Updates in 10.4.2

In this release, we've focused on enhancing code analysis capabilities and extending our support for the AUTOSAR C++ 14 coding standard.

Support for IDEs

We've added support for:

- Eclipse 4.9

Extended Compliance Packs

We've extended the Automotive Compliance Pack to help you enforce compliance with the AUTOSAR C++14 coding guidelines.

- The "AUTOSAR C++ 14 Coding Guidelines" test configuration has been updated to help you ensure compliance with the AUTOSAR C++ 14 Coding Guidelines version 18.10.
- We've significantly extended the "AUTOSAR C++ 14 Coding Guidelines" test configuration by adding new rules that enforce the standard.

Compliance Packs require dedicated license features to be activated. Contact Parasoft Support for more details on Compliance Packs licensing.

Improved Performance of Static Analysis

We've enhanced C/C++test's static code analysis capabilities to improve performance . C/C++test is now optimized to reduce the analysis time in incremental CI builds and shorten the feedback cycles between modifying the code and reviewing static analysis results. The optimizations will also help you accelerate software development on desktop by speeding up the analysis preformed locally in your IDE. See Incremental Static Analysis for details.

Enhanced Code Coverage Analysis

We've enhanced the code coverage capabilities to facilitate monitoring coverage metrics from system and functional testing.

Standalone Code Coverage

C/C++test now ships with `cpptestcc`–a lightweight code coverage utility that can be easily integrated into your build system, saving the time and complexity of creating projects dedicated for code instrumentation. Now you can effortlessly create the test binary as part of your regular build process, and then upload the coverage data to your IDE to review the results using regular C/C++test's reporting capabilities. You can monitor a full range of coverage metrics, including MC/DC coverage, decision coverage, function coverage, and more. See Collecting Application Coverage with cpptestcc for details.

Merging Code Coverage Results From Multiple Test Runs

You can now aggregate code coverage results from multiple testing sessions to generate a cumulative report. See Merging Results from Multiple Test Runs for details.

Extended Support for Collecting Assembly Coverage

We've extended support for collecting coverage information at the assembly level by adding support for Linux-based development environments. The assembly coverage tool now supports GCC compilers for x86 (32-bit) platforms; see Assembly Code Coverage for details.

New and Updated Code Analysis Rules

We've added new static analysis rules to extend coverage of compliance standards; see New Rules, Updated Rules, and Removed Rules for details.

Flow Analysis Improvements

- You can now specify the functions you always want to be analyzed when encountered on the execution path; see Flow Analysis for details.
- We've improved reporting of setup problems.

New Test Configurations

We've added the following test configurations:

