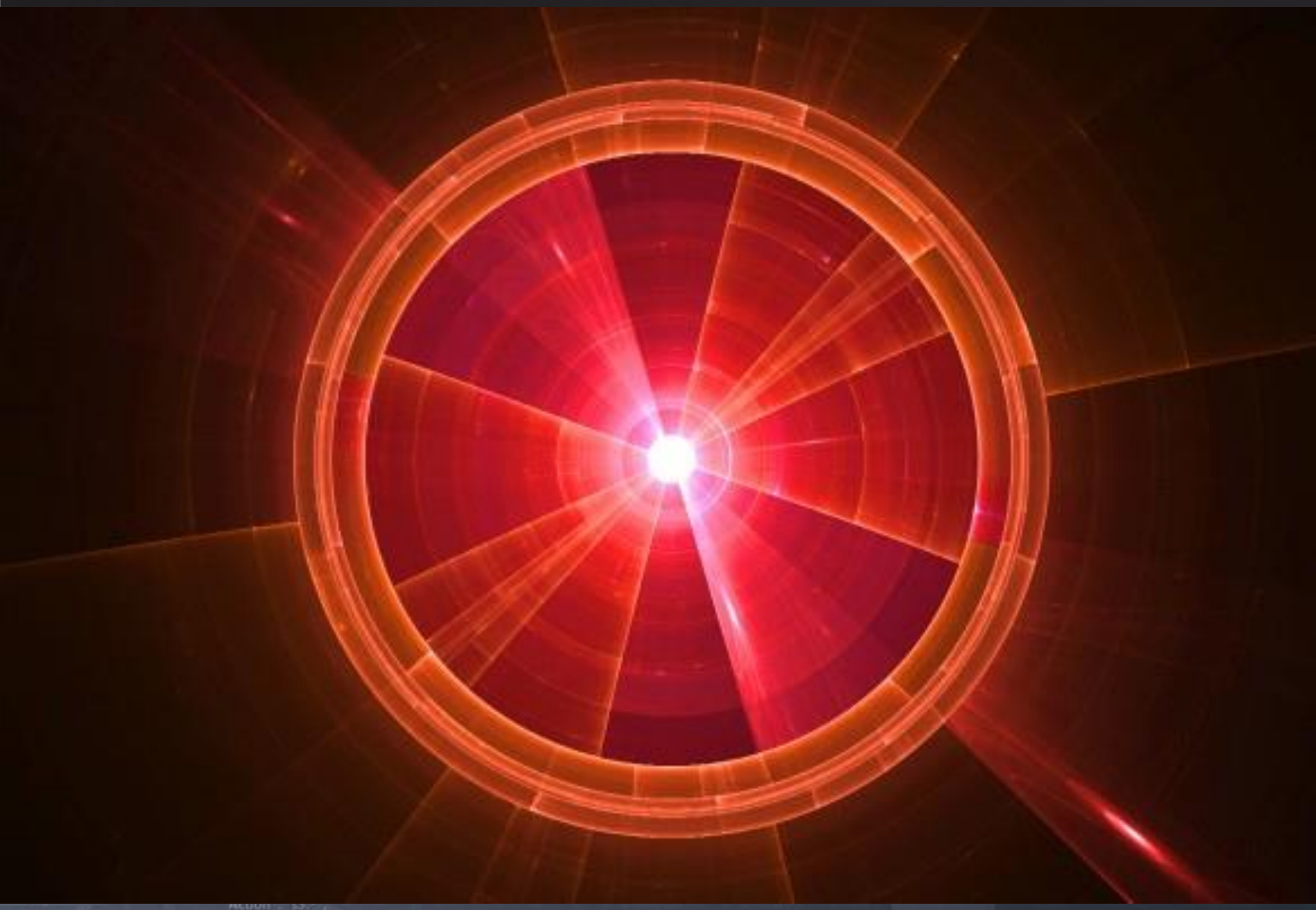
CODESECURE

JOINT STRIKE FORCE RULES | MAPPED TO CODESONAR® 8.0



TRUSTED LEADERS OF SOFTWARE ASSURANCE AND ADVANCED CYBER-SECURITY SOLUTIONS

WWW.CODESECURE.COM

INTRODUCTION

Joint Strike Fighter Air Vehicle C++ (JSF AV C++) is a coding standard developed by Lockheed Martin and is designed for the aerospace and defense industry.

CodeSecure is a leading global provider of application testing (AST) solutions used by the world's most security conscious organizations to detect, measure, analyze and resolve vulnerabilities for software they develop or use. The company is also a trusted cybersecurity and artificial intelligence research partner for the nation's civil, defense, and intelligence agencies.

JSF AV C++ CODING STANDARD CLOSE MAPPING (CODESONAR V8.0)

The following table shows the CodeSonar warning classes that are closely mapped to JSF AV C++ categories.

