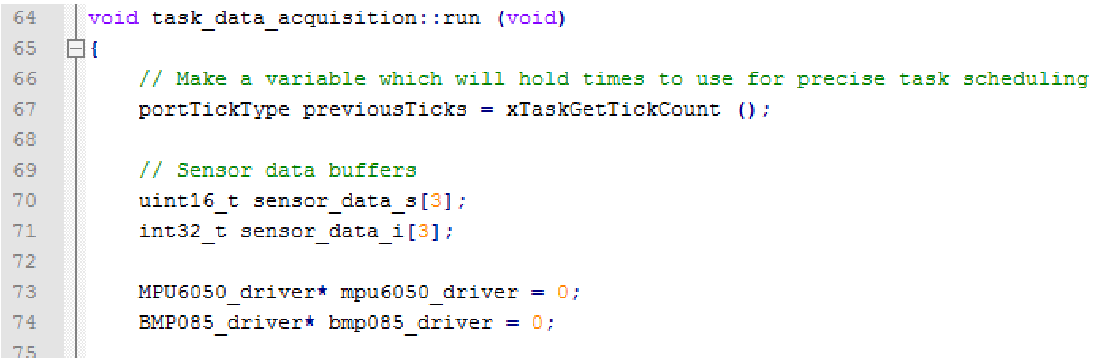
It is helpful to create an adapted board to interface your to the sensor port, however the use of a breadboard and jumper cables. Note that the current release of the *Poly_Sense* system does not support the use analog sensors, so the ADC pin in the sensor port should not be used.

System Programming

Adding a New Sensor in Software- Driver API

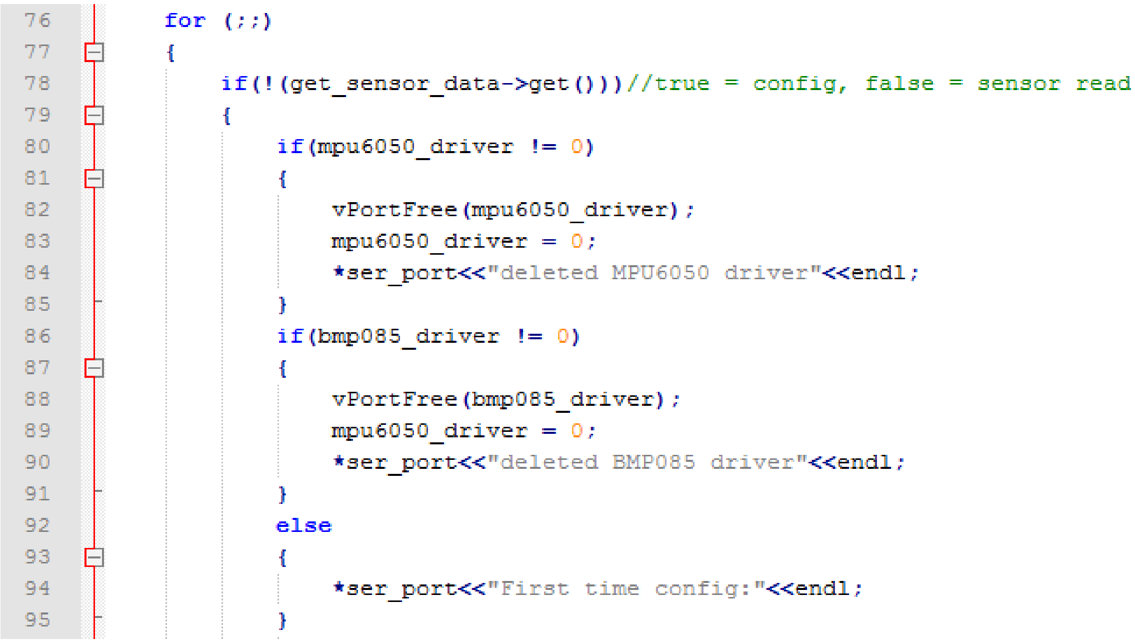
To add a new sensor to the *Poly_Sense* node, you must first create a driver to be included in the project file for the node. This can be done by extending the class "i2c_master" such that the methods I2C_write() and I2C_read can be utilized within the driver. Use the MPU6050_driver.cpp and MPU6050_driver.h files as an example in your development. Be sure to include a constructor containing a method to initialize any internal registers, also include methods to acquire data from your sensor within the driver.

Once a tested driver is complete, open the file: task_data_acquisition.h and add a “#include” for the header file for your new driver at the top of the file near all of the other includes. Next, open task_data_acquisition.cpp, within the ‘run’ method, create a pointer an object of your driver class and initialize its address to zero as seen below for the MPU6050 and BMP085 driver classes:



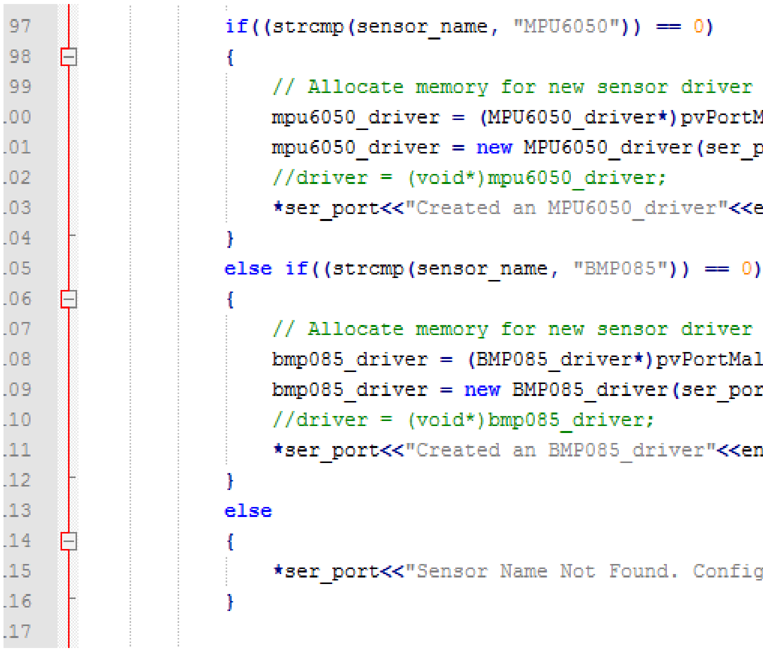
```
void task_data_acquisition::run (void)
{
    // Make a variable which will hold times to use for precise task scheduling
    portTickType previousTicks = xTaskGetTickCount ();

    // Sensor data buffers
    uint16_t sensor_data_s[3];
    int32_t sensor_data_i[3];

    MPU6050_driver* mpu6050_driver = 0;
    BMP085_driver* bmp085_driver = 0;
```

Next, add an ‘if statement’ to the main ‘for loop’ as seen below. You can simply copy and paste the contents of the MPU6050 or BMP085 if statement and replace the driver name with your new driver’s name. This allows the node to free the memory used by your driver if you decide to run a different sensor driver.



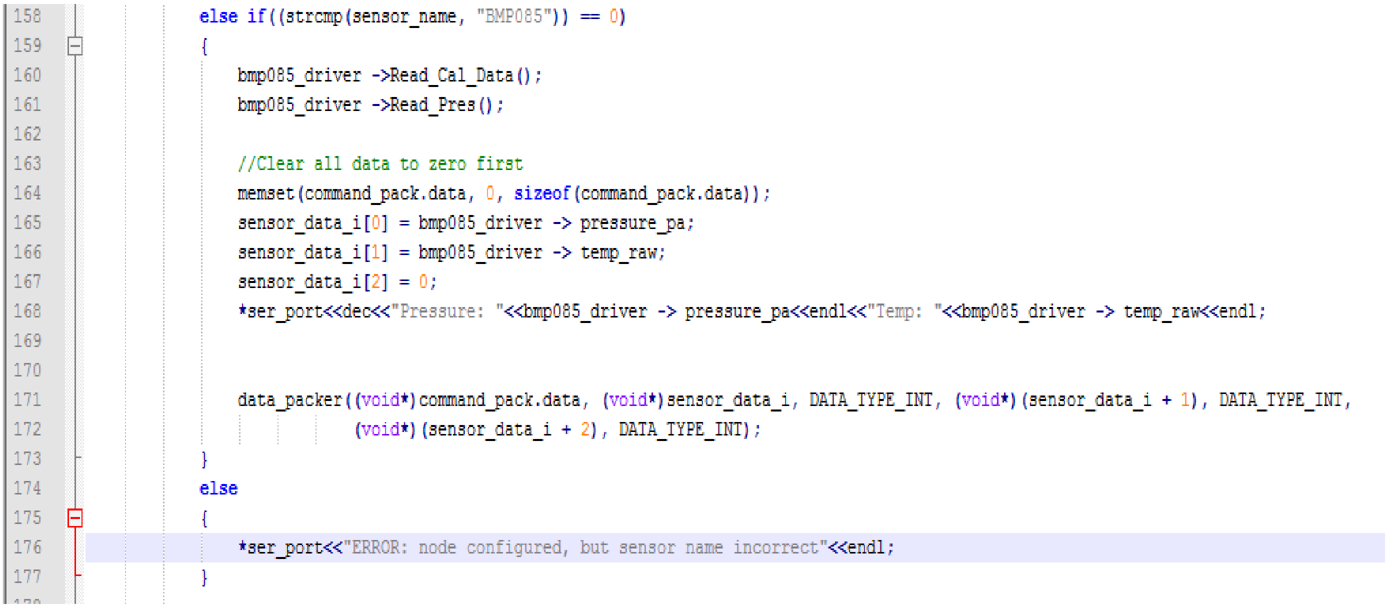
```
    for (;;)
    {
        if(!(get_sensor_data->get()))//true = config, false = sensor read
        {
            if(mpu6050_driver != 0)
            {
                vPortFree(mpu6050_driver);
                mpu6050_driver = 0;
                *ser_port<<"deleted MPU6050 driver"<<endl;
            }
            if(bmp085_driver != 0)
            {
                vPortFree(bmp085_driver);
                mpu6050_driver = 0;
                *ser_port<<"deleted BMP085 driver"<<endl;
            }
            else
            {
                *ser_port<<"First time config:"<<endl;
            }
```

