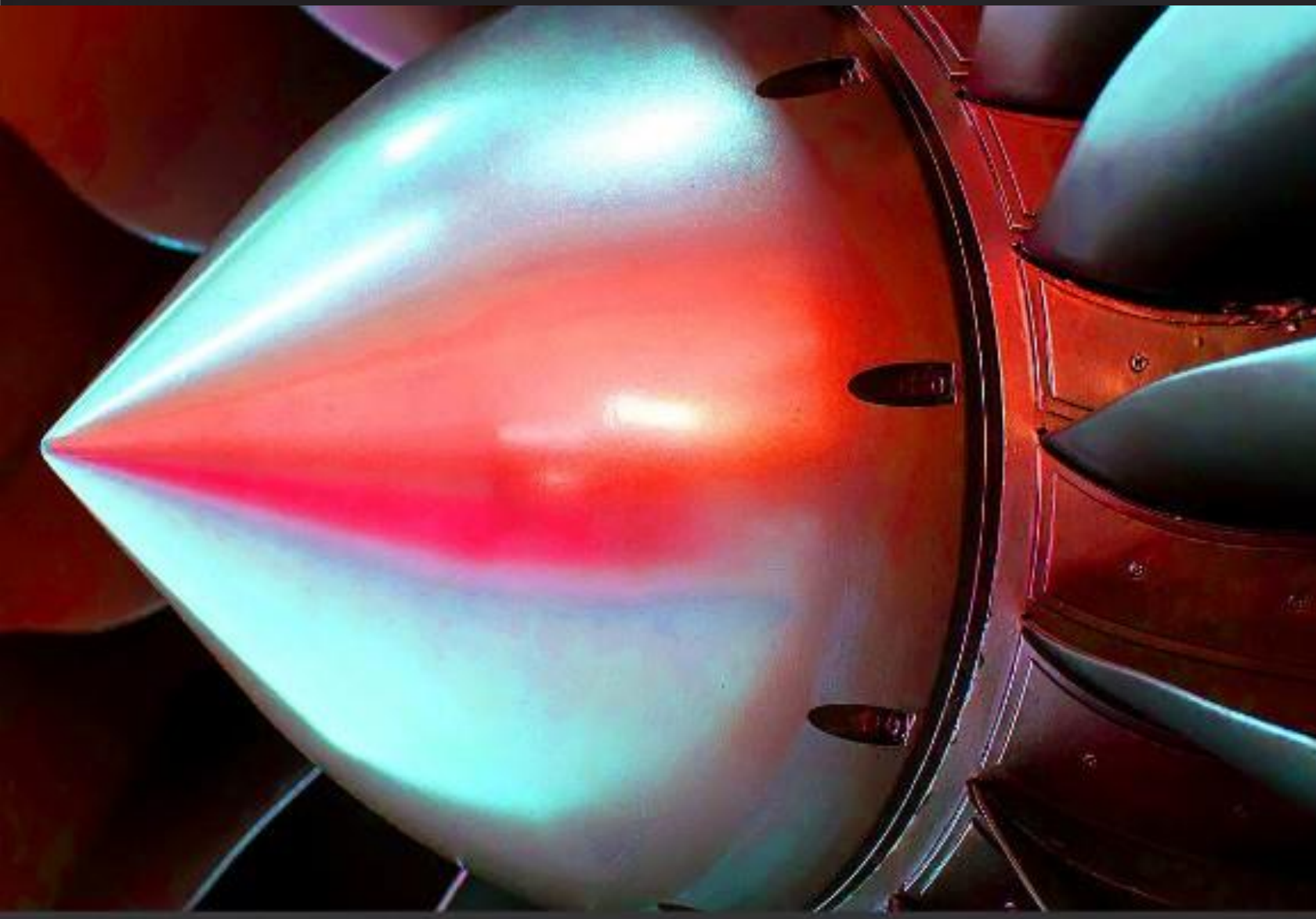
GRAMMATECH

JPL INSTITUTIONAL CODING STANDARD FORTHE C PROGRAMMING LANGUAGE | CLOSE & BROAD MAPPING TO CODESONAR® 6.2



TRUSTED LEADERS OF SOFTWARE ASSURANCE AND ADVANCED CYBER-SECURITY SOLUTIONS

WWW.GRAMMATECH.COM

INTRODUCTION

The following table shows the CodeSonar warning classes that are associated with JPL rules.