Rule	Rule Name	Category	Supported
JSF++:1	Any one function (or method) will contain no more than 200 logical source lines of code (L-SLOCs).	Will	Yes
JSF++:2	There shall not be any self-modifying code.	Shall	No
JSF++:3	All functions shall have a cyclomatic complexity number of 20 or less.	Shall	Yes
JSF++:4	To break a "should" rule, specific approval must be received by the developer.	Shall	No
JSF++:5	To break a "will" or a "shall" rule, specific approvals must be received by the developer.	Shall	No
JSF++: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.	Shall	No
JSF++:7	Approval will not be required for a deviation from a "shall" or "will" rule that complies with an exception specified by that rule.	Will	No
JSF++:8	All code shall conform to ISO/IEC 14882:2002(E) standard C++.	Shall	No
JSF++:9	Only those characters specified in the C++ basic source character set will be used.', long_html_name='Only those characters specified in the C++ basic source character set will be used. This set includes 96 characters: the space character, the control characters representing horizontal tab, vertical tab, form feed, and newline, and the following 91 graphical characters: a b c d e f g h i j k l m n o p q r s t u v w x y z A B C D E F G H I J K L M N O P Q R S T U V W X Y Z 0 1 2 3 4 5 6 7 8 9 _ { } [ ] # ( ) < > % : ; . ? * + - / ^ & | ~ ! = , \ " '	Will	No
JSF++:10	Values of character types will be restricted to a defined and documented subset of ISO 10646-1.	Will	No
JSF++:11	Trigraphs will not be used.	Will	Yes
JSF++:12	Certain digraphs will not be used.	Will	No
JSF++:13	Multi-byte characters and wide string literals will not be used.	Will	Yes
JSF++:14	Literal suffixes shall use uppercase rather than lowercase letters.	Shall	Yes
JSF++:15	Provision shall be made for run-time checking (defensive programming).	Shall	No
JSF++:16	Only DO-178B level A certifiable or SEAL 1 C/C++ libraries shall be used with safety-critical (i.e. SEAL 1) code.	Shall	No
JSF++:17	The error indicator errno shall not be used.	Shall	No
JSF++:18	The macro offsetof, in library , shall not be used.	Shall	Yes
JSF++:19	and the setlocale function shall not be used.	Shall	No
JSF++:20	The setjmp macro and the longjmp function shall not be used.	Shall	Yes
JSF++:21	The signal handling facilities of shall not be used.	Shall	Yes
JSF++:22	The input/output library shall not be used.	Shall	Yes
JSF++:23	The library functions atof, atoi and atol from library shall not be used.	Shall	Yes
JSF++:24	The library functions abort, exit, getenv and system from library shall not be used.	Shall	Yes
JSF++:25	The time handling functions of library shall not be used.	Shall	Yes
JSF++:26	Only the following pre-processor directives shall be used: #ifndef, #define, #endif, #include.	Shall	No
JSF++: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.	Will	No
JSF++: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.	Will	No
JSF++:29	The #define pre-processor directive shall not be used to create inline macros. Inline functions shall be used instead.	Shall	Yes

JSF++: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.	Shall	No
JSF++:31	The #define pre-processor directive will only be used as part of the technique to prevent multiple inclusions of the same header file.	Will	No
JSF++:32	The #include pre-processor directive will only be used to include header (*.h) files.	Will	No
JSF++:33	The #include directive shall use the notation to include header files.	Shall	No
JSF++:34	Header files should contain logically related declarations only.	Should	No
JSF++:35	A header file will contain a mechanism that prevents multiple inclusions of itself.	Will	No
JSF++:36	Compilation dependencies should be minimized when possible.	Should	No
JSF++: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 fileâ€”not the .h file.	Should	No
JSF++:38	Declarations of classes that are only accessed via pointers (*) or references (&) should be supplied by forward headers that contain only forward declarations.	Should	No
JSF++:39	Header files (*.h) will not contain non-const variable definitions or function definitions.	Will	Yes
JSF++:40	Every implementation file shall include the header files that uniquely define the inline functions, types, and templates used.	Shall	Yes
JSF++:41	Source lines will be kept to a length of 120 characters or less.	Will	No
JSF++:42	Each expression-statement will be on a separate line.	Will	Yes
JSF++:43	Tabs should be avoided.	Should	No
JSF++:44	All indentations will be at least two spaces and be consistent within the same source file.	Will	No
JSF++:45	All words in an identifier will be separated by the '_' character.	Will	No
JSF++:46	User-specified identifiers (internal and external) will not rely on significance of more than 64 characters.	Will	Yes
JSF++:47	Identifiers will not begin with the underscore character '_'.	Will	Yes
JSF++:48	Identifiers will not be typographically ambiguous.	Will	Yes
JSF++:49	All acronyms in an identifier will be composed of uppercase letters.	Will	No
JSF++: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.	Will	Yes
JSF++:51	All letters contained in function and variable names will be composed entirely of lowercase letters.	Will	Yes
JSF++:52	Identifiers for constant and enumerator values shall be lowercase.	Shall	Yes
JSF++:53	Header files will always have a file name extension of ".h".	Will	No
JSF++:53.1	The following character sequences shall not appear in header file names: ', \, /*, //, or ".	Shall	Yes
JSF++:54	Implementation files will always have a file name extension of ".cpp".	Will	No
JSF++:55	The name of a header file should reflect the logical entity for which it provides declarations.	Should	No
JSF++:56	The name of an implementation file should reflect the logical entity for which it provides definitions and have a ".cpp" extension.	Should	No
JSF++: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).	Will	No
JSF++: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).	Will	No
JSF++:59	The statements forming the body of an if, else if, else, while, do...while or for statement shall always be enclosed in braces, even if the braces form an empty block.	Shall	Yes
JSF++:60	Braces ("{}") which enclose a block will be placed in the same column, on separate lines directly before and after the block.	Will	No

