Coding Standard Document
Pressure Sensing System Project

Owners: Matheus Barbosa, Rachel Mosier, Charles Abramson.

Institution: Oklahoma State University.

Purpose: The purpose of this document is to provide a standard for the code which is going to be written for this project. This document may be edited at any time by any of the owners listed above. In order to write reusable, and adaptable code, this document outlines the minimum features which may be present in any code written by any member of this project. Any code that fails to meet *any* of the requirements of the latest version of this document will be deemed unsatisfactory and will not be used.

Version: 1.0 updated on October 15th 2015.

Table of Contents

1 Files and Folder Names

In order to keep a good organizational structure we will designate naming standards for files. Also, in order for us to have a standard hierarchy, we will define where each file should be placed. Most importantly, all files must be placed in the GitHub repository which belongs to this project. The repository will maintain the same hierarchy and naming structure defined here.

1.1 Names

All files and folders will be named in order to quickly located them and associate them with their function based on the name. As of now, we plan on all files being Spin Objects (.spin) so they will all have the same extensions. The names will be pascal cased and underscores will be used as spaces in order to not conflict file naming in between operating systems. The names should reflect the object's function, for example if we have a file whose function is to convert an integer variable to a string the name could be: “Integer_To_String.spin”. If possible, the names should be kept to four words in length, but we should not sacrifice clarity to remain within four words.

1.2 Versions

It is not required that each file have different versions of itself, but it is recommended to review any older code and make necessary updates. For simplicity purposes, we will not require different versions of the same file to be saved, unless more than half of the code is changed. If any change is made whose effect is on less than 50% of the code, that version can be overwritten onto the older version. If the file is simply overwritten, a note regarding the changes should be added to the header comment. A less than 50% change will be denoted by incrementing the first decimal place from the previous version. For example: version 1.1 to version 1.2. A major revision is one which changes more than 50% of the code and the new version will be the next whole number higher than the previous version. For example: version 1.3 to 2.0.

1.3 Hierarchy

The standard hierarchy for this file system will be very simple. There will be a main folder named “Code” and under that folder there will be multiple folders diving the code files into their functions. As of now, the functions will be “Seismic_Data”, “Large_Device”, “Handheld_Device”. More may be added later if the system is used for different functions/projects. Each function folder will contain the source coed files within it, in no particular order. Any relevant objects not written by the members of the group, but which we are using, may be added to the same folders as the objects that call it. These names should remain as the original names and should not be changed to our standard. This will provide a simple way to identify objects that belong to us.

2 Variables

In order to make our code easily changeable, we will also set up a variable naming standard. This will not differ much from the accepted variable naming convention from most popular languages like C++, C, Java, etc.

2.1 Constants

Constants are not variables (clearly) but they have a special naming convention in programming which will be discussed here. Constant will be named in all capital letters, and underscores will denote spaces between words. For example: “THIS_IS_A_CONSTANT”.

2.2 Integers

Integers will be the preferred method to represent any numeric value. Integer is not an actual variable type in the spin language, but we still deal with them. Words, longs, shorts and bytes can be treated as integers. The first three are preferred because byte is often used to represent integers in base systems other than base 10. Integer names should reflect the variable function in such a way that anyone editing the code will know what it stands for. The name will be camel cased with no separation between words, for example: “thisIsAnInteger”.

2.3 Characters

In spin, characters can be treated as substrings or as integers corresponding to their ASCII value. The latter representation should not be used unless it is absolutely necessary, since it can get confusing. Character and string variables will be named the same way as integers, unless they are a constant in which case they will be named like a constant. It is important to note that spin does not always treat characters and strings in the same way. It depends on how the particular method was written.

2.4 Floating Points

Floating point variables are available in spin, but they should be avoided as much as possible. Floats require special objects to be manipulated in spin, so coding with floats can get quite involved and unreadable. The preferred method when dealing with data that might become irrational is to scale it and make it an integer. For example, the number 0.269 could be scaled to 269 and used as 269 in the program. Normally, multiply the number by power of 1,000 is sufficient.

3 Comments

This code will be professional and well documented. In order to do so, we must include comments in all code. Each code file will have a large comment block at the top called the header comment. There will also be blocks of comments integrated throughout the code as necessary.

3.1 Header Comments

The header comment will be the first thing in any coding file. This comment will include the program name, the date in which it was first written, the date in which it was last modified (if applicable), a short description of the program function, and a log of the changes made to each version. The Heder comment should look like this:

```
{{
Name_Of_Program
Written on: dd/mm/yyyy
Last modified on: dd/mm/yyyy
Description: This program was written to....

**Version x.y**
-Changed x, y and z.

**Version x.z**
-Added a, b and c.
}}
```

Note that the test above is a template, and the actual heading comments should look like it, with the fields filled in appropriately.

3.2 Integrated Comments

There should be other comments throughout the code explaining what exactly is being done at that section of code. “Line-by-line” comments should be avoided because these can get messy. Rather, there should be a block of comment before a block of code, explaining what the code does. This will make our code more readable and understandable.