- Load Test Results
- Load Archived Results

Resolved Bugs and FRs

Bug/FR ID	Description
CPP-36917	MISRA2012-13_2_f (MISRA2004-12_2_f) reports false positive
CPP-38031	MISRA2012-RULE-11_1_a: false positive when an element of multidimensional array is initialized
CPP-39310	error: 'visibility' was not declared in this scope
CPP-39915	MISRA2004-10_6 (MISRA2012-RULE-7_2) rule reports incorrectly on signed values used in initializer of big array
CPP-40091	MISRA2012-RULE-10_3_b (CODSTA-163_b) reports false positives for ARM v7.8 compiler
CPP-40400	[RW] Add property for detecting use of default arguments in calls (for Argument node)
CPP-40561	[CERTC] Extend rule: Understand the type issues associated with variadic functions (scanf)
CPP-41278	MISRA2004-12_8 (MISRA-038) does not report violation when enumeration const is used as RHS operand of shift operator
CPP-41279	METRICS-19 does not report violation when there are no comments in function
CPP-41829	line 160: internal error: assertion failed at: "error.c", line 1123
CPP-42436	The statement 'if' (or other) is incorrectly detected in context of block (C only)
CPP-42458	[VS] Improve support for testing projects in solution folders.
CPP-42462	line 66: internal error: assertion failed at: "scope_stk.c", line 5106
CPP-42467	Error reading command line option -edg.restrict_keyword_enabled.
CPP-42524	MISRA-071_b (MISRA2012-RULE-17_3) should not report on calls of built-in functions
CPP-42551	[RW] RuleWizard displays incorrect version number
CPP-42577	Parse error on GNU GCC 8 *_attribute_*((fallthrough))
CPP-42578	cwc crashes on custom rule analyzing template code
CPP-42663	PB.33_b: incorrectly flagging rule for enumerations that are defined with the "packed" attribute.
CPP-42687	C++test is unable to handle user-defined literals.
CPP-42795	'cppmode' tag with parens has no effects
CPP-42859	[DOCS] Correct license edition names
CPP-42878	MISRA2004-16_7 reports incorrect violation on parameter of pointer to struct containing anonymous union
CPP-42879	CODSTA-132 reports incorrect violation on parameter of pointer to struct containing anonymous union
CPP-42901	IAR EWARM parse error with __packed
CPP-42936	Expression a->b is detected in RW as a.b when 'b' is a member of anonymous union
CPP-42951	OPT-03 (AUTOSAR A0-1-4) does not support attribute 'unused' (c++11)
CPP-43007	Intel icc 18: Linker options are not extracted from BDF files
CPP-43010	CODSTA-52 (JSF-198) reports incorrect violation when an assignment operator function is used to initialization of a loop counter
CPP-43080	METRICS-29, -33 should report only for defined and instantiated template functions

CPP-43103	cwc crashes with exit code 4 on custom rule
CPP-43105	[RW] Extend property 'OperatorName' for nodes: a.b, a->b, a.*b, a->*b
CPP-43299	[DOCS] CPPTEST_REPORT_ENUM actually requires 3 arguments
CPP-43407	[TEST_EDITOR] Improve widgets behavior when scrolling and resizing
FA-5720	False positive BD-PB-OVERFNZT
FA-6283	BD-PB-OVERFNZT False Positive
FA-6745	Terminator function is not recognized
FA-6758	Current block is not ScopeBlock error when building graphs
FA-6801	BD-PB-OVERFNZT false positive
FA-6807	False positive of BD-RES-LEAKS
FA-6811	BD-PB-ARRAY showing a violation for index that is not set to be negative
FA-6857	BD-PB-EOFCOMP rule should also cover fputs method.
FA-6974	BD-PB-STDEXC false positive
FA-6976	BD-PB-NP false positive on bool/int comparison evaluation

---

New Rules

We've added the following rules:

Rule ID	Header
AUTOSAR-A0_1_6-a	A project shall not contain unused type declarations
AUTOSAR-A0_4_2-a	Type long double shall not be used
AUTOSAR-A0_4_4-a	Validate values passed to library functions
AUTOSAR-A10_3_1-a	Only one of virtual override or final should be specified in a member function declaration
AUTOSAR-A10_3_5-a	A user-defined assignment operator shall not be virtual
AUTOSAR-A10_4_1-a	Hierarchies should be based on abstract classes
AUTOSAR-A11_0_1-a	A non-POD type should be defined as class
AUTOSAR-A11_0_2-a	Structs should only contain public data members and should not be a base or inherit
AUTOSAR-A12_0_2-a	Do not compare objects of a class that may contain padding bits with C standard library functions
AUTOSAR-A12_4_2-a	If a public destructor of a class is non-virtual then the class should be declared final
AUTOSAR-A12_7_1-a	Define special members =default if the behavior is equivalent
AUTOSAR-A12_8_3-a	Do not rely on the value of a moved-from object
AUTOSAR-A13_1_2-a	User defined suffixes of the user defined literal operators shall start with underscore followed by one or more letters
AUTOSAR-A13_1_3-a	User defined literals operators shall only perform conversion of passed parameters
AUTOSAR-A13_2_2-a	A binary arithmetic operator and a bitwise operator shall return a 'prvalue'
AUTOSAR-A13_2_3-a	A relational operator shall return a boolean value
AUTOSAR-A13_5_3-a	Do not use user-defined conversion functions
AUTOSAR-A13_5_4-a	When two operators are opposites (such as == and !=) it is appropriate to define both
AUTOSAR-A13_5_4-b	When two operators are opposites (such as == and !=) both will be defined and one will be defined in terms of the other
AUTOSAR-A13_5_5-a	Prefer non-member operators than member ones to support mixed-mode arithmetic
AUTOSAR-A13_6_1-a	Digit sequences separators ' shall only be used consistently

