Document WG23 N0564

This is a cross check of the Joint Strike Fighter (JSF) C++ Coding Standards document 2RDU00001 REV C Dec 2005. (available at http://www.stroustrup.com/JSF-AV-rules.pdf) with draft 3 (N0461).

The document has been modified to align all section numbers with the proposed format of TR24772-1, i.e. by reducing the subclause count by 1 for every section 6.xx reference.

Currently, TR24772 categories that do not reference JSF rules: 6.14, 17, 19, 20, 22, 31, 43, 45, 47, 50, 51; 7.5-10

After the suggested changes (below), here are the TR24772 categories that do not reference JSF rules: 6.17, 20, 22, 31, 41, 45, 47, 50; 7.5-10

The term "AV" in the rules is for JSF AV.

Number	**Usage in TR24772**	**JSF Rule**	**Adjudication** "X" means that JSF rule already appears in TR24772, Style or Performance issue means that the issue isn't a vulnerability, remainder are suggested places for inclusion in TR24772	**Comments - SGM**
1.		**AV Rule 1** Any one function (or method) **will** contain no more than 200 logical source lines of code (L-SLOCs).	Style issue	
2.	6.46 [NYY]	**AV Rule 2** There **shall not** be any self-modifying code.	X	
3.		**AV Rule 3** All functions **shall** have a cyclomatic complexity number of 20 or less.	Style issue	
4.		**AV Rule 4** To break a **"should"** rule, the following approval must be received by the developer: • approval from the software engineering lead (obtained by the unit approval in the developmental CM tool)	Style issue	

5.		**AV Rule 5** To break a **“will”** or a **“shall”** rule, the following approvals must be received by the developer: • approval from the software engineering lead (obtained by the unit approval in the developmental CM tool) • approval from the software product manager (obtained by the unit approval in the developmental CM tool)	Style issue	
6.		**AV Rule 6** Each deviation from a **“shall”** rule **shall** be documented in the file that contains the deviation). Deviations from this rule **shall not** be allowed, AV Rule 5 notwithstanding.	Style issue	
7.		**AV Rule 7** Approval **will not** be required for a deviation from a **“shall”** or **“will”** rule that complies with an **exception** specified by that rule.	Style issue	
8.	6.55 [MEM]	**AV Rule 8** All code **shall** conform to ISO/IEC 14882:2002(E) standard C++.	X	
9.		**AV Rule 9 (MISRA Rule 5, Revised)** Only those characters specified in the C++ basic source character set **will** be used.	Style issue	
10.		**AV Rule 10 (MISRA Rule 6)** Values of character types will be restricted to a defined and documented subset of ISO 10646-1.	Add to 6.17 [NAI] Choice of Clear Names, maybe 7.11 [HTS] Resource Names	Possibly only a style issue. Reflect and come back.
11.	6.55 [MEM]	**AV Rule 11 (MISRA Rule 7)** Trigraphs **will not** be used.	X	C and C++ issue. Part 1 issue is only an example. Consider moving JSF rules (and Ada style and guide rules) that are clearly language-specific to the relative annex. Consider adding reference to CERT C coding guidelines.
12.		**AV Rule 12 (Extension of MISRA Rule 7)** The following digraphs **will not** be used: Alternative – Primary – alternative – Primary <% – { – :> –] %> – } – %: – # <: – [– %:%: – ##	Style issue	
13.		**AV Rule 13 (MISRA Rule 8)** Multi-byte characters	Add to 6.54 [FAB]	Guidance - don’t mix multi-byte

		and wide string literals **will not** be used.	Implementation-defined Behaviour	and wide string literals. Consider adding to Part 3 and Part C++.
14.		**AV Rule 14** Literal suffixes **shall** use uppercase rather than lowercase letters.	Style issue	
15.	6.8 [HCB] 6.9 [XYZ] 6.10 [XYW] 6.15 [FIF] 6.16 [PIK]	**AV Rule 15 (MISRA Rule 4, Revised)** Provision **shall** be made for run-time checking (defensive programming).	X	
16.	6.43 [TRJ]	**AV Rule 16** Only DO-178B level A [15] certifiable or SEAL 1 C/C++ libraries **shall** be used with safety-critical (i.e. SEAL 1) code [13].	X	Disagree with the reference in TR24772-1. TRJ is about passing arguments to library functions, JSF AV 16 is about certifiable libraries.
17.	6.52 [BQF] 6.53 [EWF] 6.54 [FAB]	**AV Rule 17 (MISRA Rule 119)** The error indicator *errno* **shall not** be used.	X	Propose to remove AV Rule 17 from Part 1 and put in Part 3, Part C++. Probably doesn't belong in BQF, EWF, FAB.
18.	6.43 [TRJ], 6.52 [BQF] 6.53 [EWF] 6.54 [FAB]	**AV Rule 18 (MISRA Rule 120)** The macro *offsetof,* in library <stddef.h>, **shall not** be used.	X	Propose to remove AV Rule 17 from Part 1 and put in Part 3, Part C++. Wrong for C, ok for C++.
19.	6.43 [TRJ] 6.52 [BQF] 6.53 [EWF] 6.54 [FAB]	**AV Rule 19 (MISRA Rule 121)** <locale.h> and the *setlocale* function **shall not** be used.	X	Does not apply?
20.	6.32 [CSJ] 6.43 [TRJ] 6.52 [BQF] 6.53 [EWF] 6.54 [FAB]	**AV Rule 20 (MISRA Rule 122)** The *setjmp* macro and the *longjmp* function **shall not** be used.	X	Include in the transfer of control vulnerability(s)
21.	6.43 [TRJ] 6.52 [BQF] 6.53 [EWF] 6.54 [FAB]	**AV Rule 21 (MISRA Rule 123)** The signal handling facilities of <signal.h> **shall not** be used.	X	Signal is not thread(task)-aware, therefore problematic.
22.	6.43 [TRJ] 6.52 [BQF] 6.53 [EWF] 6.54 [FAB]	**AV Rule 22 (MISRA Rule 124, Revised)** The input/output library <stdio.h> **shall not** be used.	X	Propose to remove from Part 1 and Part 3, only (maybe) for Part C++.