JSF++:61	Braces ("{}") which enclose a block will have nothing else on the line except comments (if necessary).	Will	No
JSF++:62	The dereference operator '*' and the address-of operator '&' will be directly connected with the type-specifier.	Will	No
JSF++:63	Spaces will not be used around '.' or '->', nor between unary operators and operands.	Will	No
JSF++:64	A class interface should be complete and minimal.	Should	No
JSF++:65	A structure should be used to model an entity that does not require an invariant.	Should	No
JSF++:66	A class should be used to model an entity that maintains an invariant.	Should	No
JSF++:67	Public and protected data should only be used in structsâ€”not classes.	Should	No
JSF++:68	Unneeded implicitly generated member functions shall be explicitly disallowed.	Shall	No
JSF++:69	A member function that does not affect the state of an object (its instance variables) will be declared const.	Will	No
JSF++:70	A class will have friends only when a function or object requires access to the private elements of the class, but is unable to be a member of the class for logical or efficiency reasons.	Will	No
JSF++:70.1	An object shall not be improperly used before its lifetime begins or after its lifetime ends.	Shall	Yes
JSF++: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.	Shall	Yes
JSF++:71.1	A class's virtual functions shall not be invoked from its destructor or any of its constructors.	Shall	Yes
JSF++:72	The invariant for a class should be part of class member pre- and/or post-conditions as specified.	Should	No
JSF++:73	Unnecessary default constructors shall not be defined.	Shall	No
JSF++:74	Initialization of nonstatic class members will be performed through the member initialization list rather than through assignment in the body of a constructor.	Will	No
JSF++:75	Members of the initialization list shall be listed in the order in which they are declared in the class.	Shall	Yes
JSF++:76	A copy constructor and an assignment operator shall be declared for classes that contain pointers to data items or nontrivial destructors.	Shall	No
JSF++: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).	Shall	No
JSF++:77.1	The definition of a member function shall not contain default arguments that produce a signature identical to that of the implicitly-declared copy constructor for the corresponding class/structure.	Shall	No
JSF++:78	All base classes with a virtual function shall define a virtual destructor.	Shall	Yes
JSF++:79	All resources acquired by a class shall be released by the class's destructor.	Shall	Yes
JSF++:80	The default copy and assignment operators will be used for classes when those operators offer reasonable semantics.	Will	No
JSF++:81	The assignment operator shall handle self-assignment correctly.	Shall	Yes
JSF++:82	An assignment operator shall return a reference to *this.	Shall	No
JSF++: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).	Shall	No
JSF++:84	Operator overloading will be used sparingly and in a conventional manner.	Will	No
JSF++:85	When two operators are opposites (such as == and !=), both will be defined and one will be defined in terms of the other.	Will	No
JSF++:86	Concrete types should be used to represent simple independent concepts.	Should	No
JSF++:87	Hierarchies should be based on abstract classes.	Should	No
JSF++:88	Multiple inheritance shall only be allowed in the following restricted form: n interfaces plus m private implementations, plus at most one protected implementation.	Shall	No
JSF++:88.1	A stateful virtual base shall be explicitly declared in each derived class that accesses it.	Shall	No
JSF++:89	A base class shall not be both virtual and non-virtual in the same hierarchy.	Shall	Yes
JSF++:90	Heavily used interfaces should be minimal, general and abstract.	Should	No

JSF++:91	Public inheritance will be used to implement "is-a" relationships.	Will	No
JSF++:92	Subclass methods must expect less and deliver more than the base class methods they override	Will	No
JSF++:93	"has-a" or "is-implemented-in-terms-of" relationships will be modeled through membership or non-public inheritance.	Will	No
JSF++:94	An inherited nonvirtual function shall not be redefined in a derived class.	Shall	No
JSF++:95	An inherited default parameter shall never be redefined.	Shall	Yes
JSF++:96	Arrays shall not be treated polymorphically.	Shall	Yes
JSF++:97	Arrays shall not be used in interfaces. Instead, the Array class should be used.	Shall	No
JSF++:97.1	Neither operand of an equality operator (== or !=) shall be a pointer to a virtual member function.	Shall	No
JSF++:98	Every nonlocal name, except main(), should be placed in some namespace.	Should	No
JSF++:99	Namespaces will not be nested more than two levels deep.	Will	No
JSF++:100	Elements from a namespace should be selected as specified.	Should	No
JSF++:101	Templates shall be reviewed as specified.	Shall	No
JSF++:102	Template tests shall be created to cover all actual template instantiations.	Shall	No
JSF++:103	Constraint checks should be applied to template arguments.	Should	No
JSF++:104	A template specialization shall be declared before its use.	Shall	No
JSF++:105	A template definition's dependence on its instantiation contexts should be minimized.	Should	No
JSF++:106	Specializations for pointer types should be made where appropriate.	Should	No
JSF++:107	Functions shall always be declared at file scope.	Shall	Yes
JSF++:108	Functions with variable numbers of arguments shall not be used.	Shall	Yes
JSF++:109	A function definition should not be placed in a class specification unless the function is intended to be inlined.	Should	No
JSF++:110	Functions with more than 7 arguments will not be used.	Will	Yes
JSF++:111	A function shall not return a pointer or reference to a non-static local object.	Shall	Yes
JSF++:112	Function return values should not obscure resource ownership.	Should	No
JSF++:113	Functions will have a single exit point.	Will	Yes
JSF++:114	All exit points of value-returning functions shall be through return statements.	Shall	Yes
JSF++:115	If a function returns error information, then that error information will be tested.	Will	Yes
JSF++: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.	Should	No
JSF++:117	Arguments should be passed by reference if NULL values are not possible	Should	No
JSF++:117.1	An object should be passed as const T& if the function should not change the value of the object.	Should	Yes
JSF++:117.2	An object should be passed as T& if the function may change the value of the object.	Should	No
JSF++:118	Arguments should be passed via pointers if NULL values are possible.	Should	No
JSF++:118.1	An object should be passed as const T* if its value should not be modified.	Should	No
JSF++:118.2	An object should be passed as T* if its value may be modified.	Should	No
JSF++:119	Functions shall not call themselves, either directly or indirectly (i.e. recursion shall not be allowed).	Shall	Yes
JSF++: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.	Should	No
JSF++:121	Only functions with 1 or 2 statements should be considered candidates for inline functions.	Should	No
JSF++:122	Trivial accessor and mutator functions should be inlined.	Should	No
JSF++:123	The number of accessor and mutator functions should be minimized.	Should	No
JSF++:124	Trivial forwarding functions should be inlined.	Should	No
JSF++:125	Unnecessary temporary objects should be avoided.	Should	No