Similarly, add another ‘if statement’ to the section shown below for your particular sensor. This portion of the the code allocates memory a new driver.



```
if((strcmp(sensor_name, "MPU6050")) == 0)
{
    // Allocate memory for new sensor driver
    mpu6050_driver = (MPU6050_driver*)pvPortMalloc(sizeof(MPU6050_driver));
    mpu6050_driver = new MPU6050_driver(ser_port);
    //driver = (void*)mpu6050_driver;
    *ser_port<<"Created an MPU6050_driver"<<endl;
}
else if((strcmp(sensor_name, "BMP085")) == 0)
{
    // Allocate memory for new sensor driver
    bmp085_driver = (BMP085_driver*)pvPortMalloc(sizeof(BMP085_driver));
    bmp085_driver = new BMP085_driver(ser_port);
    //driver = (void*)bmp085_driver;
    *ser_port<<"Created an BMP085_driver"<<endl;
}
else
{
    *ser_port<<"Sensor Name Not Found. Config Failed"<<endl;
}
```

To acquire data from your sensor, add an ‘else if statement’ in the section of code seen below, in a similar fashion as previous steps.



```
else if((strcmp(sensor_name, "BMP085")) == 0)
{
    bmp085_driver ->Read_Cal_Data();
    bmp085_driver ->Read_Pres();

    //Clear all data to zero first
    memset(command_pack.data, 0, sizeof(command_pack.data));
    sensor_data_i[0] = bmp085_driver -> pressure_pa;
    sensor_data_i[1] = bmp085_driver -> temp_raw;
    sensor_data_i[2] = 0;
    *ser_port<<dec<<"Pressure: "<<bmp085_driver -> pressure_pa<<endl<<"Temp: "<<bmp085_driver -> temp_raw<<endl;


    data_packer((void*)command_pack.data, (void*)sensor_data_i, DATA_TYPE_INT, (void*)(sensor_data_i + 1), DATA_TYPE_INT,
                (void*)(sensor_data_i + 2), DATA_TYPE_INT);
}
else
{
    *ser_port<<"ERROR: node configured, but sensor name incorrect"<<endl;
}
```

Use the ‘data_packer()’ function to load sensor data into the data packet which will be sent to the base station.

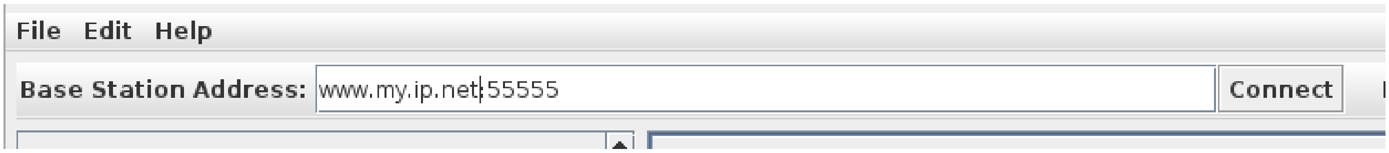
Using the Poly_Sense Server

To use the *Poly_Sense* Server simply navigate to the folder containing your compiled executable and type "./cMWSN-Server [com port]" (note: linux users require root privileges to access com ports). If you do not specify a com port, the default of "/dev/USB01" will be used.

Using the Poly_Sense Monitor

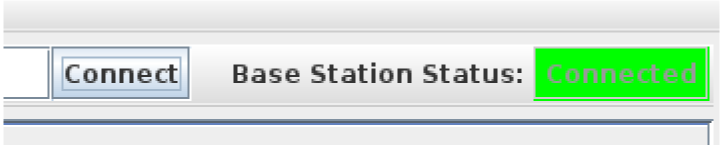
Connecting to the base station

To connect to the base station the user fill in its ip address followed by a colon and the bound port in this field and press the connect button. You can also use "localhost" as the ip address if you are running the server on the same computer.



Base Station Address: www.my.ip.net:55555

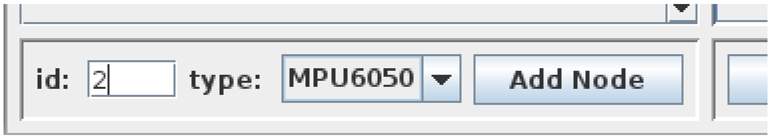
To get the base stations ip address the user can use ipconfig /all (windows) or ip addr (linux). The *Poly_Sense* Server binds to port 55555 by default. If the connection was successful, then the status on the right changes to "Connected".



Base Station Status: Connected

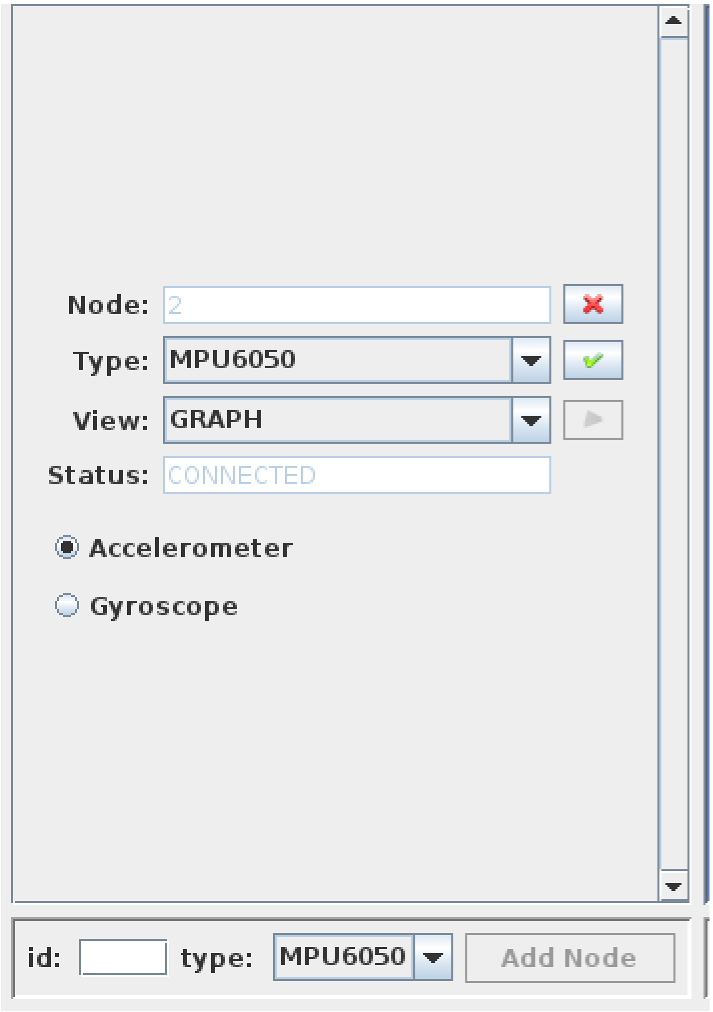
Adding and configuring a new node

To add and configure a new node the user must first be connected to the base station and have at least 1 node on and in range. Once this prerequisite is met, you can add a new node by typing its address (as labeled on the back of the node) and choosing what type of sensor is attached to it and pressing "add node".



id: 2 type: MPU6050

If the node was configured successfully then the node information should appear in the side panel.