23.	6.43 [TRJ] 6.52 [BQF] 6.53 [EWF] 6.54 [FAB]	**AV Rule 23 (MISRA Rule 125)** The library functions *atof, atoi* and *atol* from library <stdlib.h> **shall not** be used.	X	lack error indicators. Recommend better alternatives. C-specific.
24.	6.37 [REU] 6.43 [TRJ] 6.52 [BQF] 6.53 [EWF] 6.54 [FAB]	**AV Rule 24 (MISRA Rule 126)** The library functions *abort, exit, getenv* and *system* from library <stdlib.h> **shall not** be used.	X	Finalize issue. Move to a finalize vulnerability when (if) written for abort and exit.
25.	6.8 [HCB] 6.43 [TRJ] 6.52 [BQF] 6.53 [EWF] 6.54 [FAB]	**AV Rule 25 (MISRA Rule 127)** The time handling functions of library <time.h> **shall not** be used.	X	We may need a vulnerability associated with time handling, clock time, real time, differences in time between program partitions.
26.	6.48 [NMP]	**AV Rule 26** Only the following pre-processor directives **shall** be used: 1. #ifndef 2. #define 3. #endif 4. #include	X	Propose considering AV Rule 26 for Part 1 and put in Part 3.
27.	6.48 [NMP]	**AV Rule 27** #ifndef, #define and #endif **will** be used to prevent multiple inclusions of the same header file. Other techniques to prevent the multiple inclusions of header files **will not be** used.	X	Check TR 224772-3 [NMP] to ensure addressed adequately.
28.	6.48 [NMP]	**AV Rule 28** The #ifndef and #endif pre-processor directives **will** only be used as defined in AV Rule 27 to prevent multiple inclusions of the same header file.	X	Check TR 224772-3 [NMP] to ensure addressed adequately.
29.	6.48 [NMP]	**AV Rule 29** The *#define* pre-processor directive **shall not** be used to create inline macros. Inline functions **shall** be used instead.	X	Check TR 224772-3 [NMP] to ensure addressed adequately.
30.	6.48 [NMP]	**AV Rule 30** The *#define* pre-processor directive **shall not** be used to define constant values. Instead, the *const* qualifier **shall** be applied to variable declarations to specify constant values.	X	Check TR 224772-3 [NMP] to ensure addressed adequately.
31.	6.48 [NMP]	**AV Rule 31** The *#define* pre-processor directive **will** only be used as part of the technique to prevent multiple inclusions of the same header file.	X	Check TR 224772-3 [NMP] to ensure addressed adequately.

32.	6.48 [NMP]	**AV Rule 32** The *#include* pre-processor directive **will** only be used to include header (*.h) files.	X	Check TR 224772-3 [NMP] to ensure addressed adequately.
33.		**AV Rule 33** The *#include* directive **shall** use the *<filename.h>* notation to include header files.	Style issue	OK
34.		**AV Rule 34** Header files **should** contain logically related declarations only.	Style issue	OK
35.		**AV Rule 35** A header file **will** contain a mechanism that prevents multiple inclusions of itself.	Should 6.36 Recursion be expanded to include this?	This is not the classic style of recursion. Should go in 6.48 Pre-processor directives. Our guidance should include indirect inclusions (A -> B -> A). Is there really a vulnerability in executable code that can result?
36.		**AV Rule 36** Compilation dependencies **should** be minimized when possible.	Style issue	OK
37.		**AV Rule 37** Header (include) files **should** include only those header files that are required for them to successfully compile. Files that are only used by the associated .cpp file should be placed in the .cpp file—not the .h file.	Style issue	OK
38.		**AV Rule 38** Declarations of classes that are only accessed via pointers (*) or references (&) **should** be supplied by *forward headers* that contain only *forward declarations*.	Style/performance issue	Not performance issue.
39.		**AV Rule 39** Header files (*.h) **will not** contain non-const variable definitions or function definitions. (See also AV Rule 139.)	Style issue	More than a style case. Edge case exists?
40.		**AV Rule 40** Every implementation file **shall** include the header files that uniquely define the inline functions, types, and templates used.	Style issue, but inconsistency could be a problem ala Heartbleed. Suggest adding an “inconsistency” category	Not clear where “inconsistency: heading would go – new section or in 6.50? Version skews. Maybe a new vulnerability. Include One definition rule. Added, SM, 20150629. This may be a new sect 7 vulnerability - avoidance of incompatible versions between oarts of a system - open ended.

41.		**AV Rule 41** Source lines **will** be kept to a length of 120 characters or less.	Style issue	OK
42.		**AV Rule 42** Each expression-statement **will** be on a separate line.	Style issue	OK
43.		**AV Rule 43** Tabs **should** be avoided.	Style issue	Agreed for C and C++, but this may be a vulnerability. Some languages use indentation exclusively to tell the language processor when nested indentation ends. Some may use spaces, and some may use tabs. Python may worry about this.
44.		**AV Rule 44** All indentations **will** be at least two spaces and be consistent within the same source file.	Style issue	Same as previous.
45.		**AV Rule 45** All words in an identifier **will** be separated by the '_' character.	Style issue	Check
46.	7.11 [HTS]	**AV Rule 46 (MISRA Rule 11, Revised)** User-specified identifiers (internal and external) **will not** rely on significance of more than 64 characters.	X	Add [NAI]
47.		**AV Rule 47** Identifiers **will not** begin with the underscore character '_'.	Style issue	Much more than a style issue. Most libraries are C-based and the convention is that library-level global names begin with "_", hence this avoids replacing a library function with something local. This may be part of Namespace Issues vulnerability.
48.	6.17 [NAI]	**AV Rule 48** Identifiers **will not** differ by: • Only a mixture of case • The presence/absence of the underscore character • The interchange of the letter 'O', with the number '0' or the letter 'D' • The interchange of the letter 'I', with the number '1' or the letter 'l' • The interchange of the letter 'S' with the number '5' • The interchange of the letter 'Z' with the number 2 • The interchange of the letter 'n' with the letter 'h'.	X	Check. OK.