JSF++:126	Only valid C++ style comments (//) shall be used.	Shall	Yes
JSF++:127	Code that is not used (commented out) shall be deleted.	Shall	Yes
JSF++:128	Comments that document actions or sources (e.g. tables, figures, paragraphs, etc.) outside of the file being documented will not be allowed.	Will	No
JSF++:129	Comments in header files should describe the externally visible behavior of the functions or classes being documented.	Should	No
JSF++: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.	Should	No
JSF++:131	One should avoid stating in comments what is better stated in code (i.e. do not simply repeat what is in the code).	Should	No
JSF++:132	Each variable declaration, typedef, enumeration value, and structure member will be commented.	Will	No
JSF++: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).	Will	No
JSF++:134	Assumptions (limitations) made by functions should be documented in the function's preamble.	Should	No
JSF++:135	Identifiers in an inner scope shall not use the same name as an identifier in an outer scope, and therefore hide that identifier.	Shall	Yes
JSF++:136	Declarations should be at the smallest feasible scope.	Should	Yes
JSF++:137	All declarations at file scope should be static where possible.	Should	Yes
JSF++:138	Identifiers shall not simultaneously have both internal and external linkage in the same translation unit.	Shall	No
JSF++:139	External objects will not be declared in more than one file.	Will	Yes
JSF++:140	The register storage class specifier shall not be used.	Shall	No
JSF++:141	A class, structure, or enumeration will not be declared in the definition of its type.	Will	No
JSF++:142	All variables shall be initialized before use.	Shall	Yes
JSF++:143	Variables will not be introduced until they can be initialized with meaningful values.	Will	No
JSF++:144	Braces shall be used to indicate and match the structure in the non-zero initialization of arrays and structures.	Shall	Yes
JSF++:145	In an enumerator list, the '=' construct shall not be used to explicitly initialize members other than the first, unless all items are explicitly initialized.	Shall	Yes
JSF++:146	Floating point implementations shall comply with a defined floating point standard. The standard that will be used is the ANSI/IEEE Std 754.	Shall	No
JSF++:147	The underlying bit representations of floating point numbers shall not be used in any way by the programmer.	Shall	Yes
JSF++:148	Enumeration types shall be used instead of integer types (and constants) to select from a limited series of choices.	Shall	No
JSF++:149	Octal constants (other than zero) shall not be used.	Shall	Yes
JSF++:150	Hexadecimal constants will be represented using all uppercase letters.	Will	No
JSF++:151	Numeric values in code will not be used; symbolic values will be used instead.	Will	No
JSF++:151.1	A string literal shall not be modified.	Shall	Yes
JSF++:152	Multiple variable declarations shall not be allowed on the same line.	Shall	Yes
JSF++:153	Unions shall not be used.	Shall	Yes
JSF++:154	Bit-fields shall have explicitly unsigned integral or enumeration types only.	Shall	Yes
JSF++:155	Bit-fields will not be used to pack data into a word for the sole purpose of saving space.	Will	No
JSF++:156	All the members of a structure (or class) shall be named and shall only be accessed via their names.	Shall	No
JSF++:157	The right hand operand of a && or || operator shall not contain side effects.	Shall	Yes

JSF++:158	The operands of a logical && or || shall be parenthesized if the operands contain binary operators.	Shall	Yes
JSF++:159	Operators ||, &&, and unary & shall not be overloaded.	Shall	Yes
JSF++:160	An assignment expression shall be used only as the expression in an expression statement.	Shall	Yes
JSF++:162	Signed and unsigned values shall not be mixed in arithmetic or comparison operations.	Shall	Yes
JSF++:163	Unsigned arithmetic shall not be used.	Shall	No
JSF++:164	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).	Shall	Yes
JSF++:164.1	The left-hand operand of a right-shift operator shall not have a negative value.	Shall	No
JSF++:165	The unary minus operator shall not be applied to an unsigned expression.	Shall	Yes
JSF++:166	The sizeof operator will not be used on expressions that contain side effects.	Will	Yes
JSF++:167	The implementation of integer division in the chosen compiler shall be determined, documented and taken into account.	Shall	No
JSF++:168	The comma operator shall not be used.	Shall	Yes
JSF++:169	Pointers to pointers should be avoided when possible.	Should	Yes
JSF++:170	More than 2 levels of pointer indirection shall not be used.	Shall	Yes
JSF++:171	Relational operators shall not be applied to pointer types except where both operands are of the same type and point to the same entity.	Shall	Yes
JSF++:173	The address of an object with automatic storage shall not be assigned to an object which persists after the object has ceased to exist.	Shall	Yes
JSF++:174	The null pointer shall not be de-referenced.	Shall	Yes
JSF++:175	A pointer shall not be compared to NULL or be assigned NULL; use plain 0 instead.	Shall	No
JSF++:176	A typedef will be used to simplify program syntax when declaring function pointers.	Will	No
JSF++:177	User-defined conversion functions should be avoided.	Should	No
JSF++:178	Down casting (casting from base to derived class) shall only be allowed through certain mechanisms.	Shall	No
JSF++:179	A pointer to a virtual base class shall not be converted to a pointer to a derived class.	Shall	No
JSF++:180	Implicit conversions that may result in a loss of information shall not be used.	Shall	Yes
JSF++:181	Redundant explicit casts will not be used.	Will	No
JSF++:182	Type casting from any type to or from pointers shall not be used.	Shall	Yes
JSF++:183	Every possible measure should be taken to avoid type casting.	Should	Yes
JSF++: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.	Shall	Yes
JSF++:185	C++ style casts (const_cast, reinterpret_cast, and static_cast) shall be used instead of the traditional C-style casts.	Shall	Yes
JSF++:186	There shall be no unreachable code.	Shall	Yes
JSF++:187	All non-null statements shall potentially have a side-effect.	Shall	Yes
JSF++:188	Labels will not be used, except in switch statements.	Will	No
JSF++:189	The goto statement shall not be used.	Shall	Yes
JSF++:190	The continue statement shall not be used.	Shall	Yes
JSF++:191	The break statement shall not be used (except to terminate the cases of a switch statement).	Shall	Yes
JSF++:192	All if, else if constructs will contain either a final else clause or a comment indicating why a final else clause is not necessary.	Will	Yes
JSF++:193	Every non-empty case clause in a switch statement shall be terminated with a break statement.	Shall	Yes
JSF++:194	All switch statements that do not intend to test for every enumeration value shall contain a final default clause.	Shall	Yes
JSF++:195	A switch expression will not represent a Boolean value.	Will	Yes