Node: 2

Type: MPU6050

View: GRAPH

Status: CONNECTED

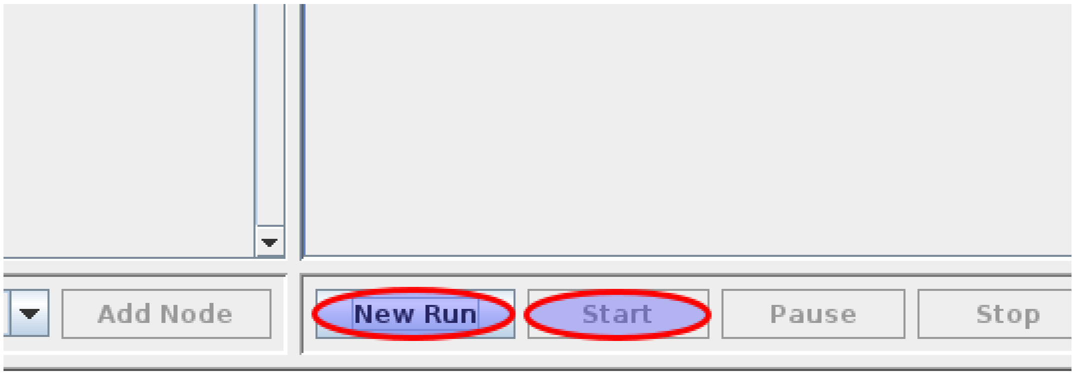
● Accelerometer

○ Gyroscope

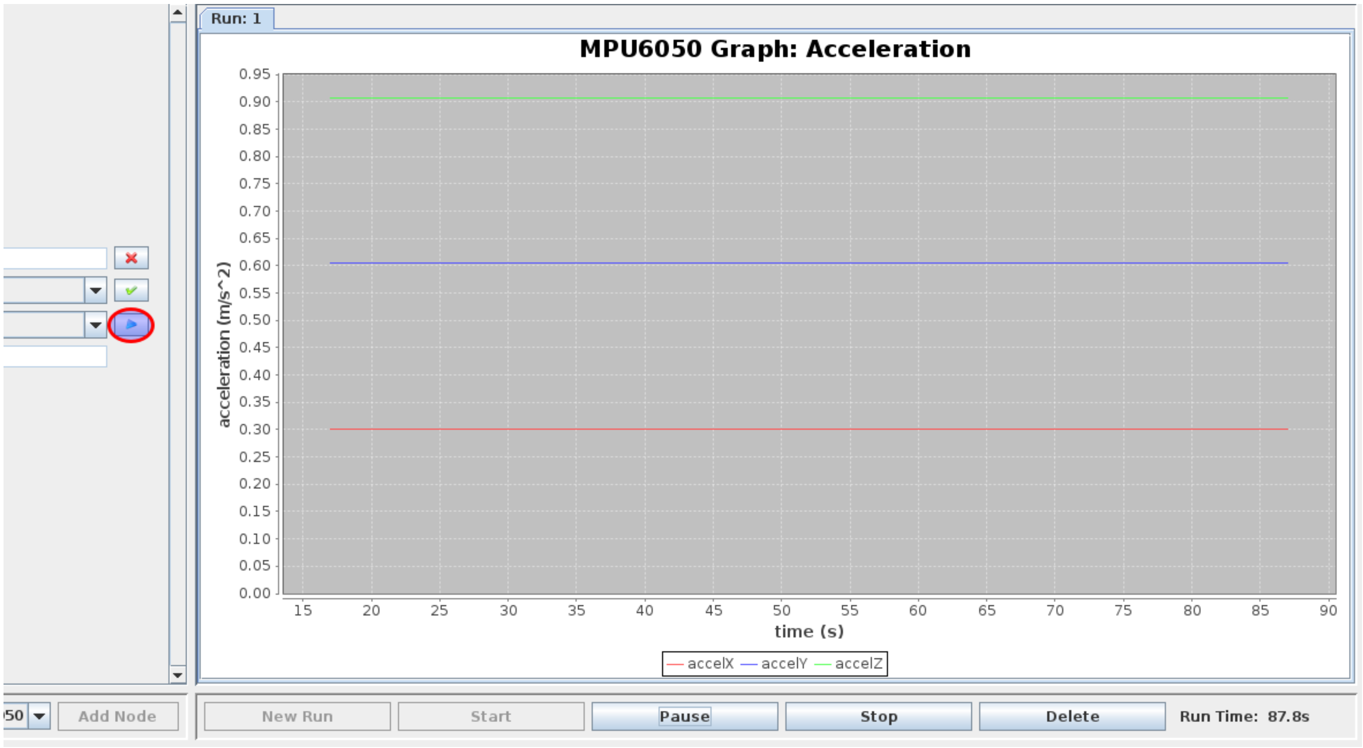
id: type: MPU6050

Collecting Data

To add and configure a new node the user must first be connected to the base station and have at least 1 configured node on and in range. Once this prerequisite is met, you can begin collecting data by adding a new run by clicking the “New Run” button and then pressing the “Start” button.



This process starts requesting and storing (depending on the Sensor class implementation) data from the sensors. To view data the user should click on the blue right arrow associated with the sensor whose data the user wants to view.

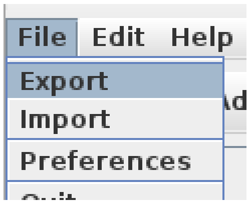


The user can manipulate the data requests by:

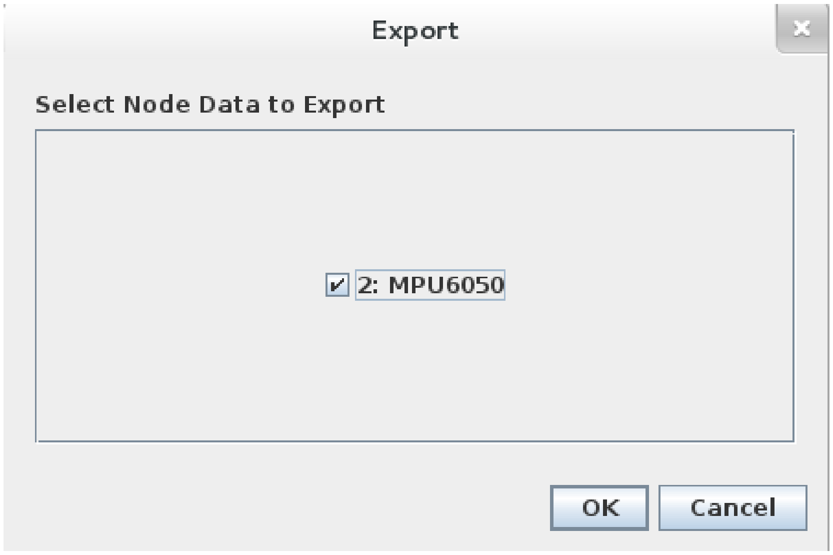
1. Pausing data requests - stops requesting and storing data
2. Stopping data requests - stops requesting and storing data as well as resetting the timer
3. Deleting this run - removes all gathered data and clears the run view
4. Starting data requests - starts/resumes requesting and storing data

Exporting Data

To add and configure a new node the user must first be connected to the base station and have at least 1 configured node on and in range, it is not strictly required that the *Poly_Sense* monitor has gathered data from the nodes. Once this prerequisite is met, the user can export the gathered sensor data by navigating to file->export.

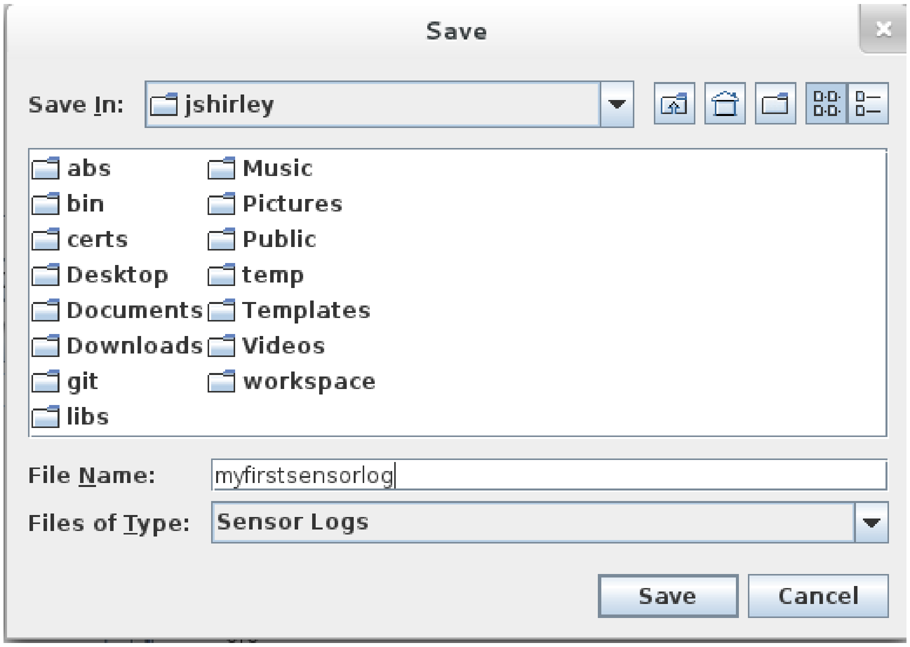


In the window that appears, the user should select the sensors whose data the user would like to export.



☑ 2: MPU6050

Finally, the user must select a output file and click “Save”.



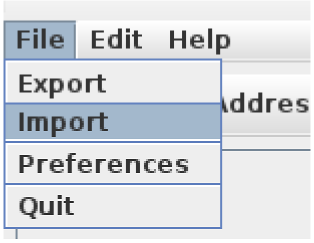
Save In: jshirley

File Name: myfirstsensorlog

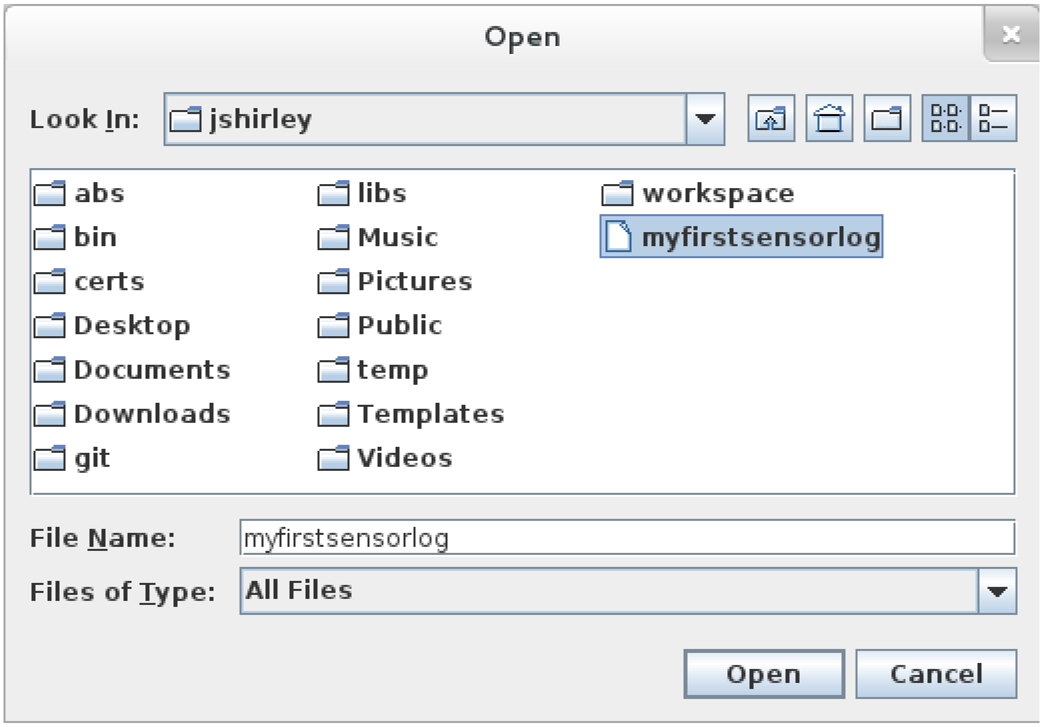
Files of Type: Sensor Logs

Importing Data

To import sensor data the user must simply have a backup file created by exporting sensor data. Once this prerequisite is met, the user can import data by navigating to file->import.