49.	6.17 [NAI]	**AV Rule 49** All acronyms in an identifier **will** be composed of uppercase letters.	X	Check OK
50.	6.17 [NAI]	**AV Rule 50** The first word of the name of a class, structure, namespace, enumeration, or type created with *typedef* **will** begin with an uppercase letter. All others letters **will** be lowercase.	X	Check OK
51.	6.17 [NAI] 7.11 [HTS]	**AV Rule 51** All letters contained in function and variable names **will** be composed entirely of lowercase letters.	X	Check OK
52.	6.17 [NAI]	**AV Rule 52** Identifiers for constant and enumerator values **shall** be lowercase.	X	OK
53.	6.17 [NAI] 7.11 [HTS]	**AV Rule 53** Header files **will** always have a file name extension of ".h".	X	Not a language issue. Remove from [NAI]
54.	6.17 [NAI] 7.11 [HTS]	**AV Rule 54** Implementation files **will** always have a file name extension of ".cpp".	X	Not a language issue. Remove from [NAI]
55.	6.17 [NAI] 7.11 [HTS]	**AV Rule 55** The name of a header file **should** reflect the logical entity for which it provides declarations.	X	Not a language issue. Remove from [NAI]
56.	6.17 [NAI] 7.11 [HTS]	**AV Rule 56** The name of an implementation file **should** reflect the logical entity for which it provides definitions and have a ".cpp" extension (this name will normally be identical to the header file that provides the corresponding declarations.)	X	Not a language issue. Remove from [NAI]
57.		**AV Rule 57** The public, protected, and private sections of a class **will** be declared in that order (the public section is declared before the protected section which is declared before the private section).	Style issue	OK
58.		**AV Rule 58** When declaring and defining functions with more than two parameters, the leading parenthesis and the first argument **will** be written on the same line as the function name. Each additional argument **will** be written on a separate line (with the closing parenthesis directly after the last argument).	Style issue	OK
59.	6.28 [EOJ]	**AV Rule 59 (MISRA Rule 59, Revised)** The statements forming the body of an *if, else if*, *else, while, do…while* or *for* statement **shall** always be enclosed in	X	OK

		braces, even if the braces form an empty block.		
60.		**AV Rule 60** Braces ("{}") which enclose a block **will** be placed in the same column, on separate lines directly before and after the block.	Style issue	OK
61.		**AV Rule 61** Braces ("{}") which enclose a block **will** have nothing else on the line except comments (if necessary).	Style issue	OK
62.		**AV Rule 62** The dereference operator '*' and the address-of operator '&' **will** be directly connected with the type-specifier.	Style issue	OK
63.		**AV Rule 63** Spaces **will not** be used around '.' or '->', nor between unary operators and operands.	Style issue	OK
64.		**AV Rule 64** A class interface **should** be complete and minimal.	Style issue	OK
65.		**AV Rule 65** A structure **should** be used to model an entity that does not require an invariant.	Style issue	OK
66.		**AV Rule 66** A class **should** be used to model an entity that maintains an invariant.	Style issue	OK
67.		**AV Rule 67** Public and protected data **should** only be used in structs—not classes.	Style issue	Extend to the general visibility issue. Do we have a vulnerability that covers setting state from outside the local region. {Vulnerability Locking and visibility interact in concurrency (Java)}
68.		**AV Rule 68** Unneeded implicitly generated member functions **shall** be explicitly disallowed.	Style issue	Not style. Vulnerability about implicitly generated functions such as assignment or having side effects. Violates the Liskov substitution principle. For discussion later.. Note captured in [RIP]
69.		**AV Rule 69** A member function that does not affect the state of an object (its instance variables) **will** be declared *const*.	Style issue	Should be static if possible
70.		**AV Rule 70** A class **will** have friends only when a function or object requires access to the private	Style issue	This may be more than style. See AV Rule 67.

		elements of the class, but is unable to be a member of the class for logical or efficiency reasons.		
71.	6.21 [LAV]	**AV Rule 71** Calls to an externally visible operation of an object, other than its constructors, **shall not** be allowed until the object has been fully initialized.	X	
72.		**AV Rule 72** The invariant for a class **should** be: • a part of the postcondition of every class constructor, • a part of the precondition of the class destructor (if any), • a part of the precondition and postcondition of every other publicly accessible operation.	Style issue	
73.		**AV Rule 73** Unnecessary default constructors **shall not** be defined. (See also AV Rule 143).	Add to 6.22 Initialization of Variables [LAV], may need to add new text to 6.24 to cover this instance	Drop reference to AV 73. Claim that AV rule is problematic. AV rule example is not C++ legal code.
74.		**AV Rule 74** Initialization of nonstatic class members **will** be performed through the member initialization list rather than through assignment in the body of a constructor.	Add to 6.23 Initialization of Variables [LAV]	We need to determine if this is C++-specific or good general guidance. Don't reference from Part 1[LAV] but do reference from Part C++ [LAV]
75.		**AV Rule 75** Members of the initialization list **shall** be listed in the order in which they are declared in the class.	Style issue	Most languages have some sort of textual order dependency, and with languages that permit overriding, a different evaluation order could change the meaning of programs
76.		**AV Rule 76** A copy constructor and an assignment operator **shall** be declared for classes that contain pointers to data items or nontrivial destructors.	Doesn't seem to fit any category cleanly, so either a category needs to be expanded to include it or a new category created.	Agreed. This may be a new vulnerability. Distinction between objects that require shallow copying and deep copying. AI - Clive.
77.		**AV Rule 77** A copy constructor **shall** copy all data members and bases that affect the class invariant (a data element representing a cache, for example, would not need to be copied).	Add to 6.41 Inheritance [RIP], or could add to a new inconsistency category.	This is really saying that the copy constructor must preserve class invariance. The vulnerability is listed, but the programmer mitigation needs to be added to 6.41.5 AI - Steve.
78.		**AV Rule 78** All base classes with a virtual function	Add to 6.14 Dangling	Disagree with PIK. What does

