ANOTODE

The Web Annotator .

Coding Guidlines Revision 2.0

CS Group 1

Authors:
Avi (201451070)
Chahat (201451061)

Reviewed by:
Manohar (201451024)
Monika (201451062)

Revision Table

Revision	Author	Reviewer	Revision Date	Revision Tracking Notes
1	Avi	-	5-10-16	Initial version
2	Chahat		5-10-16	

Contents

1 Introduction

Anotode is a high-quality product and we make no compromises as per as the heart of the product is concerned. Building quality products require good code and Anotode is no exception.

Well written software offers many advantages. It will contain fewer bugs and will run more efficiently than poorly written programs. Since software has a life cycle and much of which revolves around maintenance, it will be easier for the original developers and future keepers of the code to maintain and modify the software as needed.

This document contains the standards and guidelines that the developers are supposed to follow while writing code for the Anotode project. Standards are rules which programmers are expected to follow and they will be enforced through automated systems and manual reviews where automatic check is not applicable. Guidelines can be viewed as suggestions which can help programmers write better software and are optional, but highly recommended.

At some cases, a standard or guideline may vary for different languages and it will be clearly specified in the document. This document will be subject to revisions and the standards will be evolved when needed.

2 Coding Standards

2.1 Indentation

Indentation is a very important factor for producing readable and maintainable code. It should be used whenever there is a need to -

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

JavaScript, CSS and HTML code should use 2 spaces of indentation whereas Java code should use 1 tab indentation. The above specification is consistent with the JSLint specification and the JavaDoc specifications.

.

2.2 Inline comments

Inline comment should be used to clearly define what is happening in the code where needed. The more the number of inline comments, the better it is.

2.3 Modular Programming

Code should be modular and the modules should be properly defined such as they are as much independent as possible. Deprecated instructions like GOTO should seldom be used.

2.4 Functions and Methods

Functions and methods should be small in size and should serve a particular purpose. The name should be descriptive and should use either camelCase or under$_s$*coresconventiontoseparatewords.Thenamingshouldnotbearbitrary.Thenamesoftheclasses*, *subroutine*

2.5 Code files

Code files should be name in such a way that the purpose of the code in that file can be understood from the name itself.

2.6 Variable Names

Variable shall have mnemonic or meaningful names that convey to a casual observer, the intent of its use. Variables shall be initialized prior to its first use. They should be either in camelCase or under$_s$*cores*$_c$*ase.JavaScriptcodeshouldhavevariablesincamelCase.*

2.7 Using braces

Braces should strictly be used even if it is a one-line statement.

```
Bad
If (a > b)
    console.log("test")


Good
If (a>b){
    console.log("test")
}
```

3 Coding Guidelines

3.1 Line Length

It is a good idea to keep line length less than 150 characters. This will help maintain code readability on small screen sizes. JavaScript code should be less than 120 chars as then it will be consistent with ESLint spec.

3.2 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.

```
Bad
Val = 2+4-sum
Call = func(a,b,c)


Good
    Val = 2 + 4 - sum
```

3.3 Variable Declarations

Multiple variable declarations in a single line must be separated by space after comma

```
Bad
    Var a,b,c


Good
    Var a, b, c
```

3.4 Statements

Statements should be limited to one per line to avoid complexity and improve code readability. It will also make the code look more consistent w.r.t coding style.

3.5 Meaningful error messages

Error messages should be clearly defined with error codes and help messages. This will help to understand the error more clearly as well as help devs locate the error quickly. When possible, they should indicate what the problem is, where the problem occurred, and when the problem occurred.

Error messages should also be logged to standard output to make them easily debuggable by the admin team. Optionally, they can also be recorded in a non-temporary location so that they can be reviewed upon when needed.

3.6 File length

File length (line count) should be kept less than 300 lines. If the file is exceeding that, it is better to break it into 2 or more modules.