JSF++:196	Every switch statement will have at least two cases and a potential default.	Will	Yes
JSF++:197	Floating point variables shall not be used as loop counters.	Shall	Yes
JSF++:198	The initialization expression in a for loop will perform no actions other than to initialize the value of a single for loop parameter.	Will	Yes
JSF++: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.	Will	Yes
JSF++:200	Null initialize or increment expressions in for loops will not be used; a while loop will be used instead.	Will	Yes
JSF++:201	Numeric variables being used within a for loop for iteration counting shall not be modified in the body of the loop.	Shall	Yes
JSF++:202	Floating point variables shall not be tested for exact equality or inequality.	Shall	Yes
JSF++:203	Evaluation of expressions shall not lead to overflow/underflow (unless required algorithmically and then should be heavily documented).	Shall	Yes
JSF++:204	A single operation with side-effects shall only be used in certain contexts.	Shall	No
JSF++:204.1	The value of an expression shall be the same under any order of evaluation that the standard permits.	Shall	Yes
JSF++:205	The volatile keyword shall not be used unless directly interfacing with hardware.	Shall	No
JSF++:206	Allocation/deallocation from/to the free store (heap) shall not occur after initialization.	Shall	Yes
JSF++:207	Unencapsulated global data will be avoided.	Will	No
JSF++:208	C++ exceptions shall not be used (i.e. throw, catch and try shall not be used.)	Shall	Yes
JSF++:209	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.	Shall	Yes
JSF++: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.)	Shall	Yes
JSF++:210.1	Algorithms shall not make assumptions concerning the order of allocation of nonstatic data members separated by an access specifier.	Shall	Yes
JSF++:211	Algorithms shall not assume that shorts, ints, longs, floats, doubles or long doubles begin at particular addresses.	Shall	Yes
JSF++:212	Underflow or overflow functioning shall not be depended on in any special way.	Shall	Yes
JSF++:213	No dependence shall be placed on C++'s operator precedence rules, below arithmetic operators, in expressions.	Shall	Yes
JSF++:214	Assuming that non-local static objects, in separate translation units, are initialized in a special order shall not be done.	Shall	Yes
JSF++:215	Pointer arithmetic will not be used.	Will	Yes
JSF++:216	Programmers should not attempt to prematurely optimize code.	Should	No
JSF++:217	Compile-time and link-time errors should be preferred over run-time errors.	Should	No
JSF++:218	Compiler warning levels will be set in compliance with project policies.	Will	Yes
JSF++: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.	Shall	No
JSF++:220	Structural coverage algorithms shall be applied against flattened classes.	Shall	No
JSF++: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.	Shall	No

JSF AV C++ CODING STANDARD BROAD MAPPING (CODESONAR V8.0)

The following table shows the CodeSonar warning classes that are broadly mapped to JSF AV C++ categories.

Rule	Rule Name	Category	Supported
JSF++:1	Any one function (or method) will contain no more than 200 logical source lines of code (L-SLOCs).	Will	Yes
JSF++:2	There shall not be any self-modifying code.	Shall	No
JSF++:3	All functions shall have a cyclomatic complexity number of 20 or less.	Shall	Yes
JSF++:4	To break a "should" rule, specific approval must be received by the developer.	Shall	No
JSF++:5	To break a "will" or a "shall" rule, specific approvals must be received by the developer.	Shall	No
JSF++: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.	Shall	No
JSF++:7	Approval will not be required for a deviation from a "shall" or "will" rule that complies with an exception specified by that rule.	Will	No
JSF++:8	All code shall conform to ISO/IEC 14882:2002(E) standard C++.	Shall	No
JSF++:9	Only those characters specified in the C++ basic source character set will be used.', long_html_name='Only those characters specified in the C++ basic source character set will be used. This set includes 96 characters: the space character, the control characters representing horizontal tab, vertical tab, form feed, and newline, and the following 91 graphical characters: a b c d e f g h i j k l m n o p q r s t u v w x y z A B C D E F G H I J K L M N O P Q R S T U V W X Y Z 0 1 2 3 4 5 6 7 8 9 _ { } [ ] # ( ) < > % : ; . ? * + - / ^ & | ~ ! = , \ " '	Will	No
JSF++:10	Values of character types will be restricted to a defined and documented subset of ISO 10646-1.	Will	No
JSF++:11	Trigraphs will not be used.	Will	Yes
JSF++:12	Certain digraphs will not be used.	Will	No
JSF++:13	Multi-byte characters and wide string literals will not be used.	Will	Yes
JSF++:14	Literal suffixes shall use uppercase rather than lowercase letters.	Shall	Yes
JSF++:15	Provision shall be made for run-time checking (defensive programming).	Shall	No
JSF++:16	Only DO-178B level A certifiable or SEAL 1 C/C++ libraries shall be used with safety-critical (i.e. SEAL 1) code.	Shall	No
JSF++:17	The error indicator errno shall not be used.	Shall	No
JSF++:18	The macro offsetof, in library , shall not be used.	Shall	Yes
JSF++:19	and the setlocale function shall not be used.	Shall	No
JSF++:20	The setjmp macro and the longjmp function shall not be used.	Shall	Yes
JSF++:21	The signal handling facilities of shall not be used.	Shall	Yes
JSF++:22	The input/output library shall not be used.	Shall	Yes
JSF++:23	The library functions atof, atoi and atol from library shall not be used.	Shall	Yes
JSF++:24	The library functions abort, exit, getenv and system from library shall not be used.	Shall	Yes
JSF++:25	The time handling functions of library shall not be used.	Shall	Yes
JSF++:26	Only the following pre-processor directives shall be used: #ifndef, #define, #endif, #include.	Shall	No
JSF++: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.	Will	No
JSF++: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.	Will	No
JSF++:29	The #define pre-processor directive shall not be used to create inline macros. Inline functions shall be used instead.	Shall	Yes