		shall define a virtual destructor.	Reference to Heap [XYK], 6.16 Using Shift Operations for Multiplication and Division [PIK]	shifting for multiplication or division have to do with memory allocation? Remove from [PIK]. Maybe add to Inheritance[RIP].
79.		**AV Rule 79** All resources acquired by a class **shall** be released by the class's destructor.	Add to 6.14 Dangling Reference to Heap [XYK], 6.16 Using Shift Operations for Multiplication and Division [PIK]	In 6.14.5, final bullet, – add "including the release in a class destructor of all resources acquired by the class" Clearly not PIK - remove.
80.		**AV Rule 80** The default copy and assignment operators **will** be used for classes when those operators offer reasonable semantics.	Style issue	Disagree. This goes with AV Rules 76 and 77. AV 80 conflicts with 76. Remove reference to AV 80.

81.		**AV Rule 81** The assignment operator **shall** handle self-assignment correctly
		
		AV Rule 81
		Self-assignment must be handled appropriately by the assignment operator. Example A illustrates a potential problem, whereas Example B illustrates an acceptable approach.
		Example A: Although it is not necessary to check for self-assignment in all cases, the following example illustrates a context where it would be appropriate.
		Base &operator= (const Base &rhs)
		{
		release_handle (my_handle); // Error: the resource referenced by myHandle is
		my_handle = rhs.myHandle; // erroneously released in the self-assignment case.
		*return *this;*
		}
			Could be a new category.	The general term is "idempotent". Add to 6.41 [RIP] and ensure that the idempotency requirement is included.

		Example B: One means of handling self-assignment is to check for self-assignment before further processing continues as illustrated below.
		Base &operator= (const Base& rhs)
		{
		if (this != &rhs) // Check for self assignment before continuing.
		{
		release_handle(my_handle); // Release resource.
		my_handle = rhs.my_handle; // Assign members (only one member in class).
		}
		else
		{
		}
		*return *this;*
		}		

82.	6.11 [HFC]	**AV Rule 82** An assignment operator **shall** return a reference to **this*.	X	Cannot find Rule 82 in the TR. This is a problem and rule specific to C++ and OO languages that use pointers. Add to Part C++? Remove from part 1. General OOP rule that says that a redefinition must obey the Liskov substitution rule.
83.	6.11 [HFC]	**AV Rule 83** An assignment operator **shall** assign all data members and bases that affect the class invariant (a data element representing a cache, for example, would not need to be copied).	X	Cannot find Rule 83 in the TR. Remove from -1 or put in type system vulnerability.
84.	6.51 [BRS]	**AV Rule 84** Operator overloading **will** be used sparingly and in a conventional manner.	X	
85.		**AV Rule 85** When two operators are opposites (such as == and !=), both **will** be defined and one **will** be defined in terms of the other.	Style issue	Far more than style. This is an easy way to introduce an exploitable hole in the application. I believe that static

				analysis tools check for this. Add a rule to [???] AI Steve. Ensure that you meet language expectations on constraints. Include the complete algebra.
86.	6.41 [RIP] 6.51 [BRS]	**AV Rule 86** Concrete types **should** be used to represent simple independent concepts.	X	
87.	6.41 [RIP]	**AV Rule 87** Hierarchies **should** be based on abstract classes.	X	
88.	6.41 [RIP] 6.51 [BRS]	**AV Rule 88** Multiple inheritance **shall** only be allowed in the following restricted form: *n* interfaces plus *m* private implementations, plus at most one protected implementation.	X	
89.	6.41 [RIP]	**AV Rule 89** A base class **shall not** be both virtual and non-virtual in the same hierarchy.	X	Add this mitigation to 6.41.5. We need more comprehensive vulnerability discussion on OOP.
90.	6.41 [RIP]	**AV Rule 90** Heavily used interfaces **should** be minimal, general and abstract.	X	
91.	6.41 [RIP]	**AV Rule 91** Public inheritance **will** be used to implement "is-a" relationships.	X	
92.	6.4 [RIP]	**AV Rule 92** A subtype (publicly derived classes) **will** conform to the following guidelines with respect to all classes involved in the polymorphic assignment of different subclass instances to the same variable or parameter during the execution of the system: • Preconditions of derived methods must be at least as weak as the preconditions of the methods they override. • Postconditions of derived methods must be at least as strong as the postconditions of the methods they override. In other words, subclass methods must expect less and deliver more than the base class methods they override. This rule implies that subtypes will conform to the Liskov Substitution Principle.	X	We need to decide how much to expand 6.4 [RIP] - AI Erhard. Include down to AV 96.
93.	6.4 [RIP]	**AV Rule 93** "has-a" or "is-implemented-in-terms-of" relationships **will** be modeled through membership or	X	We need to decide how much to expand 6.4 [RIP]

