ColdFusion CodeCount™

Counting Standard

University of Southern California

Center for Systems and Software Engineering

Spring 2010

Revision Sheet

Date	Version	Revision Description	Author
05/11/10	1.0	Original Release	CSSE

1.0 CHECKLIST FOR SOURCE STATEMENT COUNTS

PHYSICAL AND LOGICAL SLOC COUNTING RULES

Measurement Unit	Order of Precedence	Physical SLOC	Logical SLOC	Comments
Executable lines	1	One per line	See table below	Defined in 2.8
Non-executable lines				
Declaration (Data) lines	2	One per line	See table below	Defined in 2.4
Compiler directives	3	One per line	See table below	Defined in 2.5
Comments				Defined in 2.7
On their own lines	4	Not included (NI)	NI	
Embedded	5	NI	NI	
Banners	6	NI	NI	
Empty comments	7	NI	NI	
Blank lines	8	NI	NI	Defined in 2.6

Table 1 Physical and Logical SLOC Counting Counts

LOGICAL SLOC COUNTING RULES

No.	Structure	Order of Precedence	Logical SLOC Rules	Comments
R01	All ColdFusion tags beginning with “cf” with no nesting like <cfform>, <cfinput>, etc.	1	Count once.	All occurrences of such tags until corresponding end tags </> is assumed to be one logical statement
R02	<cfcase>, <cfloop>, <cfswitch>, either all tags having multiple steps of execution statement	2	Count once.	Logically different tags on the same line are to be counted independently
R03	Comment delimiter	3	Count once per combination of start tag and end tag statement, including empty statement.	Comments in ColdFusion are similar to HTML comments **<!--- this is a comment --->**
R04	Compiler directive	4	N/A	N/A

Table 2 Logical SLOC Counting Rules

2.0 DEFINITIONS

2.1 SLOC – Source Lines Of Code is a unit used to measure the size of a 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.

2.2 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.

2.3 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.

2.4 Data declaration line or data line – A line that contains declaration of data and used by a ColdFusion server to determine all ColdFusion variables declared in the program.

2.5 Compiler directive – A statement that tells the compiler how to compile a program, but not what to compile.

2.6 Blank line – A blank is a tab or space. What this actually means is - a blank is any chunk of white space between anything that is printable (a character or word). So a blank can be several spaces or tabs or a combination of multiples of the two.

2.7 Comment line – A comment is defined as a string of zero or more characters starting with <!--- and ending with --->.

2.8 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
  - Iteration statements (foreach, loop, switch)
  - Empty statements (pass)
  - Jump statements (return, goto, exit function)
  - Expression statements (function calls, assignment statements, operations, etc.)
  - Block statements
  - Database statements

  NOTE: See Section 3 of this document for examples of control statements.

- An executable line of code may not contain the following statements:
  - Data declaration (data) lines
  - Whole line comments, including empty comments and banners
  - Blank lines

3.0 EXAMPLES OF LOGICAL SLOC COUNTING

EXECUTABLE LINES

SELECTION STATEMENTS

ID	STATEMENT DESCRIPTION	GENERAL FORM	SPECIFIC EXAMPLE	SLOC COUNT
ESS1	cfif, cfelseif, cfelse and nested cfif statements	<cfif *expression*>	<cfif name == "USC">	1
		statements	some logic	0
		</cfif>	</cfif>	0
		<cfif *expression*>	<cfif password == "name">	1
		statements	some code	0
		<cfelse>	<cfelse>	1
		statements	statements	0
		</cfif>	</cfif>	0
		<cfif *expression*>	<cfif num > 0>	1
		statements	some code	0
		<cfelseif *expression*>	<cfelseif num < 0>	1
		statements	statements	0
		<cfelse>	<cfelse>	1
		statements	code	0
		</cfif>	</cfif>	0
ESS2	cfswitch, cfcase, cfdefaultcase	<cfswitch *expression = "expression"*>	<cfswitch expression = "#State#">	1
		<cfcase *value ="value"*>	<cfcase value="CA">	1
		HTML or CFML code	California	0
		</cfcase>	</cfcase>	0
		<cfdefaultcase>	<cfdefaultcase>	1
		HTML or CFML code	one of the other 47 states	0
		</cfdefaultcase>	</cfdefaultcase>	0
		</cfswitch>	</cfswitch>	0
ESS3	cftry-cfcatch	<cftry >	<cftry>	1
		do something	try: 1/0	0
		</cftry>	some code	0
		<cfcatch>	</cftry>	0
		cleanup	<cfcatch ZeroError>	1
		</cfcatch>	some code	0
			</cfcatch>	0

ITERATIONS STATEMENTS				
ID	**STATEMENT DESCRIPTION**	**GENERAL FORM**	**SPECIFIC EXAMPLE**	**SLOC COUNT**
EIS1	cfloop	<cfloop *expression*> *statements* </cfloop>	<cfloop index = "LoopCount" from = 1 to = 5> The loop index is <cfoutput>#LoopCount#</cfoutput>. </cfloop> <cfloop condition =”Expression”> <cfloop> <cfloop query =”Query name” startRow =”Start Row value” endRow = “ End Row value” </cfloop>	1 0 0 1 0 0 1 0 1 0 0 0

JUMP STATEMENTS (ARE COUNTED AS THEY INVOKE ACTION – PASS TO THE NEXT STATEMENT)				
ID	**STATEMENT DESCRIPTION**	**GENERAL FORM**	**SPECIFIC EXAMPLE**	**SLOC COUNT**
EJS1	cfreturn	<cfreturn *expression*>	<cfreturn true>	1
EJS2	cfbreak	<cfbreak>	<cfloop expression> <cfbreak> </cfloop>	1 1 0
EJS3	cfexit	<cfexit>	<cfloop expression> <cfexit> </cfloop>	1 1 0
EJS4	cfcontinue	<cfcontinue>	<cfloop expression> <cfcontinue> </cfloop>	1 1 0

EXPRESSION STATEMENTS				
ID	**STATEMENT DESCRIPTION**	**GENERAL FORM**	**SPECIFIC EXAMPLE**	**SLOC COUNT**
EES1	function call	<cffunction > name = “function name” description “ function description” return Type=”Data to be returned </cffuntion>	Any example	1
EES2	assignment statement	<cfset *variable_name=”value”*>	<cfset age=”22”>	1

EES3	CFScript	<cfscript>
		CFScript code
		</cfscript>
			<cfscript>
			for (i=1 ; i LE 4; i = i+1) {
			if(find("key",strings[i],1))
			break;
			}
			</cfscript>
				1
				0 (3 scr)
				0
				0
				0
				0

EES4	database query	<cfquery datasource=”DB name”	<cfquery datasource=“ABCD”	1