JSF++: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.	Shall	No
JSF++:31	The #define pre-processor directive will only be used as part of the technique to prevent multiple inclusions of the same header file.	Will	No
JSF++:32	The #include pre-processor directive will only be used to include header (*.h) files.	Will	No
JSF++:33	The #include directive shall use the notation to include header files.	Shall	No
JSF++:34	Header files should contain logically related declarations only.	Should	No
JSF++:35	A header file will contain a mechanism that prevents multiple inclusions of itself.	Will	No
JSF++:36	Compilation dependencies should be minimized when possible.	Should	No
JSF++: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 fileâ€”not the .h file.	Should	No
JSF++:38	Declarations of classes that are only accessed via pointers (*) or references (&) should be supplied by forward headers that contain only forward declarations.	Should	No
JSF++:39	Header files (*.h) will not contain non-const variable definitions or function definitions.	Will	Yes
JSF++:40	Every implementation file shall include the header files that uniquely define the inline functions, types, and templates used.	Shall	Yes
JSF++:41	Source lines will be kept to a length of 120 characters or less.	Will	No
JSF++:42	Each expression-statement will be on a separate line.	Will	Yes
JSF++:43	Tabs should be avoided.	Should	No
JSF++:44	All indentations will be at least two spaces and be consistent within the same source file.	Will	No
JSF++:45	All words in an identifier will be separated by the '_' character.	Will	No
JSF++:46	User-specified identifiers (internal and external) will not rely on significance of more than 64 characters.	Will	Yes
JSF++:47	Identifiers will not begin with the underscore character '_'.	Will	Yes
JSF++:48	Identifiers will not be typographically ambiguous.	Will	Yes
JSF++:49	All acronyms in an identifier will be composed of uppercase letters.	Will	No
JSF++: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.	Will	Yes
JSF++:51	All letters contained in function and variable names will be composed entirely of lowercase letters.	Will	Yes
JSF++:52	Identifiers for constant and enumerator values shall be lowercase.	Shall	Yes
JSF++:53	Header files will always have a file name extension of ".h".	Will	No
JSF++:53.1	The following character sequences shall not appear in header file names: ', \, /*, //, or ".	Shall	Yes
JSF++:54	Implementation files will always have a file name extension of ".cpp".	Will	No
JSF++:55	The name of a header file should reflect the logical entity for which it provides declarations.	Should	No
JSF++:56	The name of an implementation file should reflect the logical entity for which it provides definitions and have a ".cpp" extension.	Should	No
JSF++: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).	Will	No
JSF++: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).	Will	No
JSF++:59	The statements forming the body of an if, else if, else, while, do...while or for statement shall always be enclosed in braces, even if the braces form an empty block.	Shall	Yes
JSF++:60	Braces ("{}") which enclose a block will be placed in the same column, on separate lines directly before and after the block.	Will	No

JSF++:61	Braces ("{}") which enclose a block will have nothing else on the line except comments (if necessary).	Will	No
JSF++:62	The dereference operator '*' and the address-of operator '&' will be directly connected with the type-specifier.	Will	No
JSF++:63	Spaces will not be used around '.' or '->', nor between unary operators and operands.	Will	No
JSF++:64	A class interface should be complete and minimal.	Should	No
JSF++:65	A structure should be used to model an entity that does not require an invariant.	Should	No
JSF++:66	A class should be used to model an entity that maintains an invariant.	Should	No
JSF++:67	Public and protected data should only be used in structsâ€”not classes.	Should	No
JSF++:68	Unneeded implicitly generated member functions shall be explicitly disallowed.	Shall	No
JSF++:69	A member function that does not affect the state of an object (its instance variables) will be declared const.	Will	No
JSF++:70	A class will have friends only when a function or object requires access to the private elements of the class, but is unable to be a member of the class for logical or efficiency reasons.	Will	No
JSF++:70.1	An object shall not be improperly used before its lifetime begins or after its lifetime ends.	Shall	Yes
JSF++: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.	Shall	Yes
JSF++:71.1	A class's virtual functions shall not be invoked from its destructor or any of its constructors.	Shall	Yes
JSF++:72	The invariant for a class should be part of class member pre- and/or post-conditions as specified.	Should	No
JSF++:73	Unnecessary default constructors shall not be defined.	Shall	No
JSF++:74	Initialization of nonstatic class members will be performed through the member initialization list rather than through assignment in the body of a constructor.	Will	No
JSF++:75	Members of the initialization list shall be listed in the order in which they are declared in the class.	Shall	Yes
JSF++:76	A copy constructor and an assignment operator shall be declared for classes that contain pointers to data items or nontrivial destructors.	Shall	No
JSF++: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).	Shall	No
JSF++:77.1	The definition of a member function shall not contain default arguments that produce a signature identical to that of the implicitly-declared copy constructor for the corresponding class/structure.	Shall	No
JSF++:78	All base classes with a virtual function shall define a virtual destructor.	Shall	Yes
JSF++:79	All resources acquired by a class shall be released by the class's destructor.	Shall	Yes
JSF++:80	The default copy and assignment operators will be used for classes when those operators offer reasonable semantics.	Will	No
JSF++:81	The assignment operator shall handle self-assignment correctly.	Shall	Yes
JSF++:82	An assignment operator shall return a reference to *this.	Shall	No
JSF++: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).	Shall	No
JSF++:84	Operator overloading will be used sparingly and in a conventional manner.	Will	No
JSF++:85	When two operators are opposites (such as == and !=), both will be defined and one will be defined in terms of the other.	Will	No
JSF++:86	Concrete types should be used to represent simple independent concepts.	Should	No
JSF++:87	Hierarchies should be based on abstract classes.	Should	No
JSF++:88	Multiple inheritance shall only be allowed in the following restricted form: n interfaces plus m private implementations, plus at most one protected implementation.	Shall	No
JSF++:88.1	A stateful virtual base shall be explicitly declared in each derived class that accesses it.	Shall	No
JSF++:89	A base class shall not be both virtual and non-virtual in the same hierarchy.	Shall	Yes
JSF++:90	Heavily used interfaces should be minimal, general and abstract.	Should	No