AUTOSAR-A14_5_1-a	A copy constructor shall be declared when there is a template constructor with a single parameter that is a generic parameter
AUTOSAR-A14_7_2-a	Template specialization shall be declared in the same file as the primary template or a user-defined type for which the specialization is declared
AUTOSAR-A14_8_2-a	Overloaded function templates shall not be explicitly specialized
AUTOSAR-A15_0_2-a	Ensure resources are freed
AUTOSAR-A15_1_1-a	Only use instances of std::exception for exceptions
AUTOSAR-A15_1_5-a	Do not throw an exception across execution boundaries
AUTOSAR-A15_4_2-a	Avoid throwing exceptions from functions that are declared not to throw
AUTOSAR-A15_5_1-a	Never allow an exception to be thrown from a destructor deallocation and swap
AUTOSAR-A15_5_1-b	All user-provided move constructors and move assignment operators shall not exit with an exception
AUTOSAR-A16_2_2-a	A file should directly include only the headers that contain declarations and definitions required to compile that file
AUTOSAR-A16_6_1-a	#error directive shall not be used
AUTOSAR-A16_7_1-a	The #pragma directive shall not be used
AUTOSAR-A17_1_1-b	Wrap use of the C Standard Library
AUTOSAR-A17_6_1-a	Do not modify the standard namespaces 'std' and 'posix'
AUTOSAR-A18_1_4-a	A pointer pointing to an element of an array of objects shall not be passed to a smart pointer of single object type
AUTOSAR-A18_1_6-a	All std::hash specializations for user-defined types shall have a noexcept function call operator
AUTOSAR-A18_5_10-a	Do not pass a pointer that has insufficient storage capacity or that is not suitably aligned for the object being constructed to placement 'new'
AUTOSAR-A18_5_10-b	An overhead should be used when an array of objects is passed to the placement 'new' allocation function
AUTOSAR-A18_5_11-a	Write operator delete if you write operator new
AUTOSAR-A18_5_11-b	Write operator delete[] if you write operator new[]
AUTOSAR-A18_5_3-c	Properly deallocate dynamically allocated resources
AUTOSAR-A18_5_4-a	Define both sized and unsized versions of operator delete
AUTOSAR-A18_5_5-c	Properly define new handlers
AUTOSAR-A18_5_8-a	Use allocation by declaration rather than by new or malloc
AUTOSAR-A18_5_9-a	The user defined 'new' operator should throw the 'std::bad_alloc' exception when the allocation fails
AUTOSAR-A18_9_2-a	Use std::forward to forward universal references
AUTOSAR-A18_9_3-a	Do not use std::move on objects declared with the const or consttype
AUTOSAR-A2_10_1-a	Identifier declared in a local or function prototype scope shall not hide an identifier declared in a global or namespace scope
AUTOSAR-A2_10_1-b	Identifiers declared in an inner local scope should not hide identifiers declared in an outer local scope
AUTOSAR-A2_10_1-c	Identifiers declared in a local scope should not hide identifiers declared in a class scope
AUTOSAR-A2_10_1-d	Identifiers declared in a class scope should not hide identifiers declared in a global or namespace scope
AUTOSAR-A2_10_1-e	Identifiers declared in an inner class scope should not hide identifiers declared in outer class scope
AUTOSAR-A2_10_5-a	No object or function identifier with static storage duration should be reused
AUTOSAR-A2_10_5-b	No object or function identifier with static storage duration should be reused
AUTOSAR-A2_10_6-a	If an identifier refers to a type it shall not also refer to an object or a function in the same scope
AUTOSAR-A2_10_6-b	If an identifier refers to a type it shall not also refer to an object or a function in the same scope
AUTOSAR-A2_10_6-c	If an identifier refers to a type it shall not also refer to an object or a function in the same scope
AUTOSAR-A2_13_1-a	Only those escape sequences that are defined in ISO/IEC 14882:2003 shall be used