		non-public inheritance.		
94.	6.4 [RIP]	**AV Rule 94** An inherited nonvirtual function **shall not** be redefined in a derived class.	X	We need to decide how much to expand 6.4 [RIP]
95.	6.4 [RIP]	**AV Rule 95** An inherited default parameter **shall never** be redefined.	X	We need to decide how much to expand 6.41 [RIP]
96.	6.4 [RIP]	**AV Rule 96** Arrays **shall not** be treated polymorphically.	X	We need to decide how much to expand 6.41 [RIP]
97.	6.4 [RIP] 6.51 [BRS]	**AV Rule 97** Arrays **shall not** be used in interfaces. Instead, the *Array* class should be used.	X	Remove from part 1, except for [???] bounds checking.
98.		**AV Rule 98** Every nonlocal name, except main(), **should** be placed in some namespace.	Style issue	This is more than style. If the language has global namespace and packaged namespaces, then hiding or overloading is more controllable if global is not used. AI - steve Find and place in Part 1.
99.		**AV Rule 99** Namespaces **will not** be nested more than two levels deep.	Style issue	
100.		**AV Rule 100** Elements from a namespace **should** be selected as follows: • *using declaration* or *explicit qualification* for few (approximately five) names, • *using directive* for many names.	Style issue	
101.	6.40 [SYM]	**AV Rule 101** Templates **shall** be reviewed as follows: 1. with respect to the template in isolation considering assumptions or requirements placed on its arguments. 2. with respect to all functions instantiated by actual arguments.	X	
102.	6.40 [SYM]	**AV Rule 102** Template tests **shall** be created to cover all actual template instantiations.	X	
103.	6.40 [SYM]	**AV Rule 103** Constraint checks **should** be applied to template arguments.	X	
104.	6.40 [SYM]	**AV Rule 104** A template specialization **shall be** declared before its use.	X	
105.	6.40 [SYM]	**AV Rule 105** A template definition's dependence on its instantiation contexts **should** be minimized.	X	

106.		**AV Rule 106** Specializations for pointer types **should** be made where appropriate.	Style/performance issue	Disagree with guidance. Our guidance should be. - don't specialize generics or templates.
107.		**AV Rule 107 (MISRA Rule 68)** Functions **shall** always be declared at file scope.	Style issue	
108.	6.34 [OTR]	**AV Rule 108 (MISRA Rule 69)** Functions with variable numbers of arguments **shall not** be used.	X	
109.		**AV Rule 109** A function definition **should not** be placed in a class specification unless the function is intended to be inlined.	Style issue	
110.		**AV Rule 110** Functions with more than 7 arguments **will** not be used.	Style issue	
111.		**AV Rule 111** A function **shall not** return a pointer or reference to a non-static local object.	Add to 6.32 [CSJ] Passing Parameters and Return Values	
112.		**AV Rule 112** Function return values **should not** obscure resource ownership.	Add to 6.32 [CSJ] Passing Parameters and Return Values	This is not covered in 6.32, but maybe should be. Consider adding an ownership model to XYK (6.14)
113.	6.31 [EWD]	**AV Rule 113 (MISRA Rule 82, Revised)** Functions **will** have a single exit point.	X, Also add to 6.32 [CSJ] Passing Parameters and Return Values	No. Leave as-is.
114.		**AV Rule 114 (MISRA Rule 83, Revised)** All exit points of value-returning functions **shall** be through return statements.	Add to 6.31 [EWD] Structured Programming, 6.32 [CSJ] Passing Parameters and Return Values	Agree.
115.	6.36 [OYB]	**AV Rule 115 (MISRA Rule 86)** If a function returns error information, then that error information **will** be tested.	X	
116.	6.32 [CSJ]	**AV Rule 116** Small, concrete-type arguments (two or three words in size) **should** be passed by value if changes made to formal parameters should not be reflected in the calling function.	X	
117.	6.32 [CSJ]	**AV Rule 117** Arguments **should** be passed by reference if NULL values are not possible: **AV Rule 117.1** An object should be passed as *const T&*	X	C++ specific. If used, do it in Part C++. Remove references from [CSJ]. OK

		if the function should not change the value of the object. **AV Rule 117.2** An object should be passed as *T&* if the function may change the value of the object.		
118.	6.32 [CSJ]	**AV Rule 118** Arguments **should** be passed via pointers if NULL values are possible: **AV Rule 118.1** An object should be passed as *const T** if its value should not be modified. **AV Rule 118.2** An object should be passed as *T** if its value may be modified.	X	Ditto. OK
119.	6.35 [GDL]	**AV Rule 119 (MISRA Rule 70)** Functions **shall not** call themselves, either directly or indirectly (i.e. recursion **shall not** be allowed).	X	
120.	6.20 [YOW]	**AV Rule 120** Overloaded operations or methods **should** form families that use the same semantics, share the same name, have the same purpose, and that are differentiated by formal parameters.	X	After meeting. NO. YOW is about hiding names, not Overloading. What is the vulnerability?
121.		**AV Rule 121** Only functions with 1 or 2 statements **should** be considered candidates for inline functions.	Style issue	
122.		**AV Rule 122** Trivial accessor and mutator functions **should** be inlined.	Style issue	
123.		**AV Rule 123** The number of accessor and mutator functions **should** be minimized.	Style issue	
124.		**AV Rule 124** Trivial forwarding functions **should** be inlined.	Style issue	
125.		**AV Rule 125** Unnecessary temporary objects **should** be avoided.	Style issue	
126.		**AV Rule 126** Only valid C++ style comments (//) **shall** be used.	Style issue	Disagree that this is a style issue. Block-oriented comments are susceptible to having disabled code (hidden in comments) introduced if block comment terminator is moved. AI Steve - see if existing vulnerability, or propose one.
127.	6.26 [XYQ] 7.3 [BVQ]	**AV Rule 127** Code that is not used (commented out) **shall** be deleted.	X	