In the next window the user should select their backup file.



Look In: jshirley

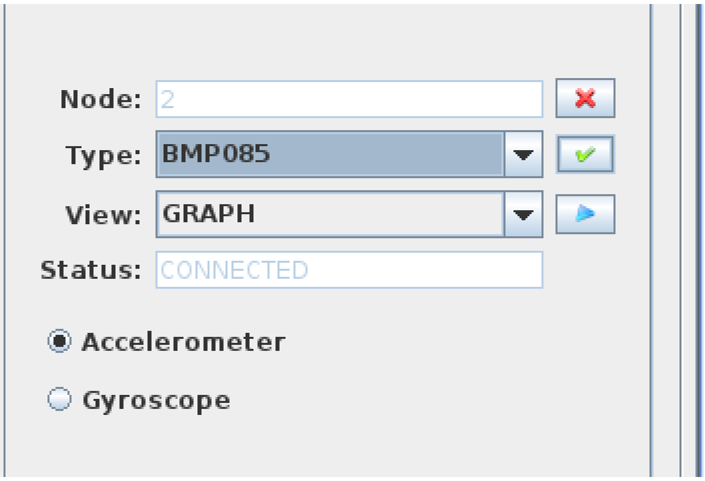
File Name: myfirstsensorlog

Files of Type: All Files

The *Poly_Sense* Monitor will attempt to restore the system to the state it was in when the export command was issued. This includes connecting to the base station and attempting to reconfigure each node to have the specified sensor type.

Changing node configurations on the fly

To change the node's configuration the user must first be connected to a node and have it set as some other configuration. Once this prerequisite is met, the user can change the node's configuration by first pausing any data collection runs and then click the drop down menu on the side bar labeled "Type" and select the sensor type the user would like to configure the node to. Once this is completed the user can send the configuration request by clicking the green checkmark on the right. The node will temporarily disappear, if it does not reappear then the configuration request has failed.



Node: 2
Type: BMP085
View: GRAPH
Status: CONNECTED

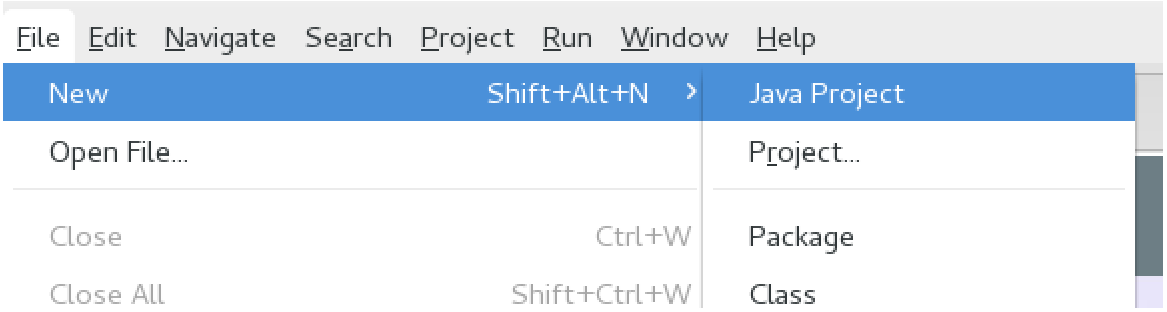
● Accelerometer
○ Gyroscope

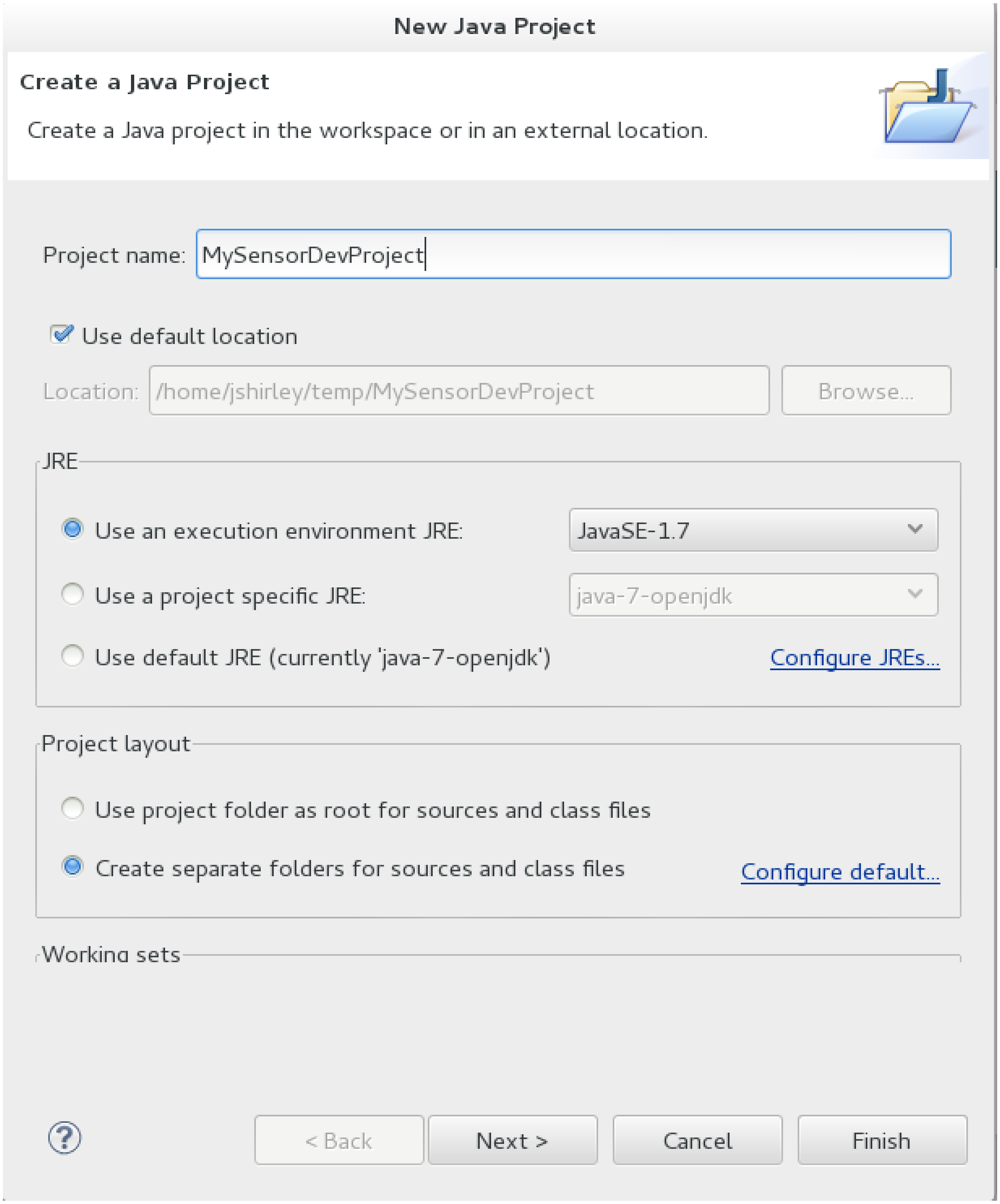
Developing new Sensors

The *Poly_Sense* Monitor provides a system for easily integrating new sensors by creating Java Classes which implement a Sensor interface. It is recommended that the user uses an IDE such as eclipse to do this.

Integrating with Eclipse

Start a new Eclipse project by clicking File->New->Java Project and follow the prompt to name and locate your project.





New Java Project

Create a Java Project

Create a Java project in the workspace or in an external location.