AUTOSAR-A2_13_2-a	String literals with different encoding prefixes shall not be concatenated
AUTOSAR-A2_13_3-a	Type wchar_t shall not be used
AUTOSAR-A2_13_4-a	A string literal shall not be modified
AUTOSAR-A2_13_5-a	Hexadecimal constants will be represented using all uppercase letters
AUTOSAR-A2_13_6-a	Universal character names shall be used only inside character or string literals
AUTOSAR-A2_3_1-a	Only use characters defined in ISO C standard
AUTOSAR-A2_5_2-a	The following digraphs will not be used %%,::%:%:%:
AUTOSAR-A2_7_1-a	Line-splicing shall not be used in // comments
AUTOSAR-A2_7_2-a	Sections of code should not be "commented out"
AUTOSAR-A20_8_1-a	Do not store an already-owned pointer value in an unrelated smart pointer
AUTOSAR-A20_8_2-a	Use smart pointers when passing a pointer to an object in a thread
AUTOSAR-A20_8_3-a	Use smart pointers when passing a pointer to an object in a thread
AUTOSAR-A20_8_4-a	Consider using 'std::unique_ptr' instead of 'std::shared_ptr' for local objects
AUTOSAR-A20_8_5-a	'std::make_unique' shall be used to construct objects owned by 'std::unique_ptr'
AUTOSAR-A20_8_6-a	Prefer 'std::make_shared' to the direct use of new
AUTOSAR-A20_8_7-a	Avoid cyclic shared_ptr references
AUTOSAR-A21_8_1-a	Do not pass incorrect values to ctype.h library functions
AUTOSAR-A23_0_2-a	Do not modify container while iterating over it
AUTOSAR-A23_0_2-b	Use valid references pointers and iterators to reference elements of a basic_string
AUTOSAR-A25_1_1-a	Make predicates const pure functions
AUTOSAR-A25_4_1-a	For associative containers never use comparison function returning true for equal values
AUTOSAR-A26_5_1-a	Do not use the rand() function for generating pseudorandom numbers
AUTOSAR-A26_5_2-a	Properly seed pseudorandom number generators
AUTOSAR-A27_0_1-c	Prevent buffer overflows from tainted data
AUTOSAR-A27_0_1-d	Avoid buffer overflow from tainted data due to defining incorrect format limits
AUTOSAR-A27_0_1-e	Avoid buffer read overflow from tainted data
AUTOSAR-A27_0_1-f	Avoid buffer write overflow from tainted data
AUTOSAR-A27_0_1-g	Protect against command injection
AUTOSAR-A27_0_1-h	Exclude unsanitized user input from format strings
AUTOSAR-A27_0_2-c	Use vector and string instead of arrays
AUTOSAR-A27_0_2-d	Avoid accessing arrays out of bounds
AUTOSAR-A27_0_2-e	Prevent buffer overflows from tainted data
AUTOSAR-A27_0_2-f	Avoid buffer write overflow from tainted data
AUTOSAR-A27_0_2-g	Avoid using unsafe string functions which may cause buffer overflows
AUTOSAR-A27_0_3-a	Do not alternately input and output from a stream without an intervening flush or positioning call
AUTOSAR-A3_1_2-a	Header files should have a file extension of: ".h" ".hpp" or ".hxx"
AUTOSAR-A3_1_5-a	Member functions shall not be defined within the no-template class definition
AUTOSAR-A3_1_6-a	Trivial accessor and mutator functions should be inlined
AUTOSAR-A3_8_1-a	Do not use resources that have been freed
AUTOSAR-A3_8_1-b	The address of an object with automatic storage shall not be returned from a function