128.		**AV Rule 128** Comments that document actions or sources (e.g. tables, figures, paragraphs, etc.) outside of the file being documented **will** not be allowed.	Style issue	
129.		**AV Rule 129** Comments in header files **should** describe the externally visible behavior of the functions or classes being documented.	Style issue	
130.		**AV Rule 130** The purpose of every line of executable code **should** be explained by a comment, although one comment may describe more than one line of code.	Style issue	
131.		**AV Rule 131** One **should** avoid stating in comments what is better stated in code (i.e. do not simply repeat what is in the code).	Style issue	
132.		**AV Rule 132** Each variable declaration, typedef, enumeration value, and structure member **will** be commented.	Style issue	
133.		**AV Rule 133** Every source file **will** be documented with an introductory comment that provides information on the file name, its contents, and any program-required information (e.g. legal statements, copyright information, etc).	Style issue	
134.		**AV Rule 134** Assumptions (limitations) made by functions **should** be documented in the function's preamble.	Style issue	
135.	6.20 [YOW]	**AV Rule 135 (MISRA Rule 21, Revised)** Identifiers in an inner scope **shall not** use the same name as an identifier in an outer scope, and therefore hide that identifier.	X	
136.	6.20 [YOW]	**AV Rule 136 (MISRA Rule 22, Revised)** Declarations **should** be at the smallest feasible scope.	X	
137.	6.20 [YOW]	**AV Rule 137 (MISRA Rule 23)** All declarations at file scope **should** be static where possible.	X	
138.	6.20 [YOW]	**AV Rule 138 (MISRA Rule 24)** Identifiers **shall not** simultaneously have both internal and external linkage in the same translation unit.	X	Check this. Clearly not YOW. Remove from [YOW] as a reference. Example in AV 138 is YOW but there may be other vulnerabilities. Reference from "namespace" AI Clive to check.

139.	6.20 [YOW]	**AV Rule 139 (MISRA Rule 27)** External objects **will** not be declared in more than one file.	X	Disagree that this is name reuse. Reference "namespace" vulnerability.
140.		**AV Rule 140 (MISRA Rule 28, Revised)** The *register* storage class specifier **shall not** be used.	Style issue	
141.		**AV Rule 141** A class, structure, or enumeration **will** not be declared in the definition of its type.	Style issue	
142.		**AV Rule 142 (MISRA Rule 30, Revised)** All variables **shall** be initialized before use. (See also AV Rule 136, AV Rule 71, and AV Rule 73, and AV Rule 143 concerning declaration scope, object construction, default constructors, and the point of variable introduction respectively.)	Add to 6.23 Initialization of Variables	
143.	6.22 [LAV]	**AV Rule 143** Variables **will not** be introduced until they can be initialized with meaningful values. (See also AV Rule 136, AV Rule 142, and AV Rule 73 concerning declaration scope, initialization before use, and default constructors respectively.)	X	
144.		**AV Rule 144 (MISRA Rule 31)** Braces **shall** be used to indicate and match the structure in the non-zero initialization of arrays and structures.	Style issue	
145.	6.5 [CCB]	**AV Rule 145 (MISRA Rule 32)** In an enumerator list, the '=' construct **shall not** be used to explicitly initialize members other than the first, unless all items are explicitly initialized.	X	Not in 6.5. Remove reference from Part 1.
146.	6.4 [PLF]	**AV Rule 146 (MISRA Rule 15)** Floating point implementations **shall** comply with a defined floating point standard.	X	
147.	6.3 [STR] 6.4 [PLF] 6.22 [LAV]	**AV Rule 147 (MISRA Rule 16)** The underlying bit representations of floating point numbers **shall not** be used in any way by the programmer.	X	
148.	6.2 [IHN] 6.27 [CLL]	**AV Rule 148** Enumeration types **shall** be used instead of integer types (and constants) to select from a limited series of choices.	X	
149.		**AV Rule 149 (MISRA Rule 19)** Octal constants (other than zero) **shall not** be used.	Style issue	

150.		**AV Rule 150** Hexadecimal constants **will** be represented using all uppercase letters.	Style issue	
151.	7.4 [KLK]	**AV Rule 151** Numeric values in code **will not** be used; symbolic values **will** be used instead.	X	We either have an existing vulnerability or need one that says no magic numbers and why. AI - Steve. - KLK (7.4) covers special numbers, but not magic numbers .Done.
152.		**AV Rule 152** Multiple variable declarations **shall not** be allowed on the same line.	Style issue	
153.	6.38 [AMV]	**AV Rule 153 (MISRA Rule 110, Revised)** Unions **shall not** be used.	X	
154.	6.3 [STR]	**AV Rule 154 (MISRA Rules 111 and 112, Revised)** Bit-fields **shall** have explicitly unsigned integral or enumeration types only.	X	
155.	6.3 [STR]	**AV Rule 155** Bit-fields **will not** be used to pack data into a word for the sole purpose of saving space.	X	
156.		**AV Rule 156 (MISRA Rule 113)** All the members of a structure (or class) **shall** be named and shall only be accessed via their names.	Doesn't seem to fit any category and is something that is error prone. Either expand one of the current categories (not clear which one) or add a new category.	C specific. Add to Part 3 and to Part C++.
157.	6.24 [SAM]	**AV Rule 157 (MISRA Rule 33)** The right hand operand of a && or || operator **shall not** contain side effects.	X	
158.	6.24 [SAM]	**AV Rule 158 (MISRA Rule 34)** The operands of a logical && or || **shall** be parenthesized if the operands contain binary operators.	X	
159.		**AV Rule 159** Operators ||, &&, and unary & **shall not** be overloaded.	Style issue	C++-specific. OK to put in Part C++. OK if we add a vulnerability about preserving the semantics of overloaded operator. Note here that overloading makes the shortcut action impossible.
160.	6.25 [KOA]	**AV Rule 160 (MISRA Rule 35, Modified)** An assignment expression **shall** be used only as the	X	

