**System Design Specification (SPS)**

**for the**

**Thermostat PC-MCU System**

**October 14th, 2015**

**Version 1.0**

Table of Contents

[1 INTRODUCTION 4](#_Toc435382362)

[2 SYSTEM DIAGRAM 4](#_Toc435382363)

[3 SYSTEM CONNECTIONS 5](#_Toc435382364)

[4 SYSTEM PARTS/COMPONENTS 6](#_Toc435382365)

[5 REFERENCES 7](#_Toc435382366)

[6 HOST-PC SOFTWARE 8](#_Toc435382367)

[6.1 Packages 8](#_Toc435382368)

[6.1.1 Package: application 8](#_Toc435382369)

[6.1.2 Package: application.thermostat package 8](#_Toc435382370)

[6.1.3 Package: application.thermostat.crc package 9](#_Toc435382371)

[6.1.4 Package: application.thermostat.db package 9](#_Toc435382372)

[6.1.5 Package: application.thermostat.log package 10](#_Toc435382373)

[6.1.6 Package: application.thermostat.message package 10](#_Toc435382374)

[6.1.7 Package: application.thermostat.message.messages package 11](#_Toc435382375)

[6.1.8 Package: application.thermostat.message.processor package 11](#_Toc435382376)

[6.1.9 Package: application.thermostat.sensors package 12](#_Toc435382377)

[6.1.10 Package: application.thermostat.xml package 12](#_Toc435382378)

[6.2 Classes 13](#_Toc435382379)

[6.2.1 ThermostatGUI.java 13](#_Toc435382380)

[6.2.2 Thermostat.java 14](#_Toc435382381)

[6.2.3 CRCGenerator.java 16](#_Toc435382382)

[6.2.4 TemperatureSensorData.java 17](#_Toc435382383)

[6.2.5 TemperatureSensorRecord.java 18](#_Toc435382384)

[6.2.6 ThermostatLogger.java 18](#_Toc435382385)

[6.2.7 MessageReceiver.java 19](#_Toc435382386)

[6.2.8 MessageSender.java 21](#_Toc435382387)

[6.2.9 Message.java 23](#_Toc435382388)

[6.2.10 MessageType.java 24](#_Toc435382389)

[6.2.11 RequestSetTempDangerLevelMsg.java 25](#_Toc435382390)

[6.2.12 RequestSetTempNormalLevelMsg.java 26](#_Toc435382391)

[6.2.13 RequestGetTempReadingMsg.java 26](#_Toc435382392)

[6.2.14 RequestSetTempWarningLevelMsg.java 27](#_Toc435382393)

[6.2.15 MessageProcessor.java 27](#_Toc435382394)

[6.2.16 AlarmLevel.java 28](#_Toc435382395)

[6.2.17 AlarmSensor.java 29](#_Toc435382396)

[6.2.18 Sensor.java 30](#_Toc435382397)

[6.2.19 SensorType.java 31](#_Toc435382398)

[6.2.20 TemperatureSensor.java 32](#_Toc435382399)

[6.2.21 XMLFileBuilder.java 33](#_Toc435382400)

[6.2.22 XMLFileLoader.java 35](#_Toc435382401)

[7 CLIENT - MCU SOFTWARE 36](#_Toc435382402)

[7.1 Packages 36](#_Toc435382403)

[7.2 Classes 36](#_Toc435382404)

[7.2.1 Thermostat 36](#_Toc435382405)

[7.2.2 AlarmSensor (.h and .cpp) 37](#_Toc435382406)

[7.2.3 CRCGenerator (.h and .cpp) 38](#_Toc435382407)

[7.2.4 MessageProcessor (.h and .cpp) 38](#_Toc435382408)

[7.2.5 MessageReceiver (.h and .cpp) 40](#_Toc435382409)

[7.2.6 MessageSender (.h and .cpp) 41](#_Toc435382410)

[7.2.7 TemperatureSensor (.h and .cpp) 42](#_Toc435382411)

[8 USER INTERACE 43](#_Toc435382412)

[8.1 Settings Tab View 43](#_Toc435382413)

[8.2 Current Tab View 43](#_Toc435382414)

[8.3 History Tab View 44](#_Toc435382415)

[8.4 File Menu Extended 44](#_Toc435382416)

[8.5 Edit Menu Extended 45](#_Toc435382417)

[9 WANTED FeATURES 46](#_Toc435382418)

[10 DESIGN SHORTFALLS/KNOWN ISSUES 46](#_Toc435382419)

[11 SYSTEM REQUIREMENTS 46](#_Toc435382420)

[12 CALCULATIONS 47](#_Toc435382421)

**Table of Figures**

[Table 1: System Pin Connections 5](#_Toc435304187)

[Table 2: Component List 6](#_Toc435304188)

[Table 3: References 7](#_Toc435304189)

# INTRODUCTION

This Personal Computer (PC) – Microcontroller Unit (MCU) System Design Specification (SPS) defines the overall Thermostat application/system design.

From here on out, the term “host” refers to the PC and Java applications(s) running on the PC. The term “Client”, refers to the Thermostat itself, encompassing the embedded electronics.

This document is divided into the following sections:

Section 2 – SYSTEM DIAGRAM

Section 3 – SYSTEM CONNECTIONS

Section 4 – SYSTEM PARTS/COMPONENTS

Section 5 – REFERENCES

Section 6 – HOST-PC SOFTWARE

Section 7 – CLIENT-MCU SOFTWARE

Section 8 – SYSTEM REQUIREMENTS

# SYSTEM DIAGRAM

The diagram below shows the Thermostat system connections between all of the system’s components. Section 3 contains a table showing each connection in detail.

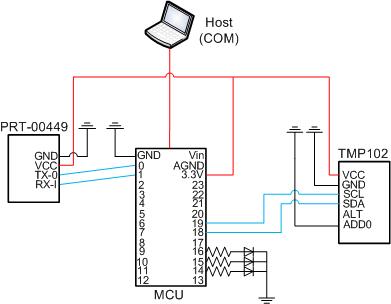


Figure 1: Connection Diagram

# SYSTEM CONNECTIONS

The table below lists all of the connections required for the Thermostat system. It is organized to show point-to-point connections including the component and pin numbers as applicable.

|  |  |  |  |
| --- | --- | --- | --- |
| **Component 1:** | **Connection/Pin:** | **Component 2:** | **Connection/Pin:** |
| PRT-00449 | GND | MCU | GND |
| PRT-00449 | VCC | MCU | 3.3V |
| PRT-00449 | TX-0 | MCU | 0 |
| PRT-00449 | RX-I | MCU | 1 |
| TMP102 | VCC | MCU | 3.3V |
| TMP102 | GND | MCU | GND |
| TMP102 | SCL | MCU | 19 |
| TMP102 | SDA | MCU | 18 |
| TMP102 | ADD0 | MCU | GND |
| MCU | USB | HOST (COM) | USB |
| MCU | 16 | LED (Danger) | Anode |
| MCU | 15 | LED (Warn) | Anode |
| MCU | 14 | LED (Normal) | Anode |
| LED (Danger) | Anode | MCU | GND |
| LED (Warn) | Anode | MCU | GND |
| LED (Normal) | Anode | MCU | GND |

Table 1: System Pin Connections

# SYSTEM PARTS/COMPONENTS

This section shows the prototype breadboard of all components wired together. Table 2 below lists all components that make up the entire Thermostat system.