Project name: MySensorDevProject

☑ Use default location

Location: /home/jshirley/temp/MySensorDevProject

JRE

● Use an execution environment JRE: JavaSE-1.7

○ Use a project specific JRE: java-7-openjdk

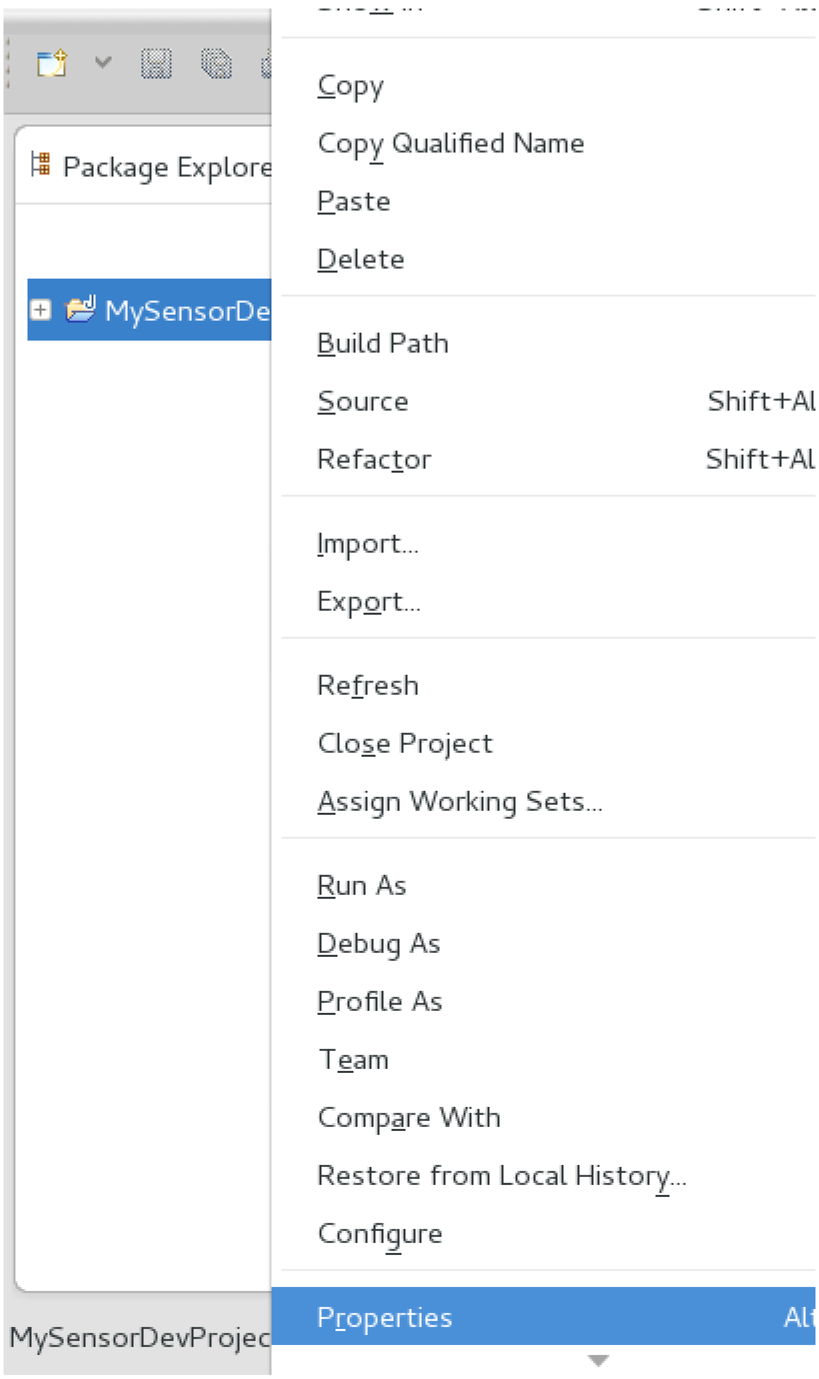
○ Use default JRE (currently 'java-7-openjdk')

Project layout

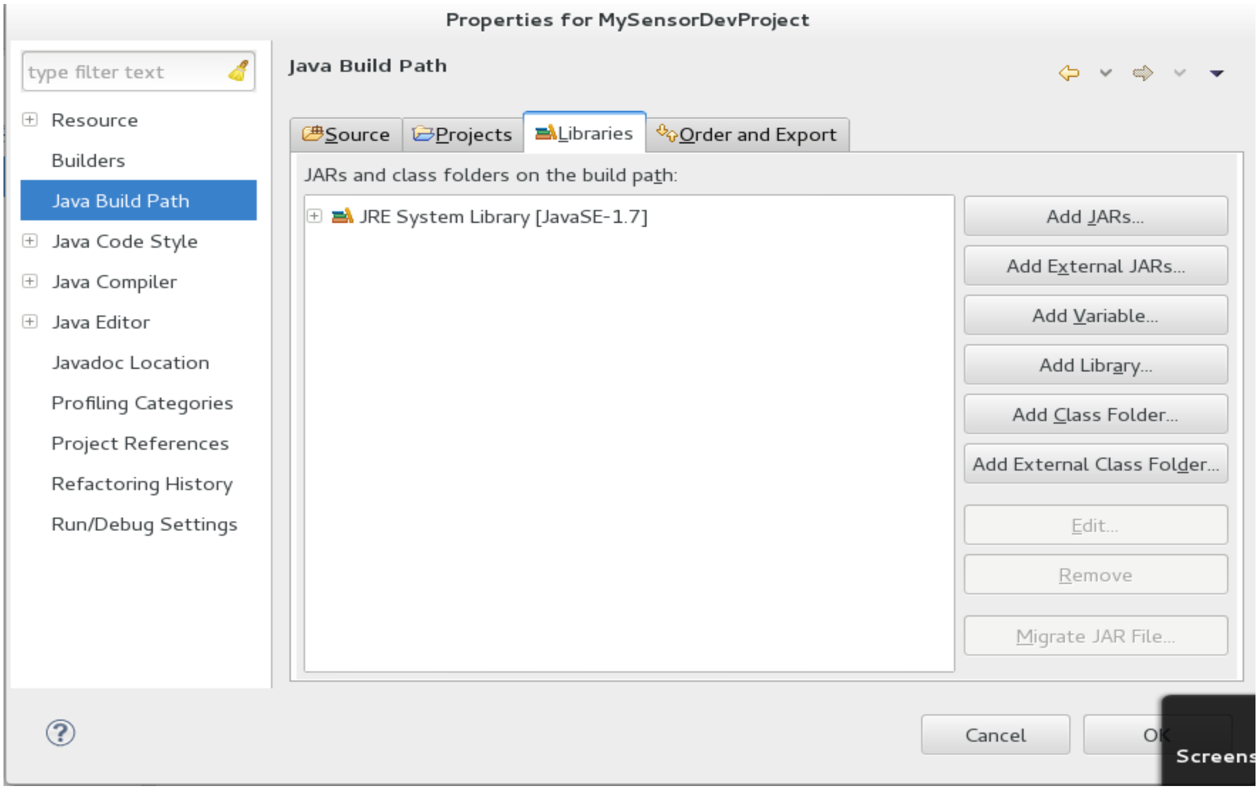
○ Use project folder as root for sources and class files

● Create separate folders for sources and class files

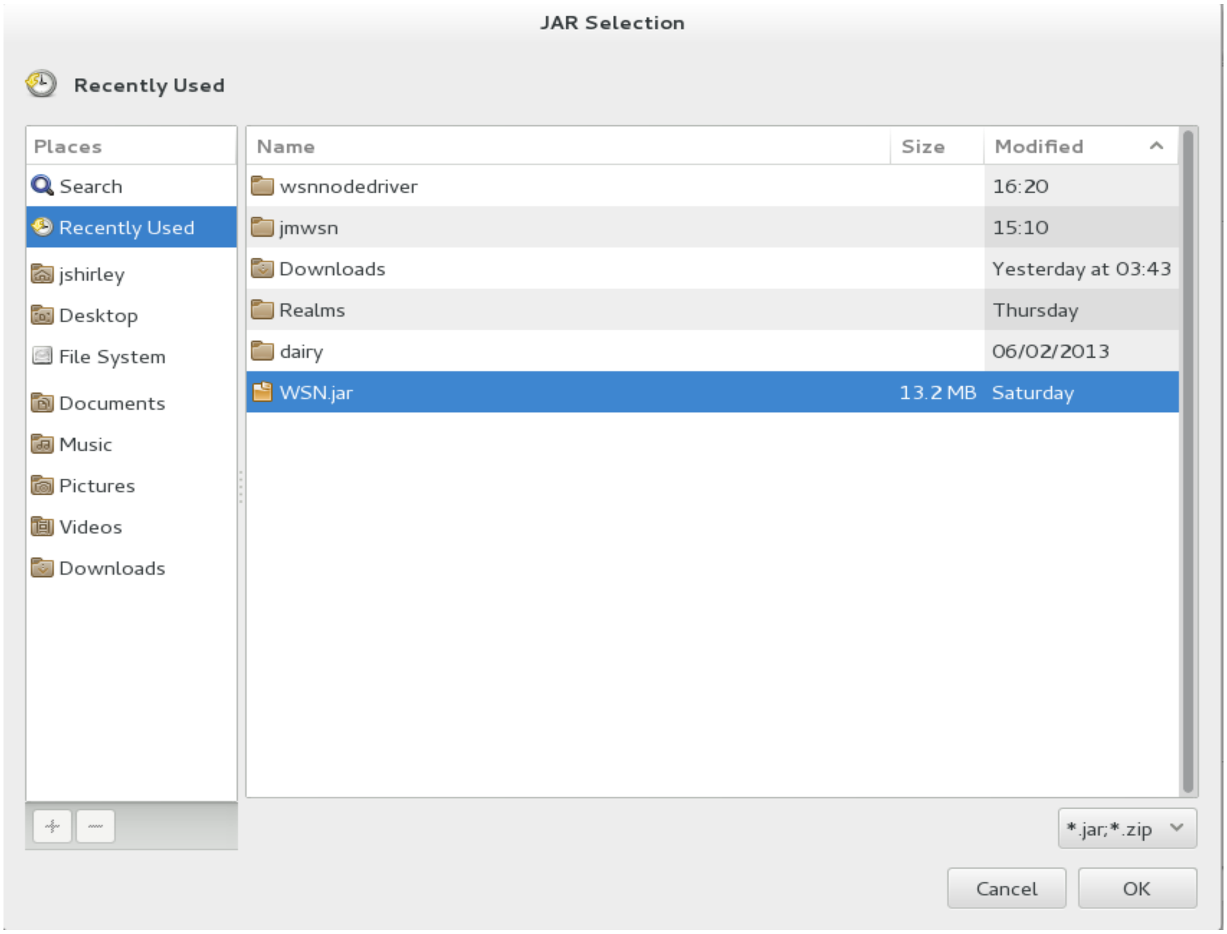
Now right click on your new project and go to properties.



Navigate to Java Build Path and clock on the Libraries tab.