		expression in an expression statement.		
161.		**No rule listed**	No rule listed	
162.		**AV Rule 162** Signed and unsigned values **shall not** be mixed in arithmetic or comparison operations.	Add to 6.6 [FLC] Numeric Conversion Errors	Add reference from [FLC]
163.		**AV Rule 163** Unsigned arithmetic **shall not** be used.	Style issue, also a subset of Rule 162.	Disagree with AV.
164.	6.9 [XYZ] 6.15 [FIF] 6.16 [PIK]	**AV Rule 164 (MISRA Rule 38)** The right hand operand of a shift operator **shall** lie between zero and one less than the width in bits of the left-hand operand (inclusive).	X	
165.		**AV Rule 165 (MISRA Rule 39)** The unary minus operator **shall not** be applied to an unsigned expression.	Add to 6.6 [FLC] Numeric Conversion Errors	
166.	6.24 [SAM] 6.25 [KOA]	**AV Rule 166 (MISRA Rule 40)** The *sizeof* operator **will not** be used on expressions that contain side effects.	X	C/C++ specific. Remove from Part 1.
167.		**AV Rule 167 (MISRA Rule 41)** The implementation of integer division in the chosen compiler **shall** be determined, documented and taken into account.	Add to 6.53 [EWF] Undefined Behaviour	Make [FAB] 6.54, not [EWF]
168.		**AV Rule 168 (MISRA Rule 42, Revised)** The comma operator **shall not** be used.	Style issue	
169.		**AV Rule 169** Pointers to pointers **should** be avoided when possible.	Add to 6.50 [SKL] Provision of Inherently Unsafe Operations	Do not reference!
170.		**No rule listed**	No rule listed.	
171.		**AV Rule 170 (MISRA Rule 102, Revised)** More than 2 levels of pointer indirection **shall not** be used.		Style.
172.		**No rule listed**	No rule listed.	
173.	6.33 [DCM]	**AV Rule 173 (MISRA Rule 106, Revised)** The address of an object with automatic storage **shall not** be assigned to an object which persists after the object has ceased to exist.	X	
174.	6.13 [XYH]	**AV Rule 174 (MISRA Rule 107)** The null pointer **shall not** be de-referenced.	X	
175.		**AV Rule 175** A pointer **shall not** be compared to NULL or be assigned NULL; use plain 0 instead.	Add to 6.12 [RVG] Pointer Arithmetic, expand text of 6.12 to	No. Style Issue.

			include this.	
176.		**AV Rule 176** A typedef **will** be used to simplify program syntax when declaring function pointers.	Style issue	
177.		**AV Rule 177** User-defined conversion functions **should** be avoided.	Style issue	
178.		**AV Rule 178** Down casting (casting from base to derived class) **shall** only be allowed through one of the following mechanism: • Virtual functions that act like dynamic casts (most likely useful in relatively simple cases) • Use of the visitor (or similar) pattern (most likely useful in complicated cases)	Add to 6.41 [RIP] Inheritance	OBE. Advice, use virtual calls instead of downcasting. AI - Erhard for Part 1.
179.		**AV Rule 179** A pointer to a virtual base class **shall not** be converted to a pointer to a derived class.	Add to 6.41 [RIP] Inheritance	
180.		**AV Rule 180 (MISRA Rule 43)** Implicit conversions that may result in a loss of information **shall not** be used.	Add to 6.40 [SYM] Templates and Generics	Consider FLC. Using templates to avoid the conversion is C+ specific. We realize that FLC should also (as well as numeric conversions) consider general conversions.
181.		**AV Rule 181 (MISRA Rule 44)** Redundant explicit casts **will not** be used.	Style issue	
182.		**AV Rule 182 (MISRA Rule 45)** Type casting from any type to or from pointers **shall not** be used.	Add to 6.11 [HFC] Pointer Type Conversions	
183.	6.2 [IHN] 6.11 [HFC] 6.38 [AMV]	**AV Rule 183** Every possible measure **should** be taken to avoid type casting.	X	
184.	6.4 [PLF]	**AV Rule 184** Floating point numbers **shall not** be converted to integers unless such a conversion is a specified algorithmic requirement or is necessary for a hardware interface.	X	Disagree that 6.4 currently covers this issue. Add vulnerability material to 6.4?
185.		**AV Rule 185** C++ style casts (const_cast, reinterpret_cast, and static_cast) **shall** be used instead of the traditional C-style casts.	Add to 6.11 [HFC] Pointer Type Conversions	Style issue. Leave alone.
186.	6.26 [XYQ]	**AV Rule 186 (MISRA Rule 52)** There **shall** be no unreachable code.	X	
187.		**AV Rule 187 (MISRA Rule 53, Revised)** All non-null	Add to 6.26 Likely	OK.

		statements **shall** potentially have a side-effect.	Incorrect Expressions	
188.		**AV Rule 188 (MISRA Rule 55, Revised)** Labels **will not** be used, except in *switch* statements.	Style issue	Add to [EWD]
189.	6.31 [EWD]	**AV Rule 189 (MISRA Rule 56)** The *goto* statement **shall not** be used.	X	
190.	6.31 [EWD]	**AV Rule 190 (MISRA Rule 57)** The *continue* statement **shall not** be used.	X	
191.	6.31 [EWD]	**AV Rule 191 (MISRA Rule 58)** The *break* statement **shall not** be used (except to terminate the cases of a *switch* statement).	X	
192.	6.28 [EOJ]	**AV Rule 192 (MISRA Rule 60, Revised)** All *if*, *else if* constructs **will** contain either a final *else* clause or a comment indicating why a final *else* clause is not necessary.	X	
193.	6.27 [CLL]	**AV Rule 193 (MISRA Rule 61)** Every non-empty *case* clause in a *switch* statement **shall** be terminated with a *break* statement.	X	
194.	6.27 [CLL]	**AV Rule 194 (MISRA Rule 62, Revised)** All *switch* statements that do not intend to test for every enumeration value **shall** contain a final *default* clause.	X	
195.	6.27 [CLL]	**AV Rule 195 (MISRA Rule 63)** A *switch* expression **will not** represent a Boolean value.	X	
196.	6.27 [CLL]	**AV Rule 196 (MISRA Rule 64, Revised)** Every *switch* statement **will** have at least two *cases* and a potential *default.*	X	
197.	6.4 [PLF]	**AV Rule 197 (MISRA Rule 65)** Floating point variables **shall not** be used as loop counters.	X	

198.		**AV Rule 198** The initialization expression in a *for* loop **will** perform no actions other than to initialize the value of a single *for* loop parameter. Note that the initialization expression may invoke an accessor that returns an initial element in a sequence:
		 for (Iter_type p = c.begin() ; p != c.end() ; ++p) // Good
		 {
			Add to 6.29 [TEX] Loop Control Variables	The added text should say "In languages that permit complex expressions in the definition of the loop control variable, ..."