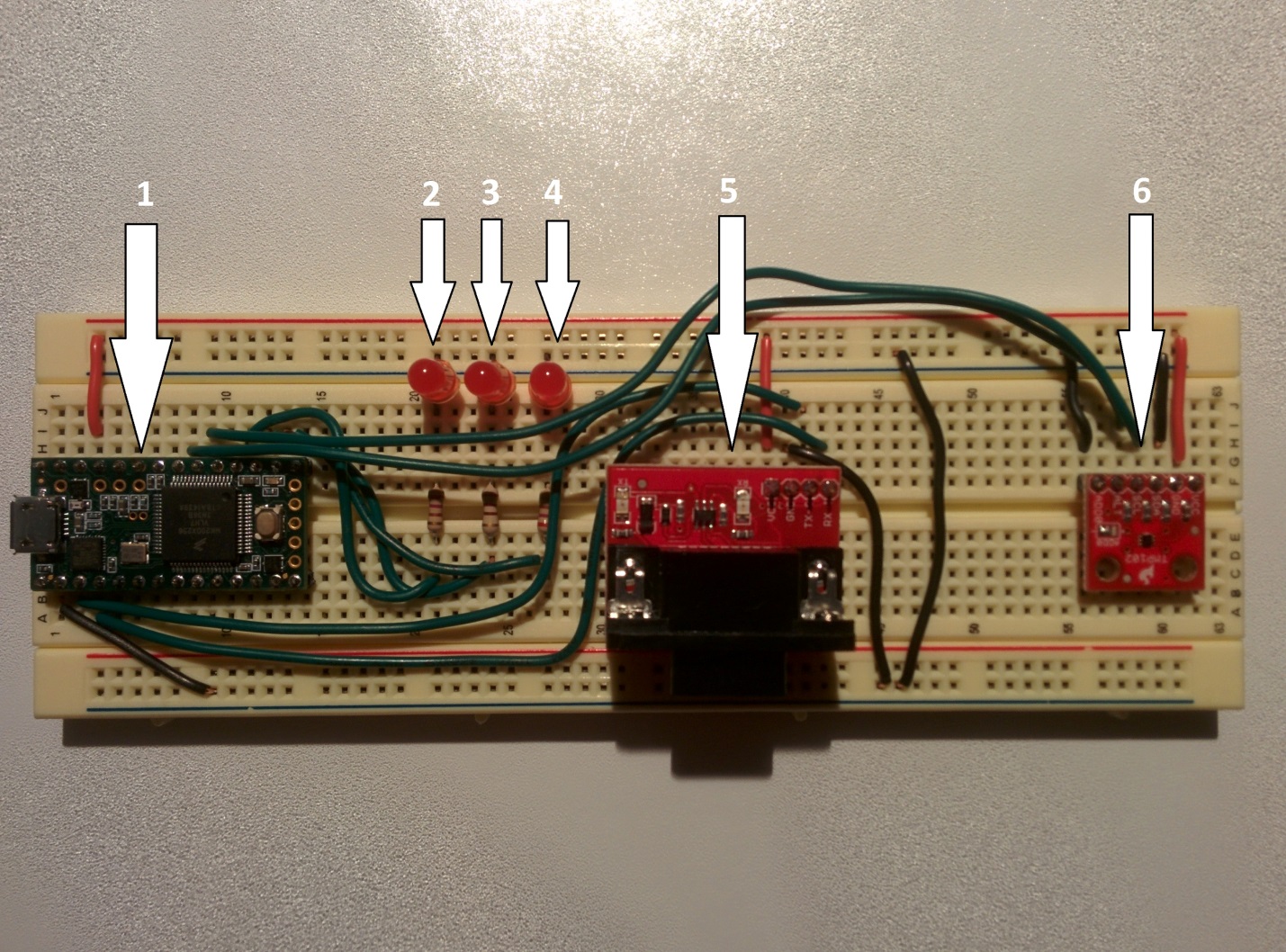


Figure 2: System Picture

|  |  |  |  |
| --- | --- | --- | --- |
| **Reference:** | **Component:** | **Qty:** | **Part Number:** |
| Not Referenced | Dell Desktop Computer | 1 | N/A |
| 1 | Teensy 3.1 | 1 | DEV-12646 |
| 5 | RS-232 Shifter SMD | 1 | PRT-00449 |
| 6 | TMP102 | 1 | SEN-11931 |
| Not Referenced | USB Cable | 1 | N/A |
| Not Referenced | USB-to-RS232 Cable | 1 | N/A |
| Not Referenced | Wires | Many | N/A |
| 4 | Normal Indictor LED | 3 | COM-09590 |
| 3 | Warning Indicator LED | 1 | COM-09590 |
| 2 | Danger Indicator LED | 1 | COM-09590 |
| N/A | Resistor (270 ohm) | 3 | Unavailable |

Table 2: Component List

# REFERENCES

The table below lists the difference references that were used for this project.

|  |  |
| --- | --- |
| **Document Name:** | **Version Number:** |
| Embedded Systems Task Specification | N/A |
| Interface Control Document (ICD) for the Thermostat PC-MCU Serial Messaging Protocol | 1.0 |

Table 3: References

# HOST-PC SOFTWARE

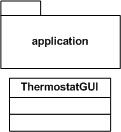
This section describes the overall design of the Host PC, Java-based Thermostat application. The PC application contains both a front-end Graphical User Interface (GUI) as well as the back-end control software for communication with the client.

## Packages

This section defines the various packages that make up the Thermostat application. It gives a brief description for the different packages as well as visually depicting each class that is contained in the packages. Section 3.2 gives a more detailed description of each class, its responsibilities, attributes and methods contained within, and finishes with design notes and architectural talking points.

### Package: application

The application package contains the Thermostat system GUI code along with the CSS file for the Thermostat application. In the current version of software, the CSS file is not currently being used to the extent possible.

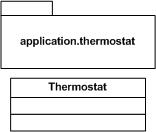


The application package contains the following classes/files:

* ThermostatGUI.java
* application.css

### Package: application.thermostat package

The application.thermostat package contains the primary Thermostat back-end code that runs on the host laptop/computer.

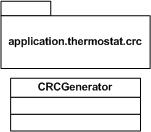


The application.thermostat package contains the following classes/files:

* Thermostat.java

### Package: application.thermostat.crc package

The application.thermostat.crc package contains necessary code for the calculation of the Cyclic Redundancy Check (CRC), which becomes appended to each serial message that is sent from the host computer to the Thermostat system.

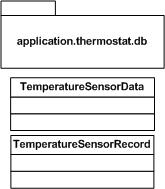


The application.thermostat.crc package contains the following classes/files:

* CRCGenerator.java

### Package: application.thermostat.db package

The application.thermostat.db package contains all of the necessary code for recording all temperature measurements.

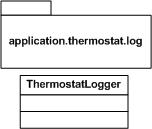


The application.thermostat.db package contains the following classes/files:

* TMP102SensorReadingHistory.java
* TMP102SensorRecord.java

### Package: application.thermostat.log package

The application.thermostat.log package contains code for logging various messages to a file during runtime. Although not specified in the requirements for the system, it has been implemented for both debugging purposes as well as applicability to additional features in the future.

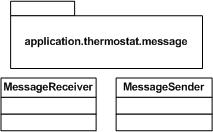


The application.thermostat.log package contains the following classes/files:

* ThermostatLogger.java

### Package: application.thermostat.message package

The application.thermostat.message package contains code for handling the serial port connections for both receiving and sending messages between the PC and the MCU

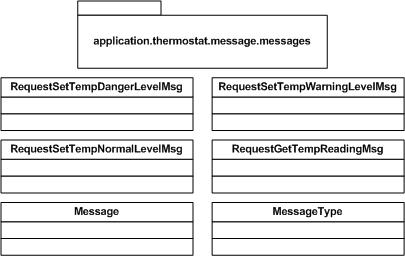


The application.thermostat.message package contains the following classes/files:

* MessageReciever.java
* MessageSender.java

### Package: application.thermostat.message.messages package

The application.thermostat.message.messages package contains all necessary code for supporting the different type of messages which are sent/received between the PC and MCU.

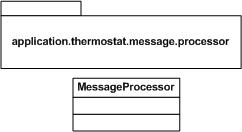


The application.thermostat.message.messages package contains the following classes/files:

* Message.java
* MessageType.java
* TemperatureDangerLevelMessage.java
* TemperatureNormalLevelMessage.java
* TemperatureReadingRequestMessage.java
* TemperatureWarningLevelMessage.java

### Package: application.thermostat.message.processor package

The application.thermostat.message.processor package contains the necessary code to process incoming messages. Although there is currently only one message to be processed on the PC application, the code in the package was developed for scalability for future feature implementation and/or code reuse between other applications.

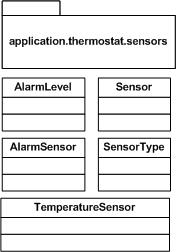


The application.thermostat.message.processor package contains the following classes/files:

* MessageProcessor.java

### Package: application.thermostat.sensors package

The application.thermostat.sensors package contains the various classes that support the system’s sensors and/or electronics.

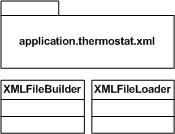


The application.thermostat.sensors package contains the following classes/files:

* AlarmLevel.java
* AlarmSensor.java
* Sensor.java
* SensorType.java
* TemperatureSensor.java

### Package: application.thermostat.xml package

The application.thermostat.xml package contains the code related to opening and saving the application’s configuration settings file(s).



The application.thermostat.xml package contains the following classes/files:

* XMLFileBuilder
* XMLFileLoader

## Classes

This section describes all of the individual classes that make up the PC (host) side of the Thermostat system.

### ThermostatGUI.java

#### Description:

The ThermostatGUI class is the front-end Java class that builds the Java GUI running on the host computer.

#### UML Architecture

#### C:\Users\DeveloperMain\Desktop\NRL Deliverables\uml class diagrams\ThermostatGUI.jpg

#### Design Details/Architecture:

No additional notes.

