Coding Standard Document

Document ID: DRSYS-CSD-01

Author(s): Chandra Sekaran.R, Rajesh.K

Version: 1.0

Date: Aug 18, 2014

Table of Contents

[1. Abstract 3](#_Toc396125579)

[2. Introduction 3](#_Toc396125580)

[3. Indentation 3](#_Toc396125581)

[4. Line Length 4](#_Toc396125582)

[Wrapping Lines 4](#_Toc396125583)

[Passing parameters 4](#_Toc396125584)

[Arithmetic Expressions 4](#_Toc396125585)

[Method Declarations 5](#_Toc396125586)

[Conditional Statements 5](#_Toc396125587)

[Ternary Expression 6](#_Toc396125588)

[5. Comments 6](#_Toc396125589)

[Implementation comment 6](#_Toc396125590)

[Documentation Comments 7](#_Toc396125591)

[6. Definitions 7](#_Toc396125592)

[Initialization 7](#_Toc396125593)

[White space 7](#_Toc396125594)

[Blank lines 7](#_Toc396125595)

[Blank spaces 8](#_Toc396125596)

[7. Ruby structures 8](#_Toc396125597)

[Classes and Modules 8](#_Toc396125598)

[Methods 9](#_Toc396125599)

[Variables 9](#_Toc396125600)

[Constants 9](#_Toc396125601)

[8. Identifier nomenclature ………………………………..](#_Toc396125582)......…………10

[Ruby structures 10](#_Toc396125601)

[Cucumber – Page Object 10](#_Toc396125601)

9. Modification history...…………………………………………………11

# Abstract

The purpose of the document is to describe the standards to be followed for Test Automation Project. This document describes about the coding standards for Ruby based Test Automation Project.

# Introduction

While developing scripts for an automation project, it is good to follow the common standards and methodologies which exist for Ruby. This will help everyone to understand the purpose of the script lines and flow created by all the team members. This will increase productivity, reduce rework efforts as well as easy to analyze and debug the code.

The following pages covers the conventions for Ruby scripting that includes programming structures and patterns followed based on **Ruby-Cucumber** **best practices.**

# Indentation

Indentation will make the script clearer to understand. Mainly we should follow the same number of spaces for TAB throughout the project.

**1 TAB = 2 space characters**

The project team members should set the indentation (i.e., number of Tabs) in their IDE settings.

For RubyMine 5.4.2 the default settings for indentation for Ruby are pre-configured. If you want to set it manually, go to

*File >> Settings >> Code Style >> Ruby*, under the Ruby tab give the indent settings. Then click Apply >> ok, to save the settings.

# Line Length

The maximum number of characters that a line can have is 80.

## Wrapping Lines

If the script line go longer, wrap the lines instead of creating a long line, break after comma (,) or any operator.

## Passing parameters

Just 1 TAB stop to pass parameters from 2’nd line onwards.

*someMethod(longExpression1, longExpression2, longExpression3,*

*longExpression4, longExpression5)*

* Method calls as Parameters:

Just 2 TAB stops from start position to pass parameters from 2’nd line onwards. Here 3rd parameter is variable and 2nd parameter is some method call. To understand the difference, use extra TAB to pass variable in 3rd line.

*var = someMethod1 (longExpression1,*

*someMethod2 ( longExpression2, longExpression3))*

## Arithmetic Expressions

Always prefer to use the below way of breaking line.

* Break lines at higher levels such as after closed parenthesis
* Put arithmetic operators in new line ie. Break line after operator
* Align new line to beginning of expression of previous line

*longName1 = longName2 \* (longName3 + longName4 - longName5) +*

*4 \* longname6*

Avoid the below method:

*longName1 = longName2 \* (longName3 + longName4*

*- longName5) + 4 \* longname6;*

## Method Declarations

*#conventional indentation*

*someMethod(int anArg, Object anotherArg, String yetAnotherArg,*

*Object andStillAnother)*

*...*

*end*

*#indent 4 spaces to avoid very deep indents*

*private static synchronized horkingLongMethodName(int anArg,*

*Object anotherArg, String yetAnotherArg,*

*Object andStillAnother)*

*...*

*end*

In the above example, up to method name itself line is longer. If we keep the conventional indentation here, 2nd and 3rd lines will go so far. So keep 1 TAB in these cases.

## Conditional Statements

Use different indentations for conditions and statements inside loops.

*#don't use this indentation*

*if ((condition1 and condition2)*

*|| (condition3 and condition4)*

*||!(condition5 and condition6)) #BAD WRAPS*

*doSomethingAboutIt() #MAKE THIS LINE EASY TO MISS*

*end*

*#use this indentation instead*

*if ((condition1 and condition2)*

*|| (condition3 and condition4)*

*||!(condition5 and condition6))*

*doSomethingAboutIt()*

*end*

## Ternary Expression

Here are three acceptable ways to format ternary expressions. You can use any of the below ways.