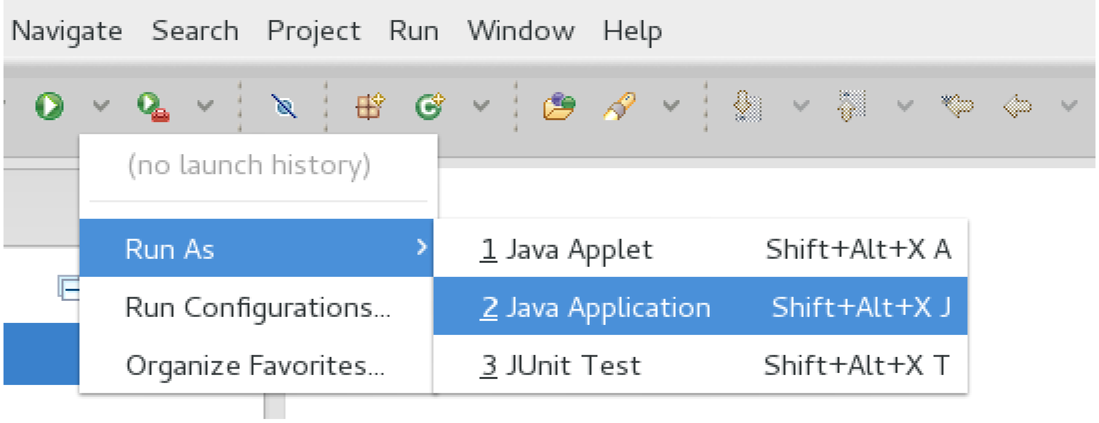


Click “Add External JARs”, select the WSN.jar file. This is the file that you normally use to launch the *Poly_Sense* Monitor.

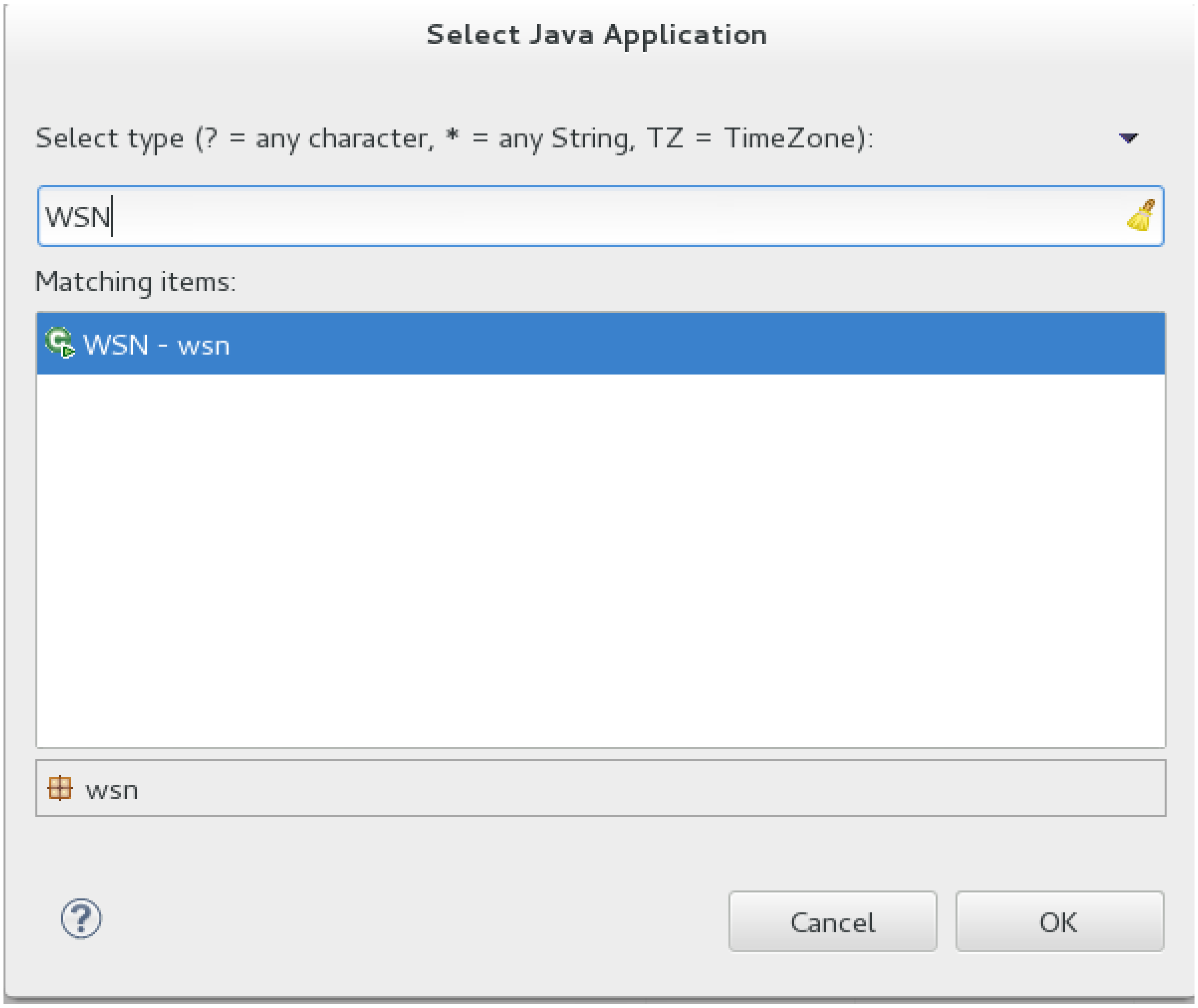


Name	Size	Modified
wsnnodedriver		16:20
jmwsn		15:10
Downloads		Yesterday at 03:43
Realms		Thursday
dairy		06/02/2013
WSN.jar	13.2 MB	Saturday

Close the properties window and then click the expand button next to the green run button near the top. Select Run As->Java Application.



In the next window you must select the class that contains the main method. For the *Poly_Sense* Monitor that is WSN.



Select Java Application

Select type (? = any character, * = any String, TZ = TimeZone):

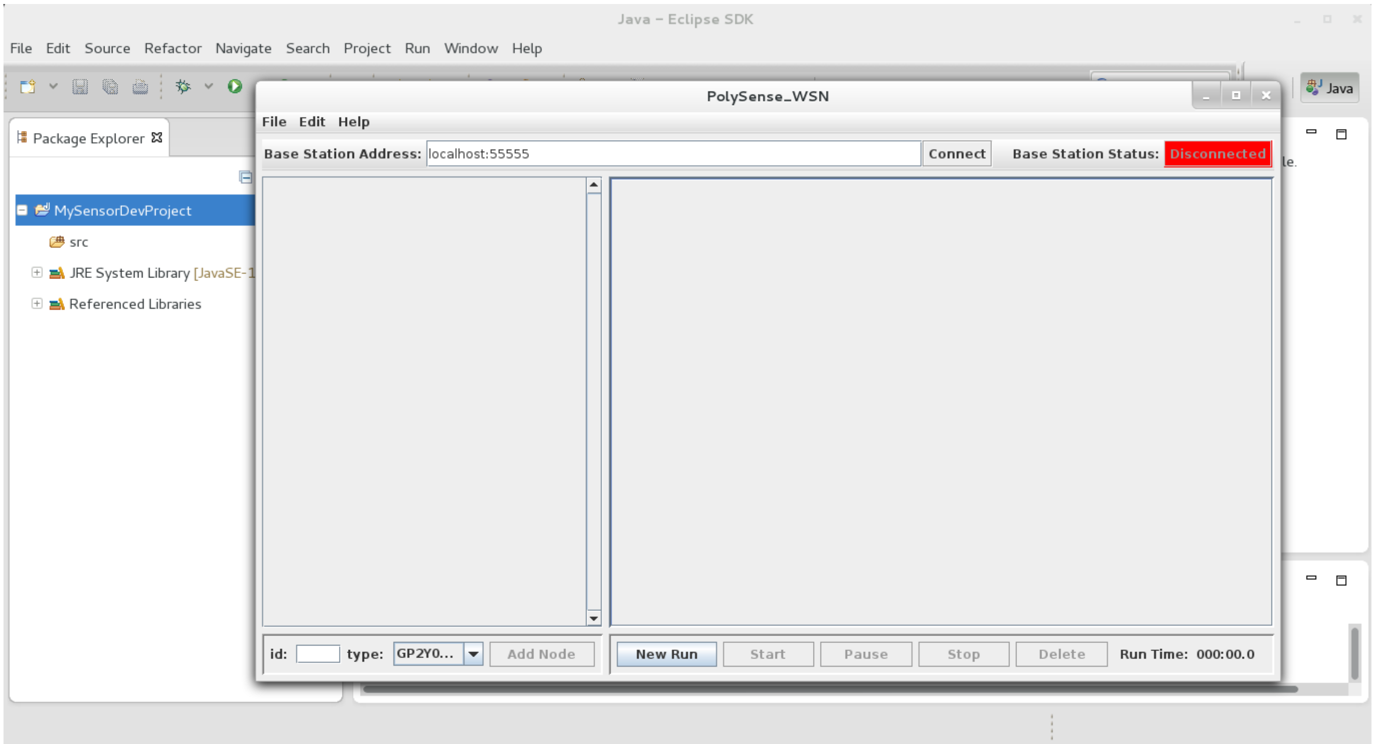
WSN

Matching items:

WSN - wsn

wsn

Your *Poly_Sense* Monitor is now integrated into Eclipse! In the future, you can test your changes and new Sensor Classes by simply pressing the green run button.



