Ruby CodeCount™

Counting Standard

University of Southern California

Center for Systems and Software Engineering

December , 2011

Revision Sheet

Date	Version	Revision Description	Author
3/1/2011	1.0	Original Release	CSSE
12/1/2011	1.1	Updated	CSSE
1/2/2013	1.2	Updated document template	CSSE

Table of Contents

1. Definitions

1.1. **SLOC –** Source Lines of Code is a unit used to measure the size of software program. SLOC counts the program source code based on a certain set of rules. SLOC is a key input for estimating project effort and is also used to calculate productivity and other measurements.

1.2. **Physical SLOC –** One physical SLOC is corresponding to one line starting with the first character and ending by a carriage return or an end-of-file marker of the same line, and which excludes the blank and comment line.

1.3. **Logical SLOC –** Lines of code intended to measure "statements", which normally terminate by a semicolon (C/C++, Java, C#) or a carriage return (VB, Assembly), etc. Logical SLOC are not sensitive to format and style conventions, but they are language-dependent.

1.4. **Data declaration line or data line –** A line that contains declaration of data and used by an assembler or compiler to interpret other elements of the program. Ruby does not contain any data declarations.

1.5. **Compiler Directives –** A statement that tells the compiler how to compile a program, but not what to compile. Ruby does not contain any compiler directivess

1.6. **Blank Line –** A physical line of code, which contains any number of white space characters (spaces, tabs, form feed, carriage return, line feed, or their derivatives).

1.7. **Comment Line –** A comment is defined as a string of zero or more characters that follow language-specific comment delimiter.

Ruby comment delimiter is "#". A whole comment line may span one line and does not contain any compliable source code. An embedded comment can co-exist with compliable source code on the same physical line. Banners and empty comments are treated as types of comments.

NOTE: The '#' character is also used for other purposes within Ruby, apart from delimiting comments.

1.8. **Executable Line of code –** A line that contains software instruction executed during runtime and on which a breakpoint can be set in a debugging tool. An instruction can be stated in a simple or compound form.

- An executable line of code may contain the following program control statements:
  - Selection statements (if, ? operator)
  - Iteration statements (for, while, do)
  - Empty statements (one or more “;”)
  - Jump statements (return, goto, last, next, exit function)
  - Expression statements (function calls, assignment statements, operations, etc.)
  - Block statements
- An executable line of code may not contain the following statements:
  - Whole line comments, including empty comments and banners
  - Blank lines

2. Checklist for source statement counts

PHYSICAL SLOC COUNTING RULES			
MEASUREMENT UNIT	ORDER OF PRECEDENCE	PHYSICAL SLOC	COMMENTS
Executable Lines	1	One Per line	Defined in 1.8
Non-executable Lines			
Declaration (Data) lines	2	One per line	Defined in 1.4
Compiler Directives	3	One per line	Defined in 1.5
Comments			Defined in 1.7
On their own lines	4	Not Included (NI)	
Embedded	5	NI	
Banners	6	NI	
Empty Comments	7	NI	
Blank Lines	8	NI	Defined in 1.6

LOGICAL SLOC COUNTING RULES				
NO.	STRUCTURE	ORDER OF PRECEDENCE	LOGICAL SLOC RULES	COMMENTS
R01	“for”, “while” or “if” statement	1	Count Once	“while” is an independent statement.
R02	*do {…} unitl (…); statement*	2	Count Once	Braces {…} and semicolon ; used with this statement are not counted.
R03	Block delimiters, braces {…}	3	Count once per pair of braces {..}, except where a closing brace is followed by a semicolon, i.e. };or an opening brace comes after a keyword “else”.	Braces used with R01 and R02 are not counted. Function definition is counted once since it is followed by {…}.

3. Examples

EXECUTABLE LINES

SELECTION Statement

ESS1 – if, elsif, else and nested if statements

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT
if <Boolean expression> [then]	if x != 0 then	1
<statements>	print “non-zero”	1
end	end	0
if <Boolean expression> [then]	if x > 0	1
<statements>	print “positive”	1
else	else	0
<statements>	print “negative”	1
end	end	0
if <Boolean expression> [then]	if x == 0	1
<statements>	print “zero”	1
elsif <Boolean expression> [then]	elsif x > 0	1
<statements>	print “positive”	1
else	else	0
<statements>	print “negative”	1
end	end	0
<statement> if <Boolean expr>	i = 1 if x > 10	2
<statement LHS> if <Boolean expr>	toss = if rand(2) == 1 then	1
<statement RHS1>	"heads"	1
else	else	0
<statement RHS2>	"tails"	1
end	end	0
NOTE: complexity is not considered, i.e. multiple “&&” or “||” as part of the expression.		

ESS2 – case-when-else-end

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT
case <expression>	case $num	1
when <constant 1>	when 0..10	1
<statements>	print “small num”	1
when <constant 2>	when 11..100	1
<statements>	print “large num”	1
else	else	0
<statements>	print “HUGE num”	1
end	end	0

ESS3 – unless statements

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT
unless <expression> [then]	unless $big	1
<statements>	print “small”	1
else	else	0
<statements>	print “big”	1
end	end	0
<statements> unless <Boolean expr>	print "Non-negative" unless x > 0	2

Iteration Statement

EIS1 – for

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT
for <control> in <expr> [do]	for i in [1, 2, 3] do	1
<statements>	print i*2	1
end	end	0

EIS2 – while

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT
while <Boolean expr> [do]	while $i < $num	1
<statements>	puts("Inside the loop i = #$i");	1
end	$i +=1;	1
	end	0
<statement> while <Boolean expr>	puts $1 += 2 while $i < 10	2
begin	begin	1
<statements>	puts("Inside the loop i = #$i");	1
end while <Boolean expr>	$i +=1;	1
	end while $i < $num	1