AUTOSAR-A3_8_1-c	The address of an object with automatic storage shall not be assigned to another object that may persist after the first object has ceased to exist
AUTOSAR-A3_8_1-d	Do not point to a wrapped object that has been freed
AUTOSAR-A4_7_1-h	Avoid integer overflows
AUTOSAR-A5_0_4-a	Don't treat arrays polymorphically
AUTOSAR-A5_0_4-b	A pointer to an array of derived class objects should not be converted to a base class pointer
AUTOSAR-A5_0_4-c	Do not treat arrays polymorphically
AUTOSAR-A5_1_7-a	A lambda shall not be an operand to typeid
AUTOSAR-A5_10_1-a	A pointer to member virtual function shall only be tested for equality with null-pointer-constant
AUTOSAR-A5_2_5-b	Avoid accessing arrays and pointers out of bounds
AUTOSAR-A5_2_5-c	A pointer operand and any pointer resulting from pointer arithmetic using that operand shall both address elements of the same array
AUTOSAR-A5_2_5-d	Avoid tainted data in array indexes
AUTOSAR-A5_2_6-a	Each operand of a logical '' or '||' shall be a postfix-expression
AUTOSAR-A5_3_2-a	Avoid null pointer dereferencing
AUTOSAR-A5_3_3-a	Do not delete objects with incomplete class at the point of deletion
AUTOSAR-A5_6_1-a	Avoid division by zero
AUTOSAR-A6_5_4-a	The initialization expression in a for loop will perform no actions other than to initialize the value of a single for loop parameter
AUTOSAR-A6_5_4-b	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
AUTOSAR-A7_1_3-a	Place CV-qualifiers on the right hand side of the type they apply to
AUTOSAR-A7_1_7-b	Multiple variable declarations shall not be allowed on the same line
AUTOSAR-A7_1_7-c	Each variable should be declared in a separate declaration statement
AUTOSAR-A7_1_9-a	A class structure or enumeration will not be declared in the definition of its type
AUTOSAR-A7_2_5-a	Enumeration types shall be used instead of integer types (and constants) as case labels
AUTOSAR-A7_3_1-a	Write a using declaration to redeclare overloaded functions
AUTOSAR-A7_5_1-a	A function shall not return a pointer or a reference to a parameter that is passed by const reference
AUTOSAR-A7_6_1-a	Never return from functions that should not return
AUTOSAR-A8_4_10-a	A parameter shall be passed by reference if it can't be NULL
AUTOSAR-A8_4_11-a	A smart pointer shall only be used as a parameter type if it expresses lifetime semantics
AUTOSAR-A8_4_12-a	Do not pass std::unique_ptr by const reference
AUTOSAR-A8_4_12-b	A smart pointer shall only be used as a parameter type if it expresses lifetime semantics
AUTOSAR-A8_4_12-c	A parameter should only be declared as a non-const lvalue reference to 'std::shared_ptr' or 'std::unique_ptr' if the function replaces the managed object
AUTOSAR-A8_4_12-d	Do not declare the type of a parameter as an rvalue reference to 'std::shared_ptr' or 'std::unique_ptr'
AUTOSAR-A8_4_13-a	A smart pointer shall only be used as a parameter type if it expresses lifetime semantics
AUTOSAR-A8_4_13-b	A parameter should only be declared as a non-const lvalue reference to 'std::shared_ptr' or 'std::unique_ptr' if the function replaces the managed object
AUTOSAR-A8_4_13-c	Do not declare the type of a parameter as an rvalue reference to 'std::shared_ptr' or 'std::unique_ptr'
AUTOSAR-A8_4_3-a	Pass objects by reference instead of by value
AUTOSAR-A8_4_3-b	Declare reference parameters as const references whenever possible
AUTOSAR-A8_4_5-a	Use std::move() on rvalue references and std::forward() on forwarding references
AUTOSAR-A8_4_6-a	Use std::move() on rvalue references and std::forward() on forwarding references