Note close and broad mappings are identical, thus only one chart below to reflect bothclose and broad mapping.

JPL Rule ID & Text		CodeSonar 6.2 Class Name
JPL:1	Do not stray outside the language definition.	C++ Comment in CGNU Extension GNU Typeof Microsoft Extension
JPL:2	Compile with all warnings enabled; use static source code analyzers.	Not All Warnings Are Enabled Warnings Not Treated As Errors
JPL:3	Use verifiable loop bounds for all loops meant to be terminating.	Potential Unbounded Loop
JPL:4	Do not use direct or indirect recursion.	Recursion
JPL:5	Do not use dynamic memory allocation after task initialization.	Dynamic Allocation After Initialization Use of <stdlib.h> Allocator/Deallocator Use of <stdlib.h> Allocator/Deallocator Macro
JPL:7	Do not use task delays for task synchronization.	Task Delay Function
JPL:9	Place restrictions on the use of semaphores and locks.	Conflicting Lock Order Lock/Unlock MismatchLocked Twice Missing Lock AcquisitionMissing Lock Release Nested Locks Unknown Lock
JPL:11	Do not use goto, setjmp or longjmp.	Goto Statement Use of <setjmp.h> Use of longjmp Use of setjmp
JPL:12	Do not use selective value assignments to elements of an enum list.	Inconsistent Enumerator Initialization
JPL:13	Declare data objects at smallest possible level of scope.	Non-distinct Identifiers: NestedScope Scope Could Be File Static Scope Could Be Local Static
JPL:14	Check the return value of non-void functions, or explicitly cast to (void).	Ignored Return Value

JPL:15	Check the validity of values passed to functions.	Unchecked Parameter Dereference
JPL:16	Use static and dynamic assertions as sanity checks.	Not Enough Assertions
JPL:17	Use U32, I16, etc instead of predefined C data types such as int, short, etc.	Basic Numerical Type Used
JPL:18	Make the order of evaluation in compound expressions explicit.	Missing Parentheses
JPL:19	Do not use expressions with side effects.	Assignment Result in ExpressionCondition Contains Side Effects Side Effects in Expression withDecrement Side Effects in Expression withIncrement Side Effects in Initializer List Side Effects in Logical OperandSide Effects in sizeof
JPL:20	Make only very limited use of the C pre-processor.	## Follows # Operator Conditional Compilation Macro Does Not End With } or) Macro Does Not Start With { or (Macro Name is C Keyword Macro Uses # Operator Macro Uses ## Operator Non-Boolean PreprocessorExpression Preprocessing Directives in MacroArgument Recursive Macro Unbalanced Parenthesis Use of <stdio.h> Input/Output Macro Use of <wchar.h> Input/OutputMacro Variadic Macro
JPL:21	Do not define macros within a function or a block.	Macro Defined in Function Body Macro Undefined in Function Body

JPL:22	Do not undefine or redefine macros.	Macro Undefinition of ReservedName Use of #undef
JPL:23	Place #else, #elif, and #endif in the same file as the matching #if or #ifdef.	No Matching #endif No Matching #if
JPL:24	Place no more than one statement or declaration per line of text.	Multiple Declarations On Line Multiple Statements On Line
JPL:25	Use short functions with a limited number of parameters.	Ellipsis Function Too Long Too Many Parameters
JPL:26	Use no more than two levels of indirection per declaration.	Too Much Indirection in Declaration
JPL:27	Use no more than two levels of dereferencing per object reference.	Too Many Dereferences
JPL:28	Do not hide dereference operations inside macros or typedefs.	Macro Uses -> Operator Macro Uses Unary * OperatorMacro Uses [] Operator Pointer Type Inside Typedef
JPL:30	Do not cast function pointers into other types.	Conversion from Function Pointer Function Pointer Conversion
JPL:31	Do not place code or declarations before an #include directive.	Code Before #include

GrammaTech is a leading global provider of application testing (AST) solutions used by the world’smost security conscious organizations to detect, measure, analyze and resolve vulnerabilitiesfor 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.