#### Requirements Traceability

|  |  |
| --- | --- |
| **Requirement #:** | **Requirement Text:** |
| SRS0006 | The GUI shall display a graph of temperature vs. time: last 10 minutes, last 60 minutes, last 24 hours, last week |
| SRS0007 | The software shall load/save the user inputs from/to an XML file on startup/shutdown or on demand. |
| SRS0008 | The software shall provide an export function of the temperature history data to a CSV file, such that MS Excel can ingest the file and plot it. |
| SRS0009 | The GUI shall be coded in Java, using FX for GUI elements. |
| SRS0010 | The GUI shall implement a standard menu bar (minimum: File, Edit, Help) |
| SRS0011 | The GUI shall implement a tabbed interface (Settings, Current, History) |

### Thermostat.java

#### Description:

The Thermostat class is the primary object of the for the host-side application. The Thermostat class is responsible for the following:

* Creating/Managing the serial port connection to the system
* Managing requests to the microcontroller:
  + Temperature sensor readings
  + Alarm indicator management
* Validation/Management of the alarm indicator per SRS0002 and its sub-requirements
* Provide interface to the front-end GUI for at least the following operations:
  + Starting sensor measurements
  + Stopping sensor measurements
  + Exporting CSV Data
  + Setting different sensor refresh rates (currently not implemented on front-end)
  + Set/Get methods for the configured high threshold value
  + Set/Get methods for the configured low threshold value
  + Set/Get methods for adjusting the temperature units (Farenheit/Celcius)
  + Get method for system logger (not used currently in this release)

#### UML Architecture

#### C:\Users\DeveloperMain\Desktop\NRL Deliverables\uml class diagrams\Thermostat.jpg

#### Design Details/Architecture:

The Thermostat class contains a LinkedList of Sensor objects. Although the full functionality of this is not realized in the current version of the Thermostat system, it is planned that future implementations can make use of the early modular design. By using a List of Sensors, future sensors can easily be added simply by making use of the XML configuration settings file. Some code is needed to properly handle the sensor request logic and searching through the list, but will make the possibility of handling additional electronics in future projects.

As noted in the section below, the Thermostat.java class supports multiple requirements of the Thermostat system due to the large amount of methods that are provided.

#### Requirements Traceability

|  |  |
| --- | --- |
| **Requirement #:** | **Requirement Text:** |
| SRS0001 | The system shall record temperature readings from the sensor at a 1 Hz rate. |
| SRS0002 | The system shall activate one of three LED indicators according to the current temperature and two threshold conditions |
| SRS0002.1 | The system shall activate the Danger (RED) indicator when the current temperature is at or above the HIGH\_THRESHOLD condition. |
| SRS0002.2 | The system shall activate the Warning (YELLOW) indicator when the current temperature is above the LOW\_THRESHOLD condition but below the HIGH\_THRESHOLD condition. |
| SRS0002.3 | The system shall activate the Normal (GREEN) indicator when the current temperature is at or below the LOW\_THRESHOLD condition. |
| SRS0003 | The system shall communicate with the PC/Laptop over a Serial link, with a messaging protocol of your own design. |
| SRS0004 | The GUI shall accept two threshold settings (HIGH\_THRESHOLD, LOW\_THRESHOLD) from the user and validate that HIGH\_THRESHOLD > LOW\_THRESHOLD. |
| SRS0005 | The GUI shall display the current temperature in user-selectable units of Fahrenheit or Celsius. |
| SRS0007 | The software shall load/save the user inputs from/to an XML file on startup/shutdown or on demand. |
| SRS0008 | The software shall provide an export function of the temperature history data to a CSV file, such that MS Excel can ingest the file and plot it. |

### CRCGenerator.java

#### Description:

The CRCGenerator class holds the responsibility of creating the 16-bit,, CRC-CCITT checksum which is added to the serial messages that are sent from the host to the thermostat.

The class currently contains three differents methods, of which only one is being utilized for the Thermostat system. They are all 16 bit variations of the CRC-CCITT checksum with varying polynomials and preset values. In future iterations of the system, other methods may be added to provide various-size checksums for message integrity across multiple different mediums.

#### UML Architecture

#### C:\Users\DeveloperMain\Desktop\NRL Deliverables\uml class diagrams\CRCGenerator.jpg

#### Details:

For the Thermostat system, a 16-bit CRC is being used. By using a 16 bit CRC, it gives some flexibility in keeping the message sizes small but still add a significant message integrity check for the small data payload that makes up the entire message.

The 16-bit CRC CCITT was chosen as there are multiple references for verifying that values were correct. For this project, the following website was used: <http://www.lammertbies.nl/comm/info/crc-calculation.html> for validating the CRC implementation.

#### Requirements Traceability

|  |  |
| --- | --- |
| **Requirement #:** | **Requirement Text:** |
| SRS0003 | The system shall communicate with the PC/Laptop over a Serial link, with a messaging protocol of your own design. |

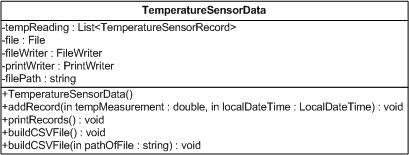
### TemperatureSensorData.java

#### Description:

The TemperatureSensorData class represents the data store for the temperature records. It contains a List of TemperatureSensorRecord objects which gets populated each time that a new measurement is received from the Thermostat.

In addition to keeping the list of temperature sensor measurements, it also provides the appropriate methods to create the required Comma Separated Value (CSV) file.

#### UML Architecture



#### Design Details/Architecture:

To keep the project simpler, a full database was not chosen for use. In a production-level system, a real database would be more ideal.

#### Requirements Traceability

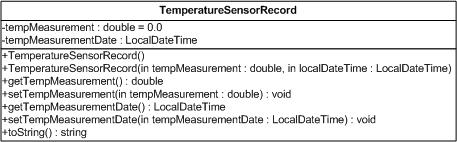
|  |  |
| --- | --- |
| **Requirement #:** | **Requirement Text:** |
| SRS0008 | The software shall provide an export function of the temperature history data to a CSV file, such that MS Excel can ingest the file and plot it. |

### TemperatureSensorRecord.java

#### Description:

The TemperatureSensorRecord class represents a single temperature sensor measurement recording. It contains the temperature measurement and the time at which it was taken.

#### UML Architecture



#### Design Details/Notes:

No additional details or notes.

#### Requirements Traceability

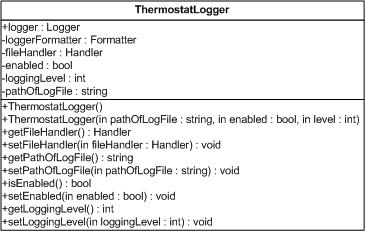
|  |  |
| --- | --- |
| **Requirement #:** | **Requirement Text:** |
| SRS0008 | The software shall provide an export function of the temperature history data to a CSV file, such that MS Excel can ingest the file and plot it. |

### ThermostatLogger.java

#### Description:

The ThermostatLogger class is part of the Thermostat project but it is currently not being used within the project itself. It was built to facilitate future implemented features.

#### UML Architecture



#### Design Details/Architecture:

At the time of writing this document, the ThermostatLogger class is an unfinished class. It will eventually be a Singleton class which all other application objects can use to log relevant messages to a file for debugging or general logging.

#### Requirements Traceability

|  |  |
| --- | --- |
| **Requirement #:** | **Requirement Text:** |
| NA | NA |

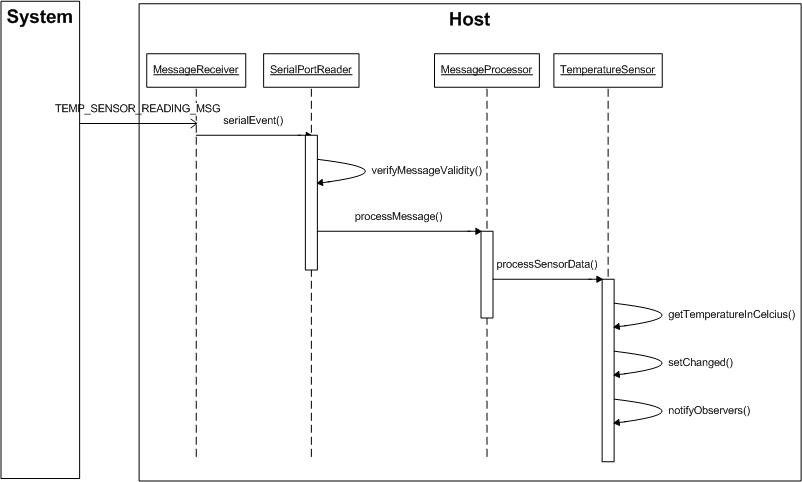
### MessageReceiver.java

#### Description:

The MessageReceiver class is responsible for setting up the serial port for communication between the host computer and the Thermostat system. It contains a class called SerialPortReader that implements the SerialPortEventListener class.