AUTOSAR-A8_4_7-a	Pass built-in-types by value unless you are modifying them
AUTOSAR-A8_4_7-b	Pass small objects with a trivial copy constructor by value
AUTOSAR-A8_4_9-a	Declare reference parameters as const references whenever possible
AUTOSAR-A8_5_0-a	Avoid use before initialization
AUTOSAR-A9_6_2-a	Do not declare member variables as bit-fields
AUTOSAR-M2_7_1-a	The character sequence /* shall not be used within a C-style comment
AUTOSAR-M9_6_4-a	Named bit-fields with signed integer type shall have a length of more than one bit
BD-PB-ARRPTR	A pointer pointing to an element of an array of objects shall not be passed to a smart pointer of single object type
BD-PB-NOEXCEPT	Avoid throwing exceptions from functions that are declared not to throw
BD-PB-OVERLAP	An object shall not be assigned or copied to an overlapping object
BD-PB-REFPARAM	A parameter shall be passed by reference if it can't be NULL
BD-RES-BADDEALLOC	Properly deallocate dynamically allocated resources
BD-RES-CSP	Avoid cyclic shared_ptr references
BD-SECURITY-TDINPUT	Exclude unsanitized user input from format strings
CERT_CPP-ERR50-m	Avoid throwing exceptions from functions that are declared not to throw
CERT_CPP-EXP59-a	Use offsetof() on valid types and members
CERT_CPP-EXP62-a	Do not compare objects of a class that may contain padding bits with C standard library functions
CERT_CPP-MEM51-d	Properly deallocate dynamically allocated resources
CERT_CPP-MEM54-a	Do not pass a pointer that has insufficient storage capacity or that is not suitably aligned for the object being constructed to placement 'new'
CERT_CPP-MEM54-b	An overhead should be used when an array of objects is passed to the placement 'new' allocation function
CERT_CPP-MEM57-a	Avoid using the default operator 'new' for over-aligned types
CERT_CPP-STR50-b	Avoid overflow due to reading a not zero terminated string
CERT_CPP-STR50-c	Avoid overflow when writing to a buffer
CERT_CPP-STR50-d	Avoid accessing arrays out of bounds
CERT_CPP-STR50-e	Prevent buffer overflows from tainted data
CERT_CPP-STR50-f	Avoid buffer write overflow from tainted data
CERT_CPP-STR50-g	Avoid using unsafe string functions which may cause buffer overflows
CODSTA-CPP-100	Do not compare objects of a class that may contain padding bits with C standard library functions
CODSTA-CPP-101	A relational operator shall return a boolean value
CODSTA-CPP-102	A binary arithmetic operator and a bitwise operator shall return a 'prvalue'
CODSTA-CPP-99	Use offsetof() on valid types and members
CODSTA-MCPP-16_e	'std::make_unique' shall be used to construct objects owned by 'std::unique_ptr'
CODSTA-MCPP-23	If a public destructor of a class is non-virtual then the class should be declared final
CODSTA-MCPP-24	Only one of virtual override or final should be specified in a member function declaration
CODSTA-MCPP-25	Digit sequences separators ' shall only be used consistently
CODSTA-MCPP-26	A pointer to member virtual function shall only be tested for equality with null-pointer-constant
CODSTA-MCPP-27	All std::hash specializations for user-defined types shall have a noexcept function call operator
CODSTA-MCPP-28	A lambda shall not be an operand to typeid
CODSTA-MCPP-29	Use smart pointers when passing a pointer to an object in a thread