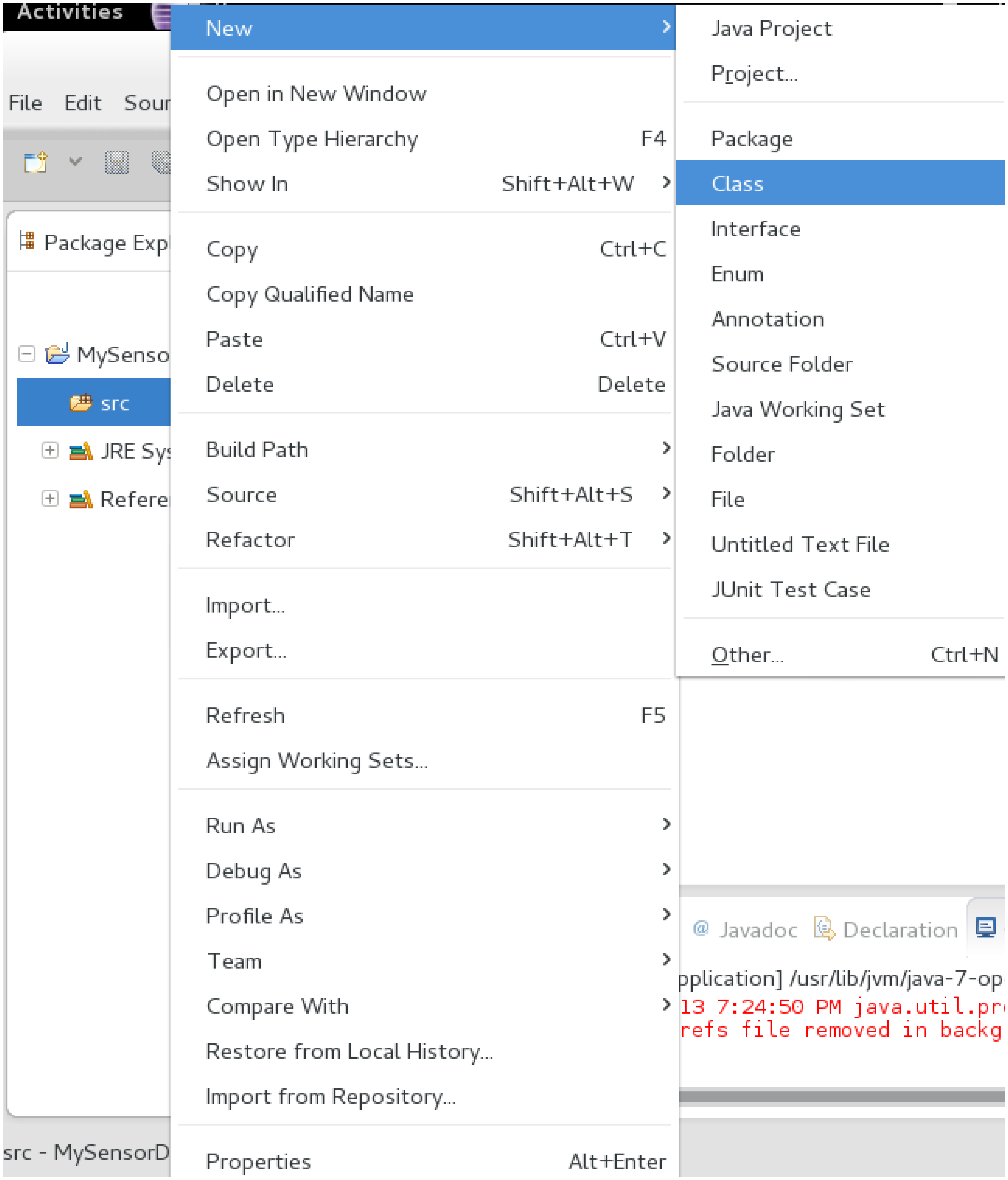
Base Station Address: localhost:55555

id:

Run Time: 000:00.0

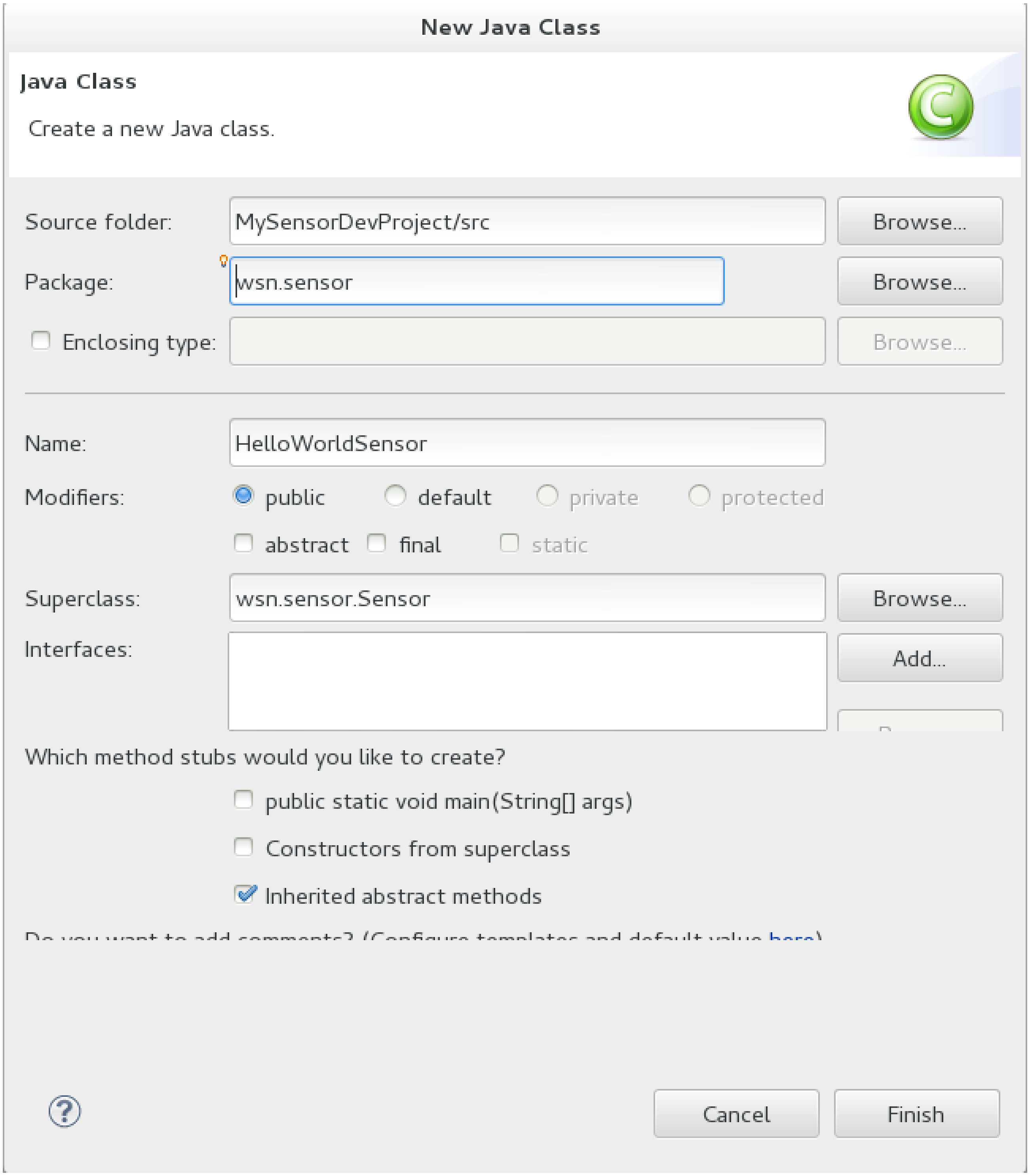
Sensor API

To develop a new Sensor the user must first create a Class that implements the Sensor interface. This can be accomplished in Eclipse by right clicking on the project and navigating to New->Class.



```
pplication] /usr/lib/jvm/java-7-op
13 7:24:50 PM java.util.pr
refs file removed in backg
```

In the next window the user can name and locate the Class. In this window the user should first type “wsn.sensor” into the package field and then click the “Browse...” button next to “Superclass” and type in “wsn.sensor.Sensor”.



New Java Class

Java Class

Create a new Java class.

Source folder: MySensorDevProject/src

Package: wsn.sensor

□ Enclosing type:

Name: HelloWorldSensor

Modifiers: ● public ○ default ○ private ○ protected

□ abstract □ final □ static

Superclass: wsn.sensor.Sensor

Interfaces:

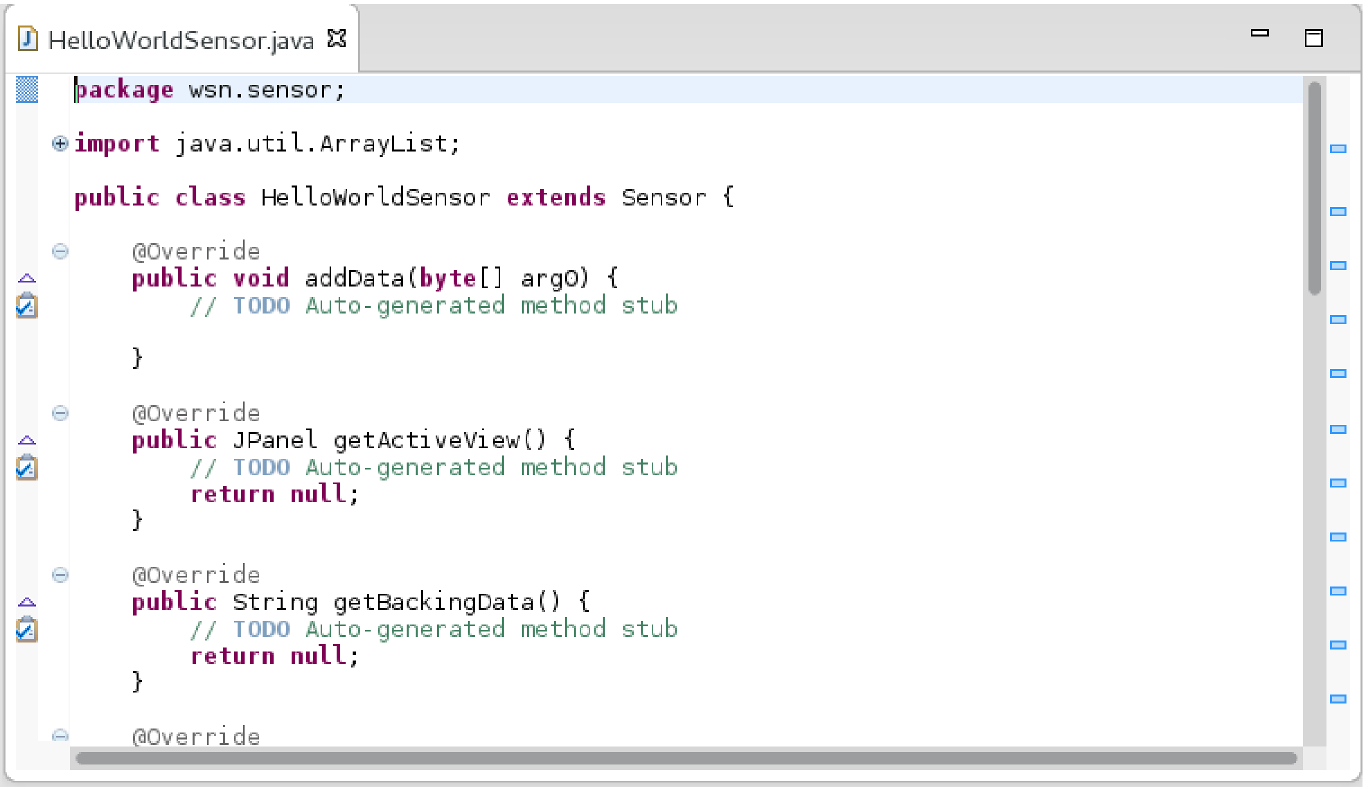
Which method stubs would you like to create?