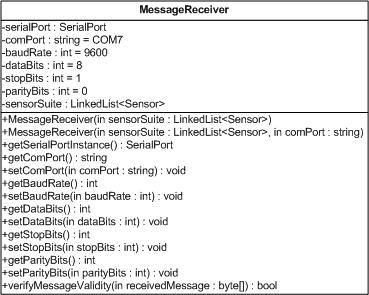
Upon instantiation of the MessageReceiver object, the serial port is setup and a SerialPortReader object is created and added to the configured serial port as an EventListener.

Whenever a message is received from the Thermostat system over the serial connection, it is first received in by the MessageReciever object. It performs a check of the message’s CRC value against what it should be to verify that the message is valid and if so, passes the received message off to the MessageProcessor. Within the processMessage() method of the MessageProcessor, it checks the MessageType byte of the message. Based upon the type of message, it passes it to the appropriate processSensorData() method of the sensor. The basic flow of actions is shown below.



In more generic terms, using the LinkedList in the architecture, multiple sensors can be added in future revisions and the TemperatureSensor object in the figure above could be replaced by Sensor, since all sensors/electronics should subclass the Sensor class. By doing so, all sensors will have their own implementations of the processSensorData() and requestSensorData() methods.

#### Class UML Architecture



#### Design Details/Architecture:

No additional information.

#### Requirements Traceability

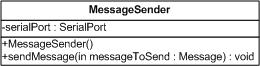
|  |  |
| --- | --- |
| **Requirement #:** | **Requirement Text:** |
| SRS0001 | The system shall record temperature readings from the sensor at a 1 Hz rate. |
| SRS0003 | The system shall communicate with the PC/Laptop over a Serial link, with a messaging protocol of your own design. |

### MessageSender.java

#### Description:

The MessageSender class is responsible for sending out messages from the host computer to the Thermostat system. It provides a single static method, sendMessage(), which handles writing the message bytes onto the serial port.

#### Class UML Architecture

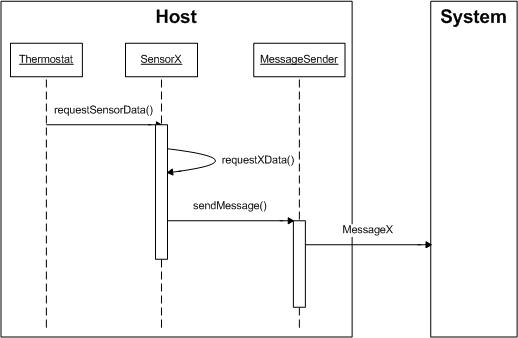


#### Design Details and Architecture:

As stated in the Description section above, the MessageSender class contains a method called sendMessage().

Each Sensor object uses this in its requestSensorData() method. All sensors which subclass the Sensor class contain their own implementation of the requestSensorData() method based upon the data that is important for that sensor or component. To obtain the data from the sensor/component, it currently does and should always should use the sendMessage() method of the MessageSender object.

The basic sequence of sending a message from the host to the Thermostat system is shown in the figure below.



Currently, in this version of the host application, the Thermostat object is the controlling object for all requests to get or set data on the Thermostat system. Upon wanting temperature data or setting the alarm indicator, the Thermostat object calls the appropriate sensor’s requestSensorData() method. Inside of the SensorX (AlarmSensor or TemperatureSensor for this release), one or more private methods are provided referred to here as requestXData(). For the TemperatureSensor, there is the requestTemperature() method, and the AlarmSensor, requestAlarmLow(), requestAlarmMed(), and requestAlarmHigh() methods. Each of these methods make a call to the MessageSender’s sendMessage() method to send the appropriate data request out of the serial port. Once the Thermostat system receives the message, it processes it, and if appropriate, will send out the requested data back to the host where it will be received by the MessageReceiver object.

In future baselines of the software, the ideal solution would be to create a timed list to add messages to and send them out there. In this version of the software, the sendMessage() method of the MessageSender object is contained using request methods from the Thermostat object and should only be used there to avoid any type of potential conflictions with using the serial port.

#### Requirements Traceability

|  |  |
| --- | --- |
| **Requirement #:** | **Requirement Text:** |
| SRS0001 | The system shall record temperature readings from the sensor at a 1 Hz rate. |
| SRS0002 | The system shall activate one of three LED indicators according to the current temperature and two threshold conditions |
| SRS0002.1 | The system shall activate the Danger (RED) indicator when the current temperature is at or above the HIGH\_THRESHOLD condition. |
| SRS0002.2 | The system shall activate the Warning (YELLOW) indicator when the current temperature is above the LOW\_THRESHOLD condition but below the HIGH\_THRESHOLD condition. |
| SRS0002.3 | The system shall activate the Normal (GREEN) indicator when the current temperature is at or below the LOW\_THRESHOLD condition. |
| SRS0003 | The system shall communicate with the PC/Laptop over a Serial link, with a messaging protocol of your own design. |

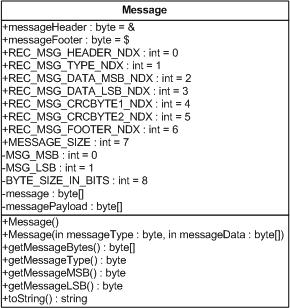
### Message.java

#### Description:

The Message class is the superclass that represents a single message that can be sent from the host to the client or vice versa. All current messages subclass the Message class.

The Message class provides the concrete structure for each message that is sent or received. It includes the byte array for the message as well as the message header, footer, and various indexes for grabbing specific portions of the message.

#### UML Architecture



#### Design Details/Architectural Notes:

As stated above, all messages should subclass the Message class, as it provides all of the necessary data to construct the messages.

#### Requirements Traceability

|  |  |
| --- | --- |
| **Requirement #:** | **Requirement Text:** |
| SRS0001 | The system shall record temperature readings from the sensor at a 1 Hz rate. |
| SRS0002 | The system shall activate one of three LED indicators according to the current temperature and two threshold conditions |
| SRS0002.1 | The system shall activate the Danger (RED) indicator when the current temperature is at or above the HIGH\_THRESHOLD condition. |
| SRS0002.2 | The system shall activate the Warning (YELLOW) indicator when the current temperature is above the LOW\_THRESHOLD condition but below the HIGH\_THRESHOLD condition. |
| SRS0002.3 | The system shall activate the Normal (GREEN) indicator when the current temperature is at or below the LOW\_THRESHOLD condition. |
| SRS0003 | The system shall communicate with the PC/Laptop over a Serial link, with a messaging protocol of your own design. |

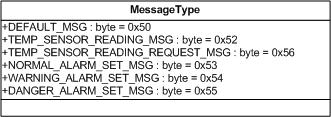
### MessageType.java

#### Description:

The MessageType class contains multiple static fields that represent each individual message type. Each message that the Thermostat system uses contains one of these fields in the second byte of the message byte array.

For more information, please refer to the ICD.

#### UML Architecture



#### Design Details/Architecture:

The MessageType byte is what the host and client keys off of to process the message. The value in this field identifies what type of processing needs to be done and by what sensor.

#### Requirements Traceability

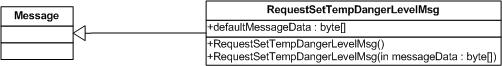
|  |  |
| --- | --- |
| **Requirement #:** | **Requirement Text:** |
| SRS0001 | The system shall record temperature readings from the sensor at a 1 Hz rate. |
| SRS0002 | The system shall activate one of three LED indicators according to the current temperature and two threshold conditions |
| SRS0002.1 | The system shall activate the Danger (RED) indicator when the current temperature is at or above the HIGH\_THRESHOLD condition. |
| SRS0002.2 | The system shall activate the Warning (YELLOW) indicator when the current temperature is above the LOW\_THRESHOLD condition but below the HIGH\_THRESHOLD condition. |
| SRS0002.3 | The system shall activate the Normal (GREEN) indicator when the current temperature is at or below the LOW\_THRESHOLD condition. |
| SRS0003 | The system shall communicate with the PC/Laptop over a Serial link, with a messaging protocol of your own design. |

### RequestSetTempDangerLevelMsg.java

#### Description:

The RequestSetTempDangerLevelMsg is the message used for setting the Danger (Temperature is at or above the value of HIGH\_THRESHOLD) indicator of the Thermostat system. It is sent from the host to the client only.

#### UML Architecture



#### Design Details/Architecture Notes:

No additional content at this time.