CODSTA-MCPP-30	Consider using 'std::unique_ptr' instead of 'std::shared_ptr' for local objects
CODSTA-MCPP-31	Define both sized and unsized versions of operator delete
CODSTA-MCPP-33	User defined literals operators shall only perform conversion of passed parameters
CODSTA-MCPP-34	A smart pointer shall only be used as a parameter type if it expresses lifetime semantics
CODSTA-MCPP-35	A parameter should only be declared as a non-const lvalue reference to 'std::shared_ptr' or 'std::unique_ptr' if the function replaces the managed object
CODSTA-MCPP-36	Do not declare the type of a parameter as an rvalue reference to 'std::shared_ptr' or 'std::unique_ptr'
EXCEPT-21	All user-provided move constructors and move assignment operators shall not exit with an exception
HICPP-13_2_2-b	A binary arithmetic operator and a bitwise operator shall return a 'prvalue'
HICPP-15_3_2-c	Avoid throwing exceptions from functions that are declared not to throw
HICPP-5_3_3-c	Properly deallocate dynamically allocated resources
MISRA2008-15_5_2_b	Avoid throwing exceptions from functions that are declared not to throw
MISRA2008-15_5_3_l	Avoid throwing exceptions from functions that are declared not to throw
MISRA2012-DIR_4_14_l	Exclude unsanitized user input from format strings
MISRA2012-RULE_19_1_c	An object shall not be assigned or copied to an overlapping object
MISRAC2012-DIR_4_14-l	Exclude unsanitized user input from format strings
MISRAC2012-RULE_19_1-c	An object shall not be assigned or copied to an overlapping object
MRM-54	Avoid using the default operator 'new' for over-aligned types
MRM-55	Do not pass a pointer that has insufficient storage capacity or that is not suitably aligned for the object being constructed to placement 'new'
MRM-55_b	An overhead should be used when an array of objects is passed to the placement 'new' allocation function
NAMING-51	User defined suffixes of the user defined literal operators shall start with underscore followed by one or more letters
NAMING-52	Universal character names shall be used only inside character or string literals
NAMING-53	Header files should have a file extension of: ".h" ".hpp" or ".hxx"
OOP-55	A non-POD type should be defined as class
OOP-56	A user-defined assignment operator shall not be virtual
OOP-57	Structs should only contain public data members and should not be a base or inherit
PB-38_b	String literals with different encoding prefixes shall not be concatenated
PB-39_b	A function shall not return a pointer or a reference to a parameter that is passed by const reference
PORT-32	Type long double shall not be used
PORT-33	Type wchar_t shall not be used
PREPROC-22	#error directive shall not be used
PREPROC-23	The #pragma directive shall not be used
PREPROC-24	All macro identifiers in preprocessor directives shall be defined before use except in #ifdef and #ifndef preprocessor directives and the defined() operator
TEMPL-16	Template specialization shall be declared in the same file as the primary template or a user-defined type for which the specialization is declared

Updated Rules

We've updated following static analysis rules to improve analysis results:

Rule Category	Rule IDs
AUTOSAR C++14 Coding Guidelines	AUTOSAR-A0_1_4-a,AUTOSAR-A12_4_1-a, AUTOSAR-A27_0_1-b, AUTOSAR-A27_0_2-a, AUTOSAR-A2_11_1-a, AUTOSAR-A2_8_1-a, AUTOSAR-A5_0_1-b, AUTOSAR-A5_0_1-f, AUTOSAR-A5_5_1-a, AUTOSAR-A8_5_1-a, AUTOSAR-A9_6_1-a, AUTOSAR-M0_1_1-b, AUTOSAR-M0_1_3-a, AUTOSAR-M0_1_3-b, AUTOSAR-M0_3_1-g. AUTOSAR-M11_0_1-a, AUTOSAR-M5_8_1-a
Flow Analysis	BD-CO-ITMOD, BD-PB-ERRNO, BD-PB-OVERFNZT, BD-PB-OVERFNZT, BD-SECURITY-RAND
SEI CERT C	CERT_C-ARR38-d, CERT_C-DCL01-b, CERT_C-DCL10-a, CERT_C-DCL11-a, CERT_C-DCL11-b, CERT_C-DCL11-c, CERT_C-DCL11-d, CERT_C-DCL11-e, CERT_C-DCL11-f, CERT_C-ERR30-a, CERT_C-ERR32-a, CERT_C-EXP10-b, CERT_C-EXP30-b, CERT_C-EXP37-d, CERT_C-FIO41-b, CERT_C-FIO47-a, CERT_C-FIO47-b, CERT_C-FIO47-c, CERT_C-FIO47-d, CERT_C-FIO47-e, CERT_C-FIO47-f, CERT_C-INT31-f, CERT_C-INT34-a, CERT_C-MSC07-b, CERT_C-MSC09-a, CERT_C-MSC12-b, CERT_C-MSC13-a, CERT_C-MSC32-d, CERT_C-POS30-a, CERT_C-STR03-a, CERT_C-STR32-a
SEI CERT C++	CERT_CPP-CTR51-a, CERT_CPP-EXP50-b, CERT_CPP-MSC51-a, CERT_CPP-OOP53-a
Coding Conventions	CODSTA-124_a, CODSTA-124_b, CODSTA-161_f, CODSTA-52, CODSTA-56
Coding Conventions for C++	CODSTA-CPP-87_c
High Integrity C++	HICPP-12_4_4-a, HICPP-14_2_1-a, HICPP-1_2_1-b, HICPP-3_1_1-b, HICPP-4_2_2-a, HICPP-5_1_2-b, HICPP-5_1_2-f, HICPP-5_7_2-a
Initialization	INIT-10
Joint Strike Fighter	JSF-009, JSF-075, JSF-104, JSF-117, JSF-125_a, JSF-135_b, JSF-143_a, JSF-164, JSF-186_b, JSF-198, JSF-204.1_b, JSF-204.1_f
Metrics	METRICS-19, METRICS-29, METRICS-33
MISRA C 1998	MISRA-005, MISRA-038, MISRA-071_b
MISRA C 2004	MISRA2004-12_2_b, MISRA2004-12_2_f, MISRA2004-12_8, MISRA2004-14_1_b, MISRA2004-5_2_b, MISRA2004-8_1_b
MISRA C++ 2008	MISRA2008-0_1_11, MISRA2008-0_1_1_b, MISRA2008-0_1_3_a, MISRA2008-0_1_3_b, MISRA2008-0_3_1_e, MISRA2008-11_0_1, MISRA2008-14_7_3, MISRA2008-2_10_2_b, MISRA2008-2_10_6_c, MISRA2008-5_0_1_b, MISRA2008-5_0_1_f, MISRA2008-5_8_1
MISRA C 2012 (Legacy)	MISRA2012-DIR-4_1_e, MISRA2012-RULE-10_1_f, MISRA2012-RULE-11_1_a, MISRA2012-RULE-11_1_b, MISRA2012-RULE-12_2, MISRA2012-RULE-13_2_b, MISRA2012-RULE-13_2_f, MISRA2012-RULE-17_3, MISRA2012-RULE-1_3_g, MISRA2012-RULE-1_3_k, MISRA2012-RULE-21_17_a, MISRA2012-RULE-22_10, MISRA2012-RULE-22_8, MISRA2012-RULE-22_9, MISRA2012-RULE-2_1_b, MISRA2012-RULE-5_3_b
MISRA C 2012	MISRAC2012-DIR_4_