		... }		
199.		**AV Rule 199** The increment expression in a *for* loop **will** perform no action other than to change a single loop parameter to the next value for the loop.	Add to 6.29[TEX] Loop Control Variables	The added text should say "In languages that do not prevent the update of the loop control variable, ..."
200.		**AV Rule 200** Null initialize or increment expressions in *for* loops **will not** be used; a *while* loop **will** be used instead.	Style issue	
201.	6.29 [TEX]	**AV Rule 201 (MISRA Rule 67, Revised)** Numeric variables being used within a *for* loop for iteration counting **shall not** be modified in the body of the loop.	X	
202.	6.4 [PLF]	**AV Rule 202 (MISRA Rule 50)** Floating point variables **shall not** be tested for exact equality or inequality.	X	
203.		**AV Rule 203 (MISRA Rule 51, Revised)** Evaluation of expressions **shall not** lead to overflow/underflow (unless required algorithmically and then should be heavily documented).	Add to 6.15 [FIF] Arithmetic Wrap-around Error	
204.	6.23 [JCW] 6.24 [SAM]	**AV Rule 204** A single operation with side-effects **shall** only be used in the following contexts: 1. by itself 2. the right-hand side of an assignment 3. a condition 4. the only argument expression with a side-effect in a function call 5. condition of a loop 6. switch condition 7. single part of a chained operation.	X	
205.		**AV Rule 205** The *volatile* keyword **shall not** be used unless directly interfacing with hardware.	Add to 6.19 [WXQ] Dead Store	OK for [WXQ] as a mitigation. Elsewhere? Don't put reference to JSF in.
206.	6.39 [XYL]	**AV Rule 206 (MISRA Rule 118, Revised)** Allocation/deallocation from/to the free store (heap) **shall not** occur after initialization. Note that the "placement" *operator new()*, although not	X	

		technically dynamic memory, may only be used in low-level memory management routines. See AV Rule 70.1 for object lifetime issues associated with placement *operator new()*.		
207.		**AV Rule 207** Unencapsulated global data **will** be avoided.	Add to 6.20 [YOW] Identifier Name Reuse	Really a namespace issue? Not YOW.
208.	6.36 [OYB] 6.47 [HJW]	**AV Rule 208** C++ exceptions **shall not** be used (i.e. *throw*, *catch* and *try* shall not be used.)	X	Huh? Get rid of references.
209.		**AV Rule 209 (MISRA Rule 13, Revised)** The basic types of *int, short, long, float* and *double* **shall not** be used, but specific-length equivalents should be *typedef'*d accordingly for each compiler, and these type names used in the code.	Style issue	Much more than style. This avoids compiler-specific default behaviours (such as reliance on sizeof(int)). Check to see if we have a comparable rule in 6.2
210.		**AV Rule 210** Algorithms **shall not** make assumptions concerning how data is represented in memory (e.g. big endian vs. little endian, base class subobject ordering in derived classes, nonstatic data member ordering across access specifiers, etc.)	Add to 6.3 [STR] Bit Representations, 6.4 [PLF] Floating-point Arithmetic	We need more – heavily document assumptions, and provide error detection and raising if assumptions are violated.
211.		**AV Rule 211** Algorithms **shall not** assume that *shorts, ints, longs*, *floats*, *doubles* or *long doubles* begin at particular addresses.	Add to 6.33 [DCM] Dangling References to Stack Frames	This is an alignment issue. Not [DCM]. Data layout - may be vulnerability. May be involved with unchecked conversion union, etc.
212.		**AV Rule 212** Underflow or overflow functioning **shall not** be depended on in any special way.	Add to 6.6 [FLC] Numeric Conversion Errors, 6.53 [EWF] Undefined Behaviour	OK
213.	6.23 [SAM] 6.24 [JCW]	**AV Rule 213 (MISRA Rule 47, Revised)** No dependence **shall** be placed on C++'s operator precedence rules, below arithmetic operators, in expressions.	X	Not SAM or common programmer errors or KOA.
214.		**AV Rule 214** Assuming that non-local static objects, in separate translation units, are initialized in a special order **shall not** be done.	Add to 6.23 [JCW] Operator Precedence/Order of Evaluation	Not JCW, potential new vulnerability around elaboration. Check LAV for possible coverage.
215.	6.12 [RVG]	**AV Rule 215 (MISRA Rule 101)** Pointer arithmetic **will not** be used.	X	

216.		**AV Rule 216** Programmers **should not** attempt to prematurely optimize code.	Performance issue	
217.		**AV Rule 217** Compile-time and link-time errors **should** be preferred over run-time errors.	Style issue	
218.		**AV Rule 218** Compiler warning levels **will** be set in compliance with project policies.	Style issue	
219.		**AV Rule 219** All tests applied to a base class interface **shall be** applied to all derived class interfaces as well. If the derived class poses stronger postconditions/invariants, then the new postconditions /invariants shall be substituted in the derived class tests.	Add to 6.42 Inheritance	OK
220.		**AV Rule 220** Structural coverage algorithms **shall** be applied against *flattened* classes.	Add to 6.42 [RIP] Inheritance	No.
221.		**AV Rule 221** Structural coverage of a class within an inheritance hierarchy containing virtual functions **shall** include testing every possible resolution for each set of identical polymorphic references.	Add to 6.42 [RIP] Inheritance	No.