JSF++:91	Public inheritance will be used to implement "is-a" relationships.	Will	No
JSF++:92	Subclass methods must expect less and deliver more than the base class methods they override	Will	No
JSF++:93	"has-a" or "is-implemented-in-terms-of" relationships will be modeled through membership or non-public inheritance.	Will	No
JSF++:94	An inherited nonvirtual function shall not be redefined in a derived class.	Shall	No
JSF++:95	An inherited default parameter shall never be redefined.	Shall	Yes
JSF++:96	Arrays shall not be treated polymorphically.	Shall	Yes
JSF++:97	Arrays shall not be used in interfaces. Instead, the Array class should be used.	Shall	No
JSF++:97.1	Neither operand of an equality operator (== or !=) shall be a pointer to a virtual member function.	Shall	No
JSF++:98	Every nonlocal name, except main(), should be placed in some namespace.	Should	No
JSF++:99	Namespaces will not be nested more than two levels deep.	Will	No
JSF++:100	Elements from a namespace should be selected as specified.	Should	No
JSF++:101	Templates shall be reviewed as specified.	Shall	No
JSF++:102	Template tests shall be created to cover all actual template instantiations.	Shall	No
JSF++:103	Constraint checks should be applied to template arguments.	Should	No
JSF++:104	A template specialization shall be declared before its use.	Shall	No
JSF++:105	A template definition's dependence on its instantiation contexts should be minimized.	Should	No
JSF++:106	Specializations for pointer types should be made where appropriate.	Should	No
JSF++:107	Functions shall always be declared at file scope.	Shall	Yes
JSF++:108	Functions with variable numbers of arguments shall not be used.	Shall	Yes
JSF++:109	A function definition should not be placed in a class specification unless the function is intended to be inlined.	Should	No
JSF++:110	Functions with more than 7 arguments will not be used.	Will	Yes
JSF++:111	A function shall not return a pointer or reference to a non-static local object.	Shall	Yes
JSF++:112	Function return values should not obscure resource ownership.	Should	Yes
JSF++:113	Functions will have a single exit point.	Will	Yes
JSF++:114	All exit points of value-returning functions shall be through return statements.	Shall	Yes
JSF++:115	If a function returns error information, then that error information will be tested.	Will	Yes
JSF++: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.	Should	No
JSF++:117	Arguments should be passed by reference if NULL values are not possible	Should	No
JSF++:117.1	An object should be passed as const T& if the function should not change the value of the object.	Should	Yes
JSF++:117.2	An object should be passed as T& if the function may change the value of the object.	Should	No
JSF++:118	Arguments should be passed via pointers if NULL values are possible.	Should	No
JSF++:118.1	An object should be passed as const T* if its value should not be modified.	Should	No
JSF++:118.2	An object should be passed as T* if its value may be modified.	Should	No
JSF++:119	Functions shall not call themselves, either directly or indirectly (i.e. recursion shall not be allowed).	Shall	Yes
JSF++: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.	Should	No
JSF++:121	Only functions with 1 or 2 statements should be considered candidates for inline functions.	Should	No
JSF++:122	Trivial accessor and mutator functions should be inlined.	Should	No
JSF++:123	The number of accessor and mutator functions should be minimized.	Should	No
JSF++:124	Trivial forwarding functions should be inlined.	Should	No
JSF++:125	Unnecessary temporary objects should be avoided.	Should	No

JSF++:126	Only valid C++ style comments (//) shall be used.	Shall	Yes
JSF++:127	Code that is not used (commented out) shall be deleted.	Shall	Yes
JSF++:128	Comments that document actions or sources (e.g. tables, figures, paragraphs, etc.) outside of the file being documented will not be allowed.	Will	No
JSF++:129	Comments in header files should describe the externally visible behavior of the functions or classes being documented.	Should	No
JSF++: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.	Should	No
JSF++:131	One should avoid stating in comments what is better stated in code (i.e. do not simply repeat what is in the code).	Should	No
JSF++:132	Each variable declaration, typedef, enumeration value, and structure member will be commented.	Will	No
JSF++: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).	Will	No
JSF++:134	Assumptions (limitations) made by functions should be documented in the function's preamble.	Should	No
JSF++:135	Identifiers in an inner scope shall not use the same name as an identifier in an outer scope, and therefore hide that identifier.	Shall	Yes
JSF++:136	Declarations should be at the smallest feasible scope.	Should	Yes
JSF++:137	All declarations at file scope should be static where possible.	Should	Yes
JSF++:138	Identifiers shall not simultaneously have both internal and external linkage in the same translation unit.	Shall	No
JSF++:139	External objects will not be declared in more than one file.	Will	Yes
JSF++:140	The register storage class specifier shall not be used.	Shall	No
JSF++:141	A class, structure, or enumeration will not be declared in the definition of its type.	Will	No
JSF++:142	All variables shall be initialized before use.	Shall	Yes
JSF++:143	Variables will not be introduced until they can be initialized with meaningful values.	Will	No
JSF++:144	Braces shall be used to indicate and match the structure in the non-zero initialization of arrays and structures.	Shall	Yes
JSF++:145	In an enumerator list, the '=' construct shall not be used to explicitly initialize members other than the first, unless all items are explicitly initialized.	Shall	Yes
JSF++:146	Floating point implementations shall comply with a defined floating point standard. The standard that will be used is the ANSI/IEEE Std 754.	Shall	No
JSF++:147	The underlying bit representations of floating point numbers shall not be used in any way by the programmer.	Shall	Yes
JSF++:148	Enumeration types shall be used instead of integer types (and constants) to select from a limited series of choices.	Shall	No
JSF++:149	Octal constants (other than zero) shall not be used.	Shall	Yes
JSF++:150	Hexadecimal constants will be represented using all uppercase letters.	Will	No
JSF++:151	Numeric values in code will not be used; symbolic values will be used instead.	Will	No
JSF++:151.1	A string literal shall not be modified.	Shall	Yes
JSF++:152	Multiple variable declarations shall not be allowed on the same line.	Shall	Yes
JSF++:153	Unions shall not be used.	Shall	Yes
JSF++:154	Bit-fields shall have explicitly unsigned integral or enumeration types only.	Shall	Yes
JSF++:155	Bit-fields will not be used to pack data into a word for the sole purpose of saving space.	Will	No
JSF++:156	All the members of a structure (or class) shall be named and shall only be accessed via their names.	Shall	No
JSF++:157	The right hand operand of a && or || operator shall not contain side effects.	Shall	Yes