#### Requirements Traceability

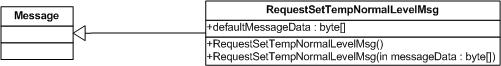
|  |  |
| --- | --- |
| **Requirement #:** | **Requirement Text:** |
| SRS0002 | The system shall activate one of three LED indicators according to the current temperature and two threshold conditions. |
| SRS0002.1 | The system shall activate the Danger (RED) indicator when the current temperature is at or above the HIGH\_THRESHOLD condition. |

### RequestSetTempNormalLevelMsg.java

#### Description:

The RequestSetTempNormalLevelMsg is the message used for setting the Normal (Temperature is at or below the value of LOW\_THRESHOLD) indicator of the Thermostat system. It is sent from the host to the client.

#### UML Architecture



#### Details:

No additional content at this time.

#### Requirements Traceability

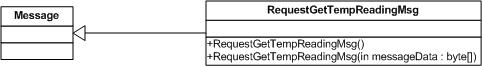
|  |  |
| --- | --- |
| **Requirement #:** | **Requirement Text:** |
| SRS0002 | The system shall activate one of three LED indicators according to the current temperature and two threshold conditions. |
| SRS0002.3 | The system shall activate the Normal (GREEN) indicator when the current temperature is at or below the LOW\_THRESHOLD condition. |

### RequestGetTempReadingMsg.java

#### Description:

The RequestGetTempReadingMsg is the message used for requesting a temperature reading from the client. It is sent from the host to the client.

#### UML Architecture



#### Details:

No additional content at this time.

#### Requirements Traceability

|  |  |
| --- | --- |
| **Requirement #:** | **Requirement Text:** |
| SRS0001 | The system shall record temperature readings from the sensor at a 1 Hz rate. |

### RequestSetTempWarningLevelMsg.java

#### Description:

The RequestSetTempWarningLevelMsg is the message used for setting the Warning (Temperature is above the value for the LOW\_THRESHOLD but below the value for HIGH\_THRESHOLD) indicator of the Thermostat system. It is sent from the host to the client.

#### UML Architecture

#### C:\Users\DeveloperMain\Desktop\NRL Deliverables\uml class diagrams\RequestSetTempWarningLevelMsg.jpg

#### Details:

No additional content at this time.

#### Requirements Traceability

|  |  |
| --- | --- |
| **Requirement #:** | **Requirement Text:** |
| SRS0002 | The system shall activate one of three LED indicators according to the current temperature and two threshold conditions. |
| SRS0002.2 | The system shall activate the Warning (YELLOW) indicator when the current temperature is above the LOW\_THRESHOLD condition but below the HIGH\_THRESHOLD condition. |

### MessageProcessor.java

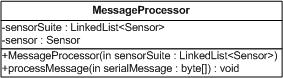
#### Description:

The MessageProcessor class is used to process the message after it has been validated by the MessageReceiver object.

The MessageProcessor class contains one primary method called processMessage(). This method examines the MessageType field of the message and, depending on the value, hands the message off to the appropriate sensor for action.

The Thermostat system currently only makes use of one message type (TEMP\_SENSOR\_READING\_MSG), however the logic has been implemented with this architecture to support future features and processing.

#### UML Architecture



#### Details:

As indicated above, all new message types should be examined by the MessageProcessor object, and then handed off to the appropriate Sensor object to be handled under this architecture.

#### Requirements Traceability

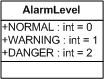
|  |  |
| --- | --- |
| **Requirement #:** | **Requirement Text:** |
| SRS0001 | The system shall record temperature readings from the sensor at a 1 Hz rate. |
| SRS0003 | The system shall communicate with the PC/Laptop over a Serial link, with a messaging protocol of your own design. |

### AlarmLevel.java

#### Description:

The AlarmLevel class provides three static integer values that the Thermostat object uses for comparing the current alarm level to the threshold values.

#### UML Architecture



#### Details:

No additional content at this time.

#### Requirements Traceability

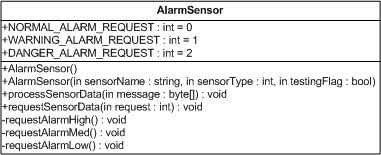
|  |  |
| --- | --- |
| **Requirement #:** | **Requirement Text:** |
| SRS0002 | The system shall activate one of three LED indicators according to the current temperature and two threshold conditions |
| SRS0002.1 | The system shall activate the Danger (RED) indicator when the current temperature is at or above the HIGH\_THRESHOLD condition. |
| SRS0002.2 | The system shall activate the Warning (YELLOW) indicator when the current temperature is above the LOW\_THRESHOLD condition but below the HIGH\_THRESHOLD condition. |
| SRS0002.3 | The system shall activate the Normal (GREEN) indicator when the current temperature is at or below the LOW\_THRESHOLD condition. |

### AlarmSensor.java

#### Description:

The AlarmSensor class represents the interface to the alarm indicators. Subclassing the Sensor class, it contains the necessary methods to communicate with each of three LEDs.

#### UML Architecture



#### Details:

No additional content at this time.

#### Requirements Traceability

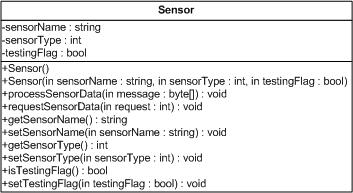
|  |  |
| --- | --- |
| **Requirement #:** | **Requirement Text:** |
| SRS0002 | The system shall activate one of three LED indicators according to the current temperature and two threshold conditions |
| SRS0002.1 | The system shall activate the Danger (RED) indicator when the current temperature is at or above the HIGH\_THRESHOLD condition. |
| SRS0002.2 | The system shall activate the Warning (YELLOW) indicator when the current temperature is above the LOW\_THRESHOLD condition but below the HIGH\_THRESHOLD condition. |
| SRS0002.3 | The system shall activate the Normal (GREEN) indicator when the current temperature is at or below the LOW\_THRESHOLD condition. |

### Sensor.java

#### Description:

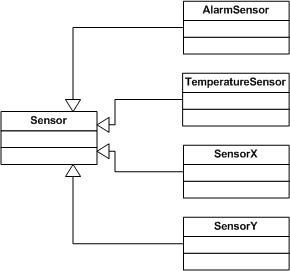
The Sensor class represents a generic sensor. This is the superclass for the TemperatureSensor and AlarmSensor objects.

#### UML Architecture



#### Design Details/Architecture Notes:

The Sensor class is meant to be subclassed by any and all connected sensors/electronics that the host needs to request or process data from. Although the Thermostat system currently only supports one I2C temperature sensor and alarm indicators, the Sensor class provides the scalability for future features to be added with little effort, supplying the basic information that can be shared among all integrated components.



The Sensor class provides two primary methods of importance; processSensorData() and requestSensorData(). The processSensorData() method is meant for processing data coming from the client, while the requestSensorData() is used to make requests to the client for data. Any and all method calls contained within the previous two methods should be private, so that any outside class requesting sensor data or process sensor data must use either of these methods, as they provide the common interface.

#### Requirements Traceability

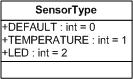
|  |  |
| --- | --- |
| **Requirement #:** | **Requirement Text:** |
| SRS0001 | The system shall record temperature readings from the sensor at a 1 Hz rate. |
| SRS0002 | The system shall activate one of three LED indicators according to the current temperature and two threshold conditions |
| SRS0002.1 | The system shall activate the Danger (RED) indicator when the current temperature is at or above the HIGH\_THRESHOLD condition. |
| SRS0002.2 | The system shall activate the Warning (YELLOW) indicator when the current temperature is above the LOW\_THRESHOLD condition but below the HIGH\_THRESHOLD condition. |
| SRS0002.3 | The system shall activate the Normal (GREEN) indicator when the current temperature is at or below the LOW\_THRESHOLD condition. |
| SRS0003 | The system shall communicate with the PC/Laptop over a Serial link, with a messaging protocol of your own design. |

### SensorType.java

#### Description:

The SensorType class defines the different sensors for the Thermostat application.

#### UML Architecture



#### Details:

The SensorType class was created for processing by the XMLFileLoader class. For the Thermostat system, it is not necessary; however the XML files have been designed in a way to easily add on additional sensors and electronics for future projects and study.

#### Requirements Traceability

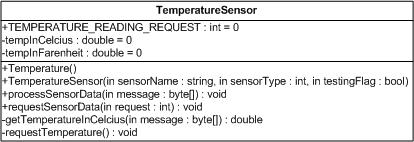
|  |  |
| --- | --- |
| **Requirement #:** | **Requirement Text:** |
| SRS0001 | The system shall record temperature readings from the sensor at a 1 Hz rate. |
| SRS0002 | The system shall activate one of three LED indicators according to the current temperature and two threshold conditions |
| SRS0002.1 | The system shall activate the Danger (RED) indicator when the current temperature is at or above the HIGH\_THRESHOLD condition. |
| SRS0002.2 | The system shall activate the Warning (YELLOW) indicator when the current temperature is above the LOW\_THRESHOLD condition but below the HIGH\_THRESHOLD condition. |
| SRS0002.3 | The system shall activate the Normal (GREEN) indicator when the current temperature is at or below the LOW\_THRESHOLD condition. |
| SRS0003 | The system shall communicate with the PC/Laptop over a Serial link, with a messaging protocol of your own design. |

### TemperatureSensor.java

#### Description:

The TemperatureSensor class represents the TMP102 I2C temperature sensor which is the primary temperature sensor integrated into the Thermostat system.