*alpha = (aLongBooleanExpression) ? beta : gamma*

*alpha = (aLongBooleanExpression) ? beta :*

*gamma*

*alpha = (aLongBooleanExpression) ?*

*beta :*

*gamma*

# Comments

Ruby has two comments style: =begin…=end and '#'. You should use =begin…=end for Documentation Comments, and '#' for both Documentation Comments and Implementation Comments.

## Implementation comment

* **Single-Line Comments**

*# This is single line comment*.

* **Tailing Comments**

*if a == 2*

*true # special case*

*else*

*prime?(a) # work only for odd a*

*end*

## Documentation Comments

**Begin-end style:**

=begin

\* Name:

\* Description

\* Author:

\* Date:

\* License:

=end

# Definitions

## Initialization

Ruby variables have no 'definitions'. So, it is good to initialize variables.One initialization per line.

Example :

*level = 0*

*size = 0*

is preferred over

*level = size = 0*

And for string, *level\_name = “INFO”* instead of *level\_name = ‘INFO’* , as in Ruby strings enclosed with double quotes or single quotes makes no difference except for the below format of strings:

*str\_xpath = “//div[@id='project-603']/td[2]/a”*

here we should use both single and double quotes.

## White space

### Blank lines

Leave a blank line

* Between sections of a source file
* Between class and module definitions
* Between methods
* Before blocks or single-line comments
* Between logical sections inside a method to improve readability

### Blank spaces

* A keyword followed by a parenthesis should be separated by a space.

Example :

*while (foo.end?)*

*end*

* Leave a blank space between operators

*a + b ## Correct*

*a+b ## AVOID!*

*a +b ## AVOID! (Erroneous: interpreted as a(+b))*

*a += b + c*

*a = (a + b) / (c \* d)*

* Provide a space between the method arguments

*Foo (a=0, b=nil) ##Correct*

*Foo (a=0,b=nil) ##Avoid*

*Foo ( a=0,b=nil ) ##Avoid*

# Ruby structures

## Classes and Modules

Class and module names should be nouns; in mixed case with the first letter of each internal word capitalized. Try to keep your class names simple and descriptive. Use whole words-avoid acronyms and abbreviations (unless the abbreviation is much more widely used than the long form, such as URL or HTML).

Example : *class Raster, class Raster::ImageSprite*

## Methods

Methods should be verbs. All lower case ASCII letters with words separated by underscores ('\_')

Example: *run(), run\_fast(), obj.background\_color()*

## Variables

* Variable names should be all lower case ASCII letters with words separated by underscore ('\_') and variable names should not start with underscore (‘\_’).
* Variable names should be short yet meaningful.
* The choice of a variant name should be mnemonic- that is, designed to indicate to the casual observer the intent of its use.
* One-character variable names should not be used.

Example:

*count = 1*

*some\_char = SomeChar.new()*

*table\_width = 0.0*

## Constants

Ruby constants should be all upper case with words separated by underscores ('\_').

Example :

*MIN\_LENGTH = 1*

*DEFAULT\_HOST = "foo.example.com"*

# Identifier nomenclature

## Ruby structures

Various ruby programming structures used in test script :

* Numeric = num\_variablename (a = 10000)
* Array = arr\_arrayname
* Hash = hash\_hashname
* String = str\_name
* Decimal = dec\_name
* Boolean – bool\_name
* Class = ClassName
* Module = ModuleName
* Constants = CONSTANT\_NAME
* Object = classname\_objectname
* Methods = method\_name
* Json = json\_name

## Cucumber – Page Objects

Various Page-Object web element classes used in test script for object identification and manipulation :

* Button = button\_name
* Link = link\_name
* Image = img\_name
* List item = li\_name
* Div = div\_name
* Text area = textarea\_name
* Text field = textfield\_name
* Span = span\_name
* Ordered list = ol\_name
* Unordered list = ul\_name
* Select list = select\_name
* Paragraph = p\_name
* Table = table\_name
* Cell = cell\_name
* Form = form\_name
* Check box = check\_name
* File field = filefield\_name
* Radio button = radio\_name
* H1, H2, H3,.. = h1\_name, h2\_name, h3\_name,..

# Modification history

|  |  |  |  |
| --- | --- | --- | --- |
| Date | Author(s) | Modification History | Version |
| 16 Aug 2014 | Chandra sekaran R | Initial document draft | 0.1 |
| 18 Aug 2014 | Rajesh K | Reviewed and modified the documents | 1.0 |