JSF++:158	The operands of a logical && or || shall be parenthesized if the operands contain binary operators.	Shall	Yes
JSF++:159	Operators ||, &&, and unary & shall not be overloaded.	Shall	Yes
JSF++:160	An assignment expression shall be used only as the expression in an expression statement.	Shall	Yes
JSF++:162	Signed and unsigned values shall not be mixed in arithmetic or comparison operations.	Shall	Yes
JSF++:163	Unsigned arithmetic shall not be used.	Shall	No
JSF++:164	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).	Shall	Yes
JSF++:164.1	The left-hand operand of a right-shift operator shall not have a negative value.	Shall	No
JSF++:165	The unary minus operator shall not be applied to an unsigned expression.	Shall	Yes
JSF++:166	The sizeof operator will not be used on expressions that contain side effects.	Will	Yes
JSF++:167	The implementation of integer division in the chosen compiler shall be determined, documented and taken into account.	Shall	No
JSF++:168	The comma operator shall not be used.	Shall	Yes
JSF++:169	Pointers to pointers should be avoided when possible.	Should	Yes
JSF++:170	More than 2 levels of pointer indirection shall not be used.	Shall	Yes
JSF++:171	Relational operators shall not be applied to pointer types except where both operands are of the same type and point to the same entity.	Shall	Yes
JSF++:173	The address of an object with automatic storage shall not be assigned to an object which persists after the object has ceased to exist.	Shall	Yes
JSF++:174	The null pointer shall not be de-referenced.	Shall	Yes
JSF++:175	A pointer shall not be compared to NULL or be assigned NULL; use plain 0 instead.	Shall	No
JSF++:176	A typedef will be used to simplify program syntax when declaring function pointers.	Will	No
JSF++:177	User-defined conversion functions should be avoided.	Should	No
JSF++:178	Down casting (casting from base to derived class) shall only be allowed through certain mechanisms.	Shall	No
JSF++:179	A pointer to a virtual base class shall not be converted to a pointer to a derived class.	Shall	No
JSF++:180	Implicit conversions that may result in a loss of information shall not be used.	Shall	Yes
JSF++:181	Redundant explicit casts will not be used.	Will	No
JSF++:182	Type casting from any type to or from pointers shall not be used.	Shall	Yes
JSF++:183	Every possible measure should be taken to avoid type casting.	Should	Yes
JSF++: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.	Shall	Yes
JSF++:185	C++ style casts (const_cast, reinterpret_cast, and static_cast) shall be used instead of the traditional C-style casts.	Shall	Yes
JSF++:186	There shall be no unreachable code.	Shall	Yes
JSF++:187	All non-null statements shall potentially have a side-effect.	Shall	Yes
JSF++:188	Labels will not be used, except in switch statements.	Will	No
JSF++:189	The goto statement shall not be used.	Shall	Yes
JSF++:190	The continue statement shall not be used.	Shall	Yes
JSF++:191	The break statement shall not be used (except to terminate the cases of a switch statement).	Shall	Yes
JSF++:192	All if, else if constructs will contain either a final else clause or a comment indicating why a final else clause is not necessary.	Will	Yes
JSF++:193	Every non-empty case clause in a switch statement shall be terminated with a break statement.	Shall	Yes
JSF++:194	All switch statements that do not intend to test for every enumeration value shall contain a final default clause.	Shall	Yes
JSF++:195	A switch expression will not represent a Boolean value.	Will	Yes

JSF++:196	Every switch statement will have at least two cases and a potential default.	Will	Yes
JSF++:197	Floating point variables shall not be used as loop counters.	Shall	Yes
JSF++:198	The initialization expression in a for loop will perform no actions other than to initialize the value of a single for loop parameter.	Will	Yes
JSF++: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.	Will	Yes
JSF++:200	Null initialize or increment expressions in for loops will not be used; a while loop will be used instead.	Will	Yes
JSF++:201	Numeric variables being used within a for loop for iteration counting shall not be modified in the body of the loop.	Shall	Yes
JSF++:202	Floating point variables shall not be tested for exact equality or inequality.	Shall	Yes
JSF++:203	Evaluation of expressions shall not lead to overflow/underflow (unless required algorithmically and then should be heavily documented).	Shall	Yes
JSF++:204	A single operation with side-effects shall only be used in certain contexts.	Shall	No
JSF++:204.1	The value of an expression shall be the same under any order of evaluation that the standard permits.	Shall	Yes
JSF++:205	The volatile keyword shall not be used unless directly interfacing with hardware.	Shall	No
JSF++:206	Allocation/deallocation from/to the free store (heap) shall not occur after initialization.	Shall	Yes
JSF++:207	Unencapsulated global data will be avoided.	Will	No
JSF++:208	C++ exceptions shall not be used (i.e. throw, catch and try shall not be used.)	Shall	Yes
JSF++:209	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.	Shall	Yes
JSF++: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.)	Shall	Yes
JSF++:210.1	Algorithms shall not make assumptions concerning the order of allocation of nonstatic data members separated by an access specifier.	Shall	Yes
JSF++:211	Algorithms shall not assume that shorts, ints, longs, floats, doubles or long doubles begin at particular addresses.	Shall	Yes
JSF++:212	Underflow or overflow functioning shall not be depended on in any special way.	Shall	Yes
JSF++:213	No dependence shall be placed on C++'s operator precedence rules, below arithmetic operators, in expressions.	Shall	Yes
JSF++:214	Assuming that non-local static objects, in separate translation units, are initialized in a special order shall not be done.	Shall	Yes
JSF++:215	Pointer arithmetic will not be used.	Will	Yes
JSF++:216	Programmers should not attempt to prematurely optimize code.	Should	No
JSF++:217	Compile-time and link-time errors should be preferred over run-time errors.	Should	No
JSF++:218	Compiler warning levels will be set in compliance with project policies.	Will	Yes
JSF++: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.	Shall	No
JSF++:220	Structural coverage algorithms shall be applied against flattened classes.	Shall	No
JSF++: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.	Shall	No