EIS3 – until

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT
until <Boolean expr> [do]	until $i > $num	1
<statements>	puts("Inside the loop i = #$i");	1
end	$i +=1;	1
	end	0
<statement> until <Boolean expr>	puts $1 += 2 until $i > 10	2
begin	begin	1
<statements>	puts("Inside the loop i = #$i");	1
end until <Boolean expr>	$i +=1;	1
	end until $i > $num	1

EIS4 – each iterator

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT
<collection>.each do <|variable|>	a.each do |i|	1
<statements>	puts i	1
end	end	0

EIS5 – collect iterator

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT
<collection> = <collection>.collect	b = a.collect	1
<collection> = <collection>.collect{|variable| expr}	c = a.collect{|x| 10*x}	2

JUMP Statement

EJS1 – throw

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT
throw <:labelname>	throw :greeting	1
throw <:labelname> <condition>	throw :greeting if TIME == 0	2

EJS2 – catch

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT
catch <:labelname> do	catch :greeting do	1
<statements>	puts(“Good morning!”);	1
end	end	0

EJS3 – return

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT
return <expr>	def test2	1
	i = 100; j = 200; k = 300	3
	return i, j, k;	1
	end	0
<condition> return	if x < 0 return	2

EJS4 – break

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT
break	if i > 2 then	1
	break	1
	end	0

EJS5 – next

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT
next	if i < 2 then	1
	next	1
	end	0

EJS6 – redo

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT
redo statement	redo	if i < 2 then
		redo
		end

EJS7 – retry

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT
begin	begin	1
<statements>	nil; # exception raised	1
rescue	rescue	1
<statements>	nil; # handles error	1
retry	retry # restart from begin block	1
end	end	0
retry <condition>	for i in 1..5	1
	retry if i > 2	2
	puts "Value of local variable is #{i}"	1
	end	0

EXPRESSION Statement

EES1 – assignment statement

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT
<name> = <value>	x = 3; x = y;	2
<name1> = <name2>	$num = 10	1
	@cust_name = name	1
	@@no_of_customers = 4	1
	PI = 3.14159	1

EES2 – empty statement (is counted as it is considered to be a placeholder for something)

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT
one or more “;”, but not following another statement	while i < 10 do	1
	puts(“Hello!”);	1
	;	1
	end	0

EES3 – function calls (general)

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT
<function_name> <parameters>	puts(“Hello!”)	1

EES4 – function calls (special)

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT
raise	begin	1
	puts 'I am before the raise.'	1
	raise 'An error has occurred.'	1
	puts 'I am after the raise.'	1
	rescue	1
	puts 'I am rescued.'	1
	end	0
require	require "Week"	1
include	class Decade	1
	include Week	1
	no_of_yrs=10	1
	def no_of_months	1
	puts Week::FIRST_DAY	1
	number = 10*12	1
	puts number	1
	end	0
	end	0

Block Statements

EBS1 – yield

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT

yield [var1, var2, …]	def test1
	yield
	end
	
	def test2
	yield 5
	end
		1
		1
		0
		
		1
		1
		0

EBS2 – do-end

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT
<method_invocation> do <statements> end	test1 do puts "You are in the block" end	1 1 0

EBS3 – {} delimiters

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT

<method_invocation> {
<statements>
}
	test2 {
	|i| puts "You are in the block #{i}"
	}
		1
		1
		0

EBS4 – begin-end

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT

BEGIN {
<statements>
}
 	
END {
<statements>
}
	BEGIN {
	puts "Initializing Ruby Program"
	}
	
	END {
	puts "Terminating Ruby Program"
	}
		1
		1
		0
		
		1
		1
		0

EBS5 – begin-rescue-else-ensure-end

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT
begin	begin	1
<statements>	puts "I'm not raising exception"	1
rescue	rescue Exception => e	1
<statements>	puts e.message	1
else	puts e.backtrace.inspect	1
<statements>	else	0
ensure	puts "Congratulations-- no errors!"	1
<statements>	ensure	1
end	puts "Ensuring execution"	1
	end	0

CLASS AND MODULE Statements

ECS1 – class

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT
class <class_name>	class Customer	1
<statements>	@@no_of_customers = 0	1
end	end	0

ECS2 – def

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT
def <method_name>[var = value]	def hello	1
<statements>	puts "Hello Ruby!"	1
end	end	0

ECS3 – undef

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT
undef <method_name>	undef hello	1

ECS4 – alias

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT
alias	alias <new_method> <old_method>	alias greeting hello
	alias <new_glob_var> <old_glob_var>	alias $angle $argument

ECS5 – super

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT
super	class Employee < Sample	1
	def initialize(fname, lname, position)	1
	super(fname,lname)	1
	@position = position	1
	end	0
	def to_s	1
	super + ", #@position"	1
	end	0
	end	0

ECS6 – module

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT
module <module_identifier>	module Trig	1
<statements>	PI = 3.141592654	1
end	def Trig.sin(x)	1
	nil; # Code for sine of x	1
	end	0
	def Trig.cos(x)	1
	nil; # Code for cosine of x	1
	end	0
	end	

Operators and Pseudo-Variables

EOP1 – defined?

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT
defined? [parameter]	defined? foo	1
	defined? $_	1
(parameter = variable, method_call, super, yield)	defined? puts	1
	defined? puts(bar)	1
	defined? super	1
	defined? yield	1

EOP1 – nil

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT
<variable> = nil;	@name = nil;	1
(functions as a variable with a logic value false)		
	def Trig.sin(x)	1
	nil # Code for sine of x	1
nil	end	0
(functions as a placeholder)		