StarContests.com

If it's not here it's not happening.

Coding Standards v1.0

Team 8

Instructor - Prof. Asim Banerjee

GROUP MEMBERS

Serial No.	Name	ID
1.	HARDIK BELADIYA	201201064
2.	ARCHIT GAJJAR	201201066
3.	SOHAM DARJI	201201070
4.	KRUPAL BAROT	201201074
5.	DHAVAL CHAUDHARY	201201075
6.	PRACHI KOTHARI	201201077
7.	YASH KUMAR JAIN	201201080
8.	RACHIT MISHRA	201201092
9.	SHIVANI THAKKER	201201108

Revision History

Date	Document	Version	Created By	Reviewed By
25th March 2015	Coding Standards	1.0	Hardik Beladiya	Prachi

1. Document Control

- **Document description**

This document intends to provide a proper outline about the Coding Standard . What are the Coding Standard used throughout the Development phase so that any one can easily understand the code .

- **Document convention**

- The following conventions would be used throughout the document,

1. Headings - Intense, 18, Bold Blue
2. Subheadings - Intense, 14
3. Body - Calibri, 12

Table of Contents

1. Introduction

The goal of these guidelines is to create uniform coding habits among software personnel in the engineering department so that reading, checking, and maintaining code written by different persons becomes easier. The intent of these standards is to define a natural style and consistency, yet leave to the authors of the engineering department source code, the freedom to practice their craft without unnecessary burden. Not only does this solution make the code easier to understand, it also ensures that any developer who looks at the code will know what to expect throughout the entire application.

When a project adheres to common standards many good things happen:

1. Programmers can go into any code and figure out what's going on, so maintainability, readability, and reusability are increased. Code walk through become less painful.
2. New people can get up to speed quickly.
3. People new to a language are spared the need to develop a personal style and defend it to death.
4. People new to a language are spared making the same mistakes over and over again, so reliability is increased.
5. People make fewer mistakes in consistent environments.
6. Idiosyncratic styles and college-learned behaviors are replaced with an emphasis on business concerns - high productivity, maintainability, shared authorship, etc.

2. Standards

2.1 Indentation

A consistent use of indentation makes code more readable and errors easier to detect. Indentation should be used to:

1. Emphasize the body of a control statement such as a loop or a select statement
2. Emphasize the body of a conditional statement
3. Emphasize a new scope block

A tab (four columns or four spaces) is the basic unit of indentation. Once the programmer chooses the number of spaces to indent by, then it is important that this indentation amount be consistently applied throughout the program.

For Example:

```
Function myFunction()

{

 /<tab>/

          code;

          code;

          if (AnotherLevel) {

         /<tab>/

                 moreCode;

                 moreCode;

                 if (AnotherLevel) {

                  /<tab>/

                          moreCode;

                          moreCode;

                  }

        }

}
```

2.2 Functions

Keep functions, and methods reasonably sized. This depends upon the language been used. When coding classes and functions, the following formatting rules are followed:

1. No space between a method name and parenthesis “(” starting its parameter list.
2. Open brace “{” appear at the beginning of the next line of the declaration statement.
3. Closing brace “}” starts a line by itself indented to match its corresponding opening statement, except when it is a null statement the “}” should appear immediately after the “{”.

The names of the classes, subroutines, functions and method shall have verbs in them, i.e., names shall specify an action.

2.3 Statements

Each line contains at most one statement.

Example:

```
Int dirlen;                    /* Correct */

string asmbase;                /* Correct */
int dirlen; string asmbase;/* AVOIDED */
```

2.3.1 "if", "if else", "if else-if else" Statements

The if-else class of statements is defined in the following form:

```
        Single statement if-else :
                if(condition)
                {

                        single statement;
                }
                else
                {
                        single statement;
        }
```

Multiple line if-else :

```
if(condition)
{

        statements;

}

else

{
        statement
}
```

Multiple line if-else if-else : if(condition)

```
{

        statements;

}

else if(condition)
{

        statements;

}

else

{
        statements;

}
```

2.3.2 “for” Statements

A for statement is given the following form:

```
for (initialization; condition; update)
{
    statements;
```

```
        }
```

2.4 Queries Statements

Queries written in our document is in the form given below:

```
1)      db.products.find( { qty : { $gt : 25 } } , { item : 1 , qty : 1 } )

2)      db.collections.insert( { item : “card”, qty : 5 } )
```

It is case sensitive.

2.5 Use of Braces

Braces are used to delimit the bodies of conditional statements, control constructs and the block of scope. Programmer should use the following bracing style:

```
for (int j = 0 ; j <max_iterations ; ++j)

{

   /* Some work is done here. */

}
```

The above bracing style is more readable and leads to neater-looking code. Make sure that style is consistent throughout the code. When editing code written by different author, adopt the style of bracing used.

Braces shall be used even when there is only one statement in control block. For example:

Bad:

```
if (j == 0)
  printf (“j=0.\n”);
```

Better:

```
 if (j == 0)
{
        printf (“j is zero.\n”);
    }
```

2.6 Comments

No specific format shall be followed for comments. In general trailing comments or End-of-Line comments shall be used. The comments have been used wherever deemed appropriate by the developer in the following way:

```
    // populate language menu in a consistent way

    code;
```

3. Guidelines

3.1 Spacing

The proper use of spaces within a line of code can enhance readability. Good rules of thumb are as follows:

A keyword followed by a parenthesis should be separated by a space. A blank space should appear after each comma in an argument list.

All binary operators except “.” Should be separated from their operands by spaces. Blank spaces should never separate unary operators such as unary minus, increment (“++”), and decrement (“--”) from their operands.

Example:

Bad:

```
 cost=price+(price*sales_tax);

printf( “The total cost is %5.2f\n”,cost);
```

Better:

```
 cost = price + ( price * sales_tax );

 printf (“The total cost is %5.2f\n”, cost) ;
```

3.2 Naming Convention

3.3.1 Method Names

Start them with a lowercase letter and use intercaps for subsequent words: measure(), updateDisplayList().

3.3.2 Literal Names

Start them with a lowercase letter and use underscore in between subsequent words: count, total_count.

3.3.4 Constant Names

Use all uppercase letters with underscores between words:
OFF, DEFAULT_WIDTH.

3.3 Abbreviations

Avoid them as a general rule. For example, calculateOptimalValue() is a better method name than calcOptVal().

Being clear is more important than minimizing keystrokes. And if you don't abbreviate, developers won't have to remember whether you shortened a word like “qualified” to “qual” or “qlfd”.

However, we have standardized on a few abbreviations:

- acc for accessibility, as in ButtonAccImpl
- auto for automatic, as in autoLayout
- eval for evaluate, as in EvalBindingResponder
- impl for implementation, as in ButtonAccImpl
- info for information, as in GridRowInfo
- num for number of, as in numChildren
- min for minimum, as in minWidth
- max for maximum, as in maxHeight
- nav for navigation, as in NavBar
- regexp for regular expression, as in RegExpValidator
- util for utility, as in StringUtil

4. References

GNU CODING STANDARDS
http://www.gnu.org/prep/standards/html_node/References.html

RAILS CODING GUIDE
http://guides.rubyonrails.org/contributing_to_ruby_on_rails.html