□ public static void main(String[] args)

□ Constructors from superclass

☑ Inherited abstract methods

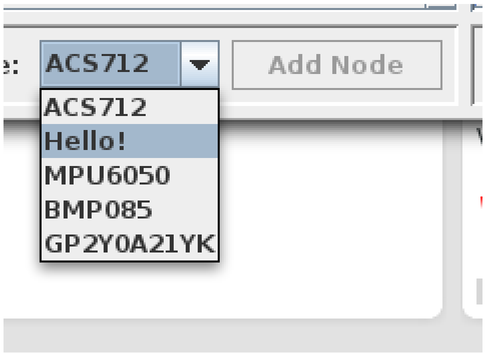
After clicking “Finish”, a new editor will appear with methods your new Sensor must implement.



```
package wsn.sensor;

import java.util.ArrayList;

public class HelloWorldSensor extends Sensor {

    @Override
    public void addData(byte[] arg0) {
        // TODO Auto-generated method stub

    }

    @Override
    public JPanel getActiveView() {
        // TODO Auto-generated method stub
        return null;
    }

    @Override
    public String getBackingData() {
        // TODO Auto-generated method stub
        return null;
    }

    @Override
```

For an explanation of how to implement these classes, refer to appendix E.1. If you simply want to see if you have integrated your project into Eclipse correctly you only need to override the getSensorName() method.

Once you have finished implementing the methods required, run your project. Your Sensor should be available in the drop down menu next to “Add Node”.



Troubleshooting

This section addresses problems a user may encounter while operating the Poly_Sense system.

Host computer cannot add a node:

1. Verify the red LED, D1, on the node is lit
2. Confirm that you have typed in the right settings for the node (i.e. node id and sensor)
3. Verify the USB connection from the base station to the host computer
4. If the host computer still cannot add the node, restart the system by restarting the base station, GUI, server, and all other nodes

Red LED , D1, is not lit:

1. Verify that the battery pack is connected to either the DC barrel jack or the header pins
2. Confirm that the switch is in the "On" position
3. Using a digital multimeter, measure the voltage across TP1 and ground. The voltage should be close to 3.3V
4. If the LED still does not turn on, verify that the battery voltage is close to 3V

No sensor data coming from a node:

1. Verify the red LED, D1, on the node is lit
2. Confirm that a sensor has made a proper connection to the sensor port
3. Check that the GUI displays the proper node address and sensor configuration
4. If the problem persists, restart the system by restarting the base station, GUI, server, and all other nodes

Host computer unable to communicate with base station

1. Verify the USB connection from the base station to the host computer
2. Confirm that the red LED, D1, on the node is lit
3. Ensure the proper FTDI drivers have been installed for the serial-to-USB device
4. If the problem persists, unplug and plug back in the base station and restart the system by restarting the GUI, server, and all other nodes

Host computer unable to connect to server

1. If the server is connected remotely, confirm that any routers that the server machine is running behind are properly configured to forward the port that the server is using to the correct machine.
2. Restart the server
3. If the problem persists, restart the GUI

APPENDIX E: Misc.

E1. *Poly_Sense* Monitor Sensor API

public abstract Sensor getNewInstance();
Because of the nature of Java Reflections, the end user must provide a (non-static) method that returns a new object of their sensor. This can be easily accomplished with a method similar to the one below.

```
public Sensor getNewInstance() {
        return new HelloWorldSensor();
}
```

public abstract String getSensorName();
The end user should create a method that returns the name of the sensor here. What the user names the sensor here is reflected on the *Poly_Sense* Monitor. This should also return the string that the node driver is expecting (Refer to user manual: System Programming) for this Sensor (class) configuration.

public abstract String getStatusString();
The end user should create a method that returns the current status of the sensor here. Examples of this include "Initialized", "Receiving Data", "Configuring", etc. This String is only for appearance and if the end user simply returns an empty String the program will still function normally.

public abstract ArrayList<viewType> getViewTypes();
The end user should create a method that returns a list of views that this Sensor (class) supports. For now we have limited the possible views to graphs, tables, specific, and other. This list could be easily extended to contain even more viewTypes however, during the design process, it was decided that these views were the most important for sensor data. This is reflected on the frontend in the drop down menu located next to "View" on the side panel.

public abstract void setActiveView(viewType view);
The end user should create a method that prepares the Sensor (class) to return a JPanel of the viewType specified in the parameter. For example, if the viewType was "TABLE" then the user may want to create a JPanel that contains a JTable.

public abstract JPanel getActiveView();

The end user should create a method that returns the Sensor's (class) active view as a JPanel. This JPanel would often be a Graph or a Table, but if the viewType was "other" then the user could return something as elaborate as a video feed or text representations. This method affects what appears in the right hand side of the *Poly_Sense* Monitor when the blue arrow is clicked next to the "View" label.

public abstract JPanel getSidePanel();

The end user should create a method that returns a panel with auxiliary information and UI elements that allow for the user to configure elements of the Sensor. For example, the MPU6050 Sensor has 2 collection modes, accelerometer data and gyroscope data. The user may add a UI element that allows the switching between these modes and changes the active view accordingly. This method affects what appears in the side panel when a configured node is added.

public abstract void requestData(NetworkListener network,short address);

The end user should create a method that uses the NetworkListener parameter to request data from the short address parameter. For most sensors this will be the same; first the Sensor (class) should use the sendPacket() method of the network listener to send a packet requesting data from the sensor. The Sensor (class) should then use the expectPacket() method of the network listener to wait for the data to arrive. Once the data has arrived the user must then check if the packet was intended for the Sensor (class) and add the data that arrived. In the future, the NetworkListener would manage checking that the packet was intended for this sensor and call the addData() method accordingly however, due to time constraints, the current implementation requires the user to handle this in their Sensor (class).

Below is an example implementation of this method:

```
network.sendPacket(Network.flags.SENSOR_DATA.getValue(), address, new String(
                                        reqType.getValue() + ""));
                    if (network.expectPacket(Network.flags.SENSOR_DATA.getValue())) {
                              if (NetworkListener.getInstance().getPacketNodeId() ==
address) {
                                        addData(NetworkListener.getInstance().getPacketData(

          Network.flags.SENSOR_DATA.getValue()));
                              }
                    }
```

The hardware used in this system has an unfortunate limitation; each Node/Base Station radio can only be sending or receiving at a given time. If the Base Station were to be sending data when a Node is sending data back the results become unpredictable. Furthermore, because the requestData() method is called from a separate thread and other threads of the *Poly_Sense* Monitor may be using the network as well the user needs to be able to gain exclusive access to the Network. This can be accomplished by calling the getNetworkLock() method of the Globals class. After the user is done using the network then the user must call relaseNetworkLock().

Below is an example of using the network locking functions:

```
Globals.getInstance().getNetworkLock();
// do network work
Globals.getInstance().releaseNetworkLock();
```

public abstract void addData(byte[] bytes);

The end user should create a method that takes the bytes parameter and binds it as sensor data to this Sensor (class). For the Sensors we supported the bytes received will be formatted as a type byte followed by the value.

And example of this could be:

[‘i’,’a’,’e’,’f’,’3’,‘i’,’8’,’o’,’h’,’2’,‘i’,’f’,’y’,’g’,’9’]

Where ‘i’ refers to integer, a signed 32bit number. The following 4 bytes are then interpreted as an integer. Users are free to define their own method of representing sensor data however they are still limited by the 15 byte data size of the packet. This method does not strictly update any UI elements however it is recommended that the end user uses this method to update any Table/Graphs/etc. that are currently being viewed.

public abstract String getBackingData();

The end user should create a method that returns a string representation of the sensor state as well as all of data currently gathered by this Sensor (class). This is used for persisting gathered sensor data across runs and should be in a format that can be loaded back into the Sensor (class). For the supported sensors in this system the getBackingData() method returns a JSON String representing the sensor’s data. It is recommended that users also use JSON to store sensor data but is not required.

public abstract void setData(String data);
The end user should create a method that accepts a String that represents a past Sensor (class) and initialize the Sensor to that state.