#### UML Architecture



#### Details:

No additional design details.

#### Requirements Traceability

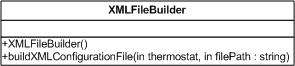
|  |  |
| --- | --- |
| **Requirement #:** | **Requirement Text:** |
| SRS0001 | The system shall record temperature readings from the sensor at a 1 Hz rate. |

### XMLFileBuilder.java

#### Description:

The XMLFileBuilder class is used to build the XML configuration settings file for the system. Using multiple classes from the javax.xml packages, it uses the primary Thermostat object, grabbing multiple pieces of data to build the configuration file.

#### UML Architecture



#### Details:

In addition to satisfying the requirements listed in section 3.2.21.4, it adds additional functionality to support future additions and features, including adding additional sensors. It does this using the populated linked list of Sensor objects from the Thermostat instance, automatically creating the XML file based upon all sensors and electronics that the Thermostat application knows about.

This additional functionality was deemed necessary to create the ability to continually add features, making the application and system more modular, hence supporting MOSA objectives.

#### XML File:

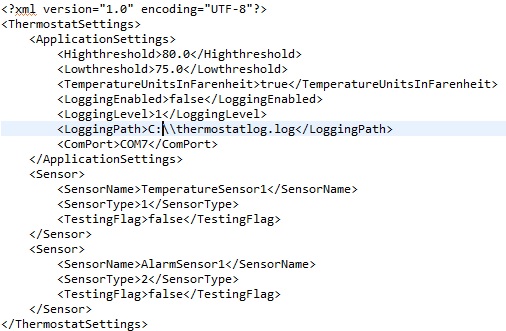


Figure 3: XML Configuration Settings

The figure above shows the overall structure of the XML Settings file. It contains multiple tags of which each are described below:

##### ThermostatSettings

The <ThermostatSettings> tag is the root tag for the XML configuration settings file.

##### ApplicationSettings

The <ApplicationSettings> denotes that everything contained within are application-level settings.

##### Highthreshold

The value contained within the <Highthreshold></Highthreshold> tags represents the user configured HIGH\_THRESHOLD value. Per requirements of the system, this value is used to activate/deactivate the alarm indicator. Please refer to SRS0002 and sub-requirements for detailed information.Lowthreshold

##### TemperatureUnitsInFarenheit

The value contained within the <TemperatureUnitsInFarenheit></TemperatureUnitsInFarenheit> tags is a boolean value of true/false. When set to true, the default display units will be set to Degrees Fahrenheit. Celcius when set to false.

##### LoggingEnabled

The value contained within the <LoggingEnabled></LoggingEnabled> tags is also a boolean value of true/false. When set to true, logging will be enabled and types of log messages saved and created determined by the value of the <Logging> tag.

##### LoggingLevel

The following values are supported for the <LoggingLevel></LoggingLevel> tag:

|  |  |
| --- | --- |
| **Level:** | **Description:** |
| 0 | Logging is disabled. |
| 1 | Log all Severe messages. |
| 2 | Log all Warning messages. |
| 3 | Log all Info. |
| 4 | Log all Config. |

##### LoggingPath

The value contained within the <LoggingPath></LoggingPath> tags defines where the logs will be saved.

##### ComPort

The value contained within the <ComPort></ComPort> defines the communications port to communicate on.

##### Sensor

The values contained within the <Sensor></Sensor> tags define each and every sensor that is in the system. For the Thermostat system, two sensors are defined.

##### SensorName

The value contained between the <SensorName></SensorName> tags define the name of the particular sensor. In the current version of the Thermostat system, this value is not used for any logic processing. It is presumed that this will be used in future releases and updates.

##### SensorType

The value contained between the <SensorType></SensorType> tags define the type of sensor. This value is currently being used for processing by the application.

##### TestingFlag

The value contained between the <TestingFlag></TestingFlag> defines whether or not the Sensor is in “test”mode. This is a feature currently not implemented in the software and will be added in future releases.

#### Requirements Traceability

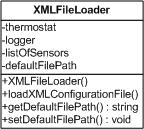
|  |  |
| --- | --- |
| **Requirement #:** | **Requirement Text:** |
| SRS0007 | The software shall load/save the user inputs from/to an XML file on startup/shutdown or on demand. |

### XMLFileLoader.java

#### Description:

The XMLFileLoader class is responsible for loading the XML application settings.

#### UML Architecture



#### Design Details/Architecture Notes:

(TODO)

#### Requirements Traceability

|  |  |
| --- | --- |
| **Requirement #:** | **Requirement Text:** |
| SRS0007 | The software shall load/save the user inputs from/to an XML file on startup/shutdown or on demand. |

# CLIENT - MCU SOFTWARE

This section describes the overall design of the MCU C++-based Thermostat application.

## Packages

This section is N/A for the MCU Thermostat application.

## Classes

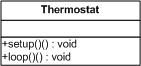
This section describes the various classes for the MCU Thermostat application

### Thermostat

#### Description:

The Thermostat class is the primary sketch for the MCU application. It contains the required Arduino functions setup(), which initiates the serial and i2c libraries, and loop(). The loop function continually runs, waiting for 7 bytes to become available. Once 7 bytes are available, it makes a call to the MessageReceiver class.

#### UML Architecture:



#### Design Details/Architecture Notes:

No additional notes at this time.

#### Requirements Traceability

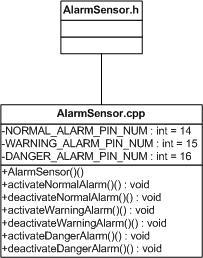
|  |  |
| --- | --- |
| **Requirement #:** | **Requirement Text:** |
| SRS0001 | The system shall record temperature readings from the sensor at a 1 Hz rate. |
| SRS0002 | The system shall activate one of three LED indicators according to the current temperature and two threshold conditions |
| SRS0002.1 | The system shall activate the Danger (RED) indicator when the current temperature is at or above the HIGH\_THRESHOLD condition. |
| SRS0002.2 | The system shall activate the Warning (YELLOW) indicator when the current temperature is above the LOW\_THRESHOLD condition but below the HIGH\_THRESHOLD condition. |
| SRS0002.3 | The system shall activate the Normal (GREEN) indicator when the current temperature is at or below the LOW\_THRESHOLD condition. |
| SRS0003 | The system shall communicate with the PC/Laptop over a Serial link, with a messaging protocol of your own design. |

### AlarmSensor (.h and .cpp)

#### Description:

The AlarmSensor class provides the necessary functions to activate/deactivate the appropriate alarm (LED) on the connected Input/Output (I/O) pin. The I/O pins are defined in the class.

#### UML Architecture:



#### Design Details/Architecture Notes:

No additional notes at this time.

#### Requirements Traceability

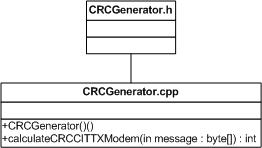
|  |  |
| --- | --- |
| **Requirement #:** | **Requirement Text:** |
| SRS0002 | The system shall activate one of three LED indicators according to the current temperature and two threshold conditions |
| SRS0002.1 | The system shall activate the Danger (RED) indicator when the current temperature is at or above the HIGH\_THRESHOLD condition. |
| SRS0002.2 | The system shall activate the Warning (YELLOW) indicator when the current temperature is above the LOW\_THRESHOLD condition but below the HIGH\_THRESHOLD condition. |
| SRS0002.3 | The system shall activate the Normal (GREEN) indicator when the current temperature is at or below the LOW\_THRESHOLD condition. |
| SRS0003 | The system shall communicate with the PC/Laptop over a Serial link, with a messaging protocol of your own design. |

### CRCGenerator (.h and .cpp)

#### Description:

The CRCGenerator class is used to create/validate the CRC for a message being sent to the host as well as messages received from the host. It provides a similar implementation to that on the host side.

#### UML Architecture:



#### Design Details/Architecture Notes:

No additional notes at this time.

#### Requirements Traceability

|  |  |
| --- | --- |
| **Requirement #:** | **Requirement Text:** |
| SRS0003 | The system shall communicate with the PC/Laptop over a Serial link, with a messaging protocol of your own design. |

### MessageProcessor (.h and .cpp)

#### Description:

The MessageProcessor class provides the necessary functionality needed to process the message once a message has been received by the MessageReceiver and it has been validated by the CRCGenerator object.

Similar to the MessageProcessor on the host, it looks at the message type field of the message and branches as necessary based on the type found. In this version of the Thermostat system, the messages defined in the following sub-sections are processed:

##### DEFAULT\_MSG

Currently no action is taken on this message.

##### TEMP\_SENSOR\_READING\_REQUEST\_MSG

When this message is received, a call is made to the TemperatureSensor object for a temperature reading.

##### NORMAL\_ALARM\_SET\_MSG

When this message is received, the Normal alarm is enabled, all others become disabled.

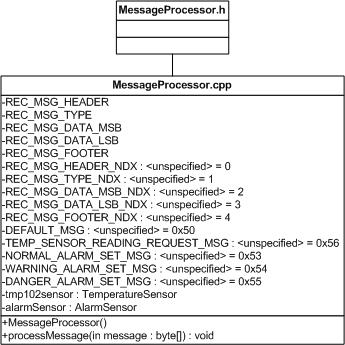
##### WARNING\_ALARM\_SET\_MSG

When this message is received, the Warning alarm is enabled, all others are disabled.

##### DANGER\_ALARM\_SET\_MSG

When this message is received, the Danger alarm is enabled, all others are disabled.

#### UML Architecture:



#### Design Details/Architecture Notes:

No additional notes at this time.

#### Requirements Traceability

|  |  |
| --- | --- |
| **Requirement #:** | **Requirement Text:** |
| SRS0001 | The system shall record temperature readings from the sensor at a 1 Hz rate. |
| SRS0002 | The system shall activate one of three LED indicators according to the current temperature and two threshold conditions |
| SRS0002.1 | The system shall activate the Danger (RED) indicator when the current temperature is at or above the HIGH\_THRESHOLD condition. |
| SRS0002.2 | The system shall activate the Warning (YELLOW) indicator when the current temperature is above the LOW\_THRESHOLD condition but below the HIGH\_THRESHOLD condition. |
| SRS0002.3 | The system shall activate the Normal (GREEN) indicator when the current temperature is at or below the LOW\_THRESHOLD condition. |
| SRS0003 | The system shall communicate with the PC/Laptop over a Serial link, with a messaging protocol of your own design. |

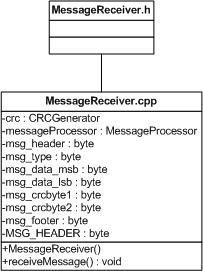
### MessageReceiver (.h and .cpp)

#### Description:

The MessageReceiver class has one function, receiveMessage(). Once the bytes are available on the serial line, the receiveMessage function is called by the Thermostat where it reads off the bytes, saving them to variables. Once all bytes have been read, the message is built, along with a payload byte array for passing off to the CRCGenerator.

The CRC is generated and checked against the CRC bytes in the message itself. It the CRC is valid, the full message is then passed off to the MessageProcessor, where action is to be taken. If the message is determined to be corrupted, then no action is taken in this version of software other than printing to the serial debugging, which is not used.

#### UML Architecture:



#### Design Details/Architecture Notes:

Further analysis should be taken and features added in future increments of the software such as re-requesting messages, etc.

#### Requirements Traceability

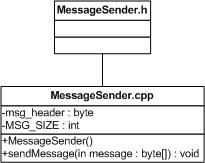
|  |  |
| --- | --- |
| **Requirement #:** | **Requirement Text:** |
| SRS0001 | The system shall record temperature readings from the sensor at a 1 Hz rate. |
| SRS0002 | The system shall activate one of three LED indicators according to the current temperature and two threshold conditions |
| SRS0002.1 | The system shall activate the Danger (RED) indicator when the current temperature is at or above the HIGH\_THRESHOLD condition. |
| SRS0002.2 | The system shall activate the Warning (YELLOW) indicator when the current temperature is above the LOW\_THRESHOLD condition but below the HIGH\_THRESHOLD condition. |
| SRS0002.3 | The system shall activate the Normal (GREEN) indicator when the current temperature is at or below the LOW\_THRESHOLD condition. |
| SRS0003 | The system shall communicate with the PC/Laptop over a Serial link, with a messaging protocol of your own design. |

### MessageSender (.h and .cpp)

#### Description:

The MessageSender class contains one function, sendMessage(). The sendMessage() function is responsible for transmission of messages from the client to the host. In this version of the Thermostat software, it is used by, and only by the TemperatureSensor class.

#### UML Architecture:



#### Design Details/Architecture Notes:

Currently the sendMessage() function is only used by one other class. If features such as additional sensors are added/required in the future, the sendMessage() function should be called though one object only, instead of being called by multiple classes.

#### Requirements Traceability

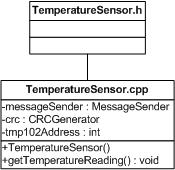
|  |  |
| --- | --- |
| **Requirement #:** | **Requirement Text:** |
| SRS0001 | The system shall record temperature readings from the sensor at a 1 Hz rate. |
| SRS0003 | The system shall communicate with the PC/Laptop over a Serial link, with a messaging protocol of your own design. |

### TemperatureSensor (.h and .cpp)

#### Description:

The TemperatureSensor class provides the interface for obtaining the temperature reading from the TMP102 sensor. It provides the single function, getTemperatureReading(), which handles the request to the connected i2c sensor, obtaining the reading, and sending to the host.

#### UML Architecture:



#### Design Details/Architecture Notes:

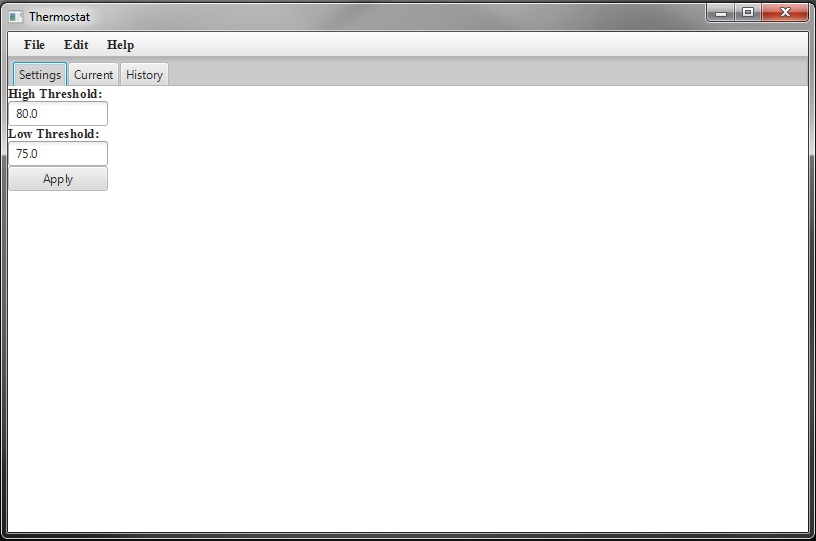
No additional notes at this time.

#### Requirements Traceability

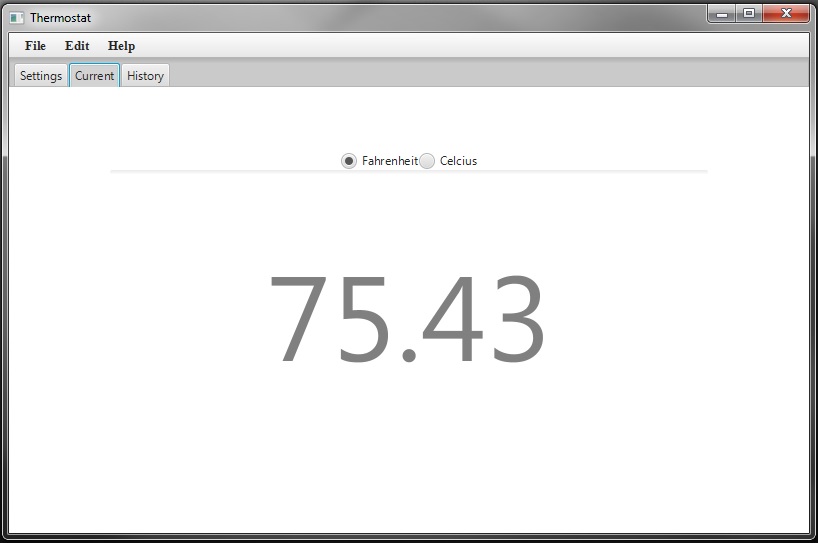
|  |  |
| --- | --- |
| **Requirement #:** | **Requirement Text:** |
| SRS0001 | The system shall record temperature readings from the sensor at a 1 Hz rate. |
| SRS0003 | The system shall communicate with the PC/Laptop over a Serial link, with a messaging protocol of your own design. |

# USER INTERACE

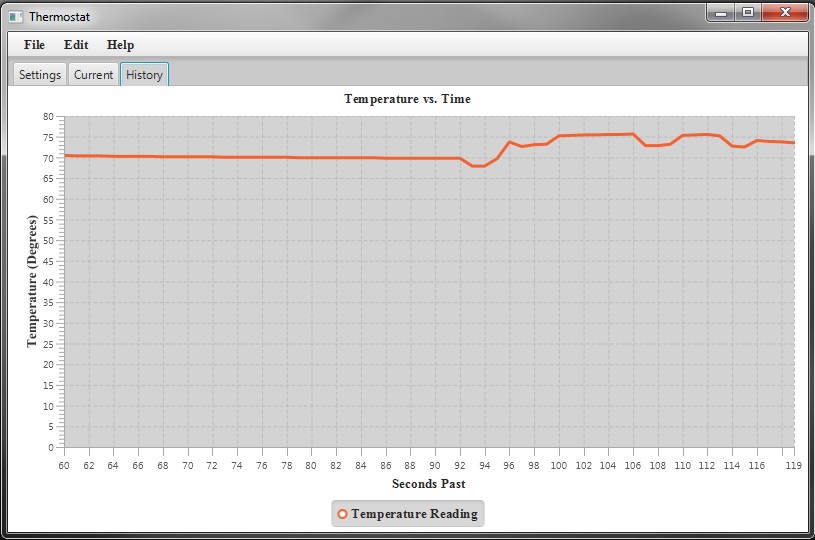
## Settings Tab View



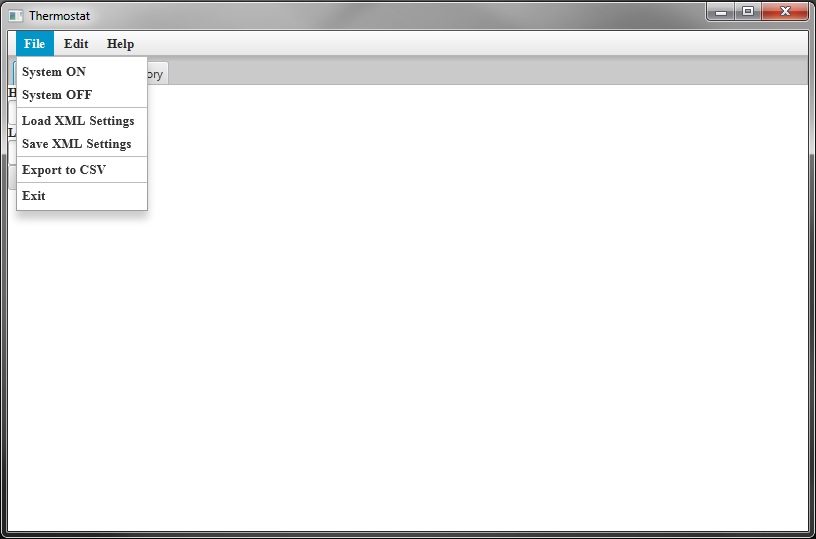
## Current Tab View



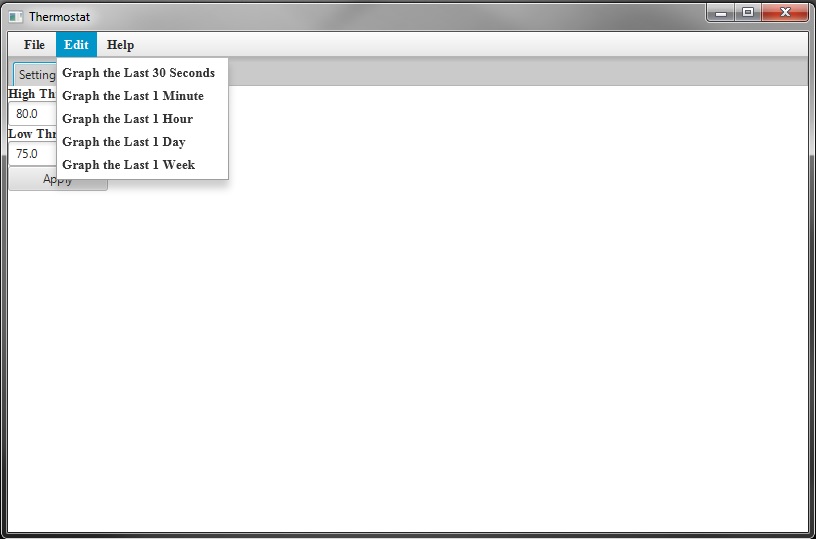
## History Tab View



## File Menu Extended



## Edit Menu Extended



# WANTED FeATURES

The items listed within this section highlight future features that should be implemented in upcoming revisions of the software.

|  |  |
| --- | --- |
| Feature Number: | Feature Description: |
| FEAT0001 | Implement autostart. Currently the host application requires the user to select “File”->”System ON” to start the system. |

# DESIGN SHORTFALLS/KNOWN ISSUES

This section highlights currently known shortcomings of the current version of software. It lists known Software Issue Report (SIR) numbers.

|  |  |
| --- | --- |
| Shortfall/Issue Number: | Description: |
| SIR0001 | Sensors are currently indexed by hard-coded values, making the applicability of the configuration file Sensor tags not fully applicable. Further architectural design is necessary prior to implementation. Since this is not a requirement of the system at this time, development of it has been put on hold. |
| SIR0002 | User can change the temperature measurement units at any point, causing potential odditites in both the graph and/or exported values due to mixture of values. Recommended solution would be to add an extra value to the TemperatureRecord to denote the values are at the time of recording so that it will be easier to filter when exported into another application such as Excel. |
| SIR0003 | Loading the XML configuration file after started does not change the selected temperature display units. |
| SIR0004 | Currently that is no error handling such as XSLT or other in the code to correctly handle incorrectly formatted XML configuration settings files. |
| SIR0005 | Messages from the host to the client should go out through a single entity such as a TimedQueue so that multiple objects do not try to access the serial port at the same time. Messages should get added to a queue, and then processed there by an additional field such as messagePeriodicy that states the frequency of the message to be sent. Further design/analysis is necessary to implement. |

# SYSTEM REQUIREMENTS

This section takes the requirements that were defined in the Embedded Systems Task Specification and gives them numbers for cross-reference in this SPS. Where applicable, they have been broken into sub-requirements as well.

|  |  |
| --- | --- |
| **Requirement #:** | **Requirement Text:** |
| SRS0001 | The system shall record temperature readings from the sensor at a 1 Hz rate. |
| SRS0002 | The system shall activate one of three LED indicators according to the current temperature and two threshold conditions |
| SRS0002.1 | The system shall activate the Danger (RED) indicator when the current temperature is at or above the HIGH\_THRESHOLD condition. |
| SRS0002.2 | The system shall activate the Warning (YELLOW) indicator when the current temperature is above the LOW\_THRESHOLD condition but below the HIGH\_THRESHOLD condition. |
| SRS0002.3 | The system shall activate the Normal (GREEN) indicator when the current temperature is at or below the LOW\_THRESHOLD condition. |
| SRS0003 | The system shall communicate with the PC/Laptop over a Serial link, with a messaging protocol of your own design. |
| SRS0004 | The GUI shall accept two threshold settings (HIGH\_THRESHOLD, LOW\_THRESHOLD) from the user and validate that HIGH\_THRESHOLD > LOW\_THRESHOLD. |
| SRS0005 | The GUI shall display the current temperature in user-selectable units of Fahrenheit or Celsius. |
| SRS0006 | The GUI shall display a graph of temperature vs. time: last 10 minutes, last 60 minutes, last 24 hours, last week |
| SRS0007 | The software shall load/save the user inputs from/to an XML file on startup/shutdown or on demand. |
| SRS0008 | The software shall provide an export function of the temperature history data to a CSV file, such that MS Excel can ingest the file and plot it. |
| SRS0009 | The GUI shall be coded in Java, using FX for GUI elements. |
| SRS0010 | The GUI shall implement a standard menu bar (minimum: File, Edit, Help) |
| SRS0011 | The GUI shall implement a tabbed interface (Settings, Current, History) |

# CALCULATIONS

This section highlights calculations used for circuitry in the Thermostat system. For the current-limiting resistors attached to the LEDs, the following formulas where used:

V = IR

R = V/I -> R = (Vs – Vf)/I

Per the LED datasheet, the red LED has a 1.8-2.2V forward drop. For the resistor calculation, I took the in between value of 2.0V. Suggested using current is 16-18 mA. With this information, the calculations become the following:

R = (3.3V – 2.0V)/0.018A = 81.25 ohms or (3.3V – 2.0V)/0.016A = 72.20 ohms

Using resistors in my inventory (instead of buying new ones), I used values of 270 ohms, well above the values listed above, to give much smaller currents through the LEDs. Although not as bright as the calculated values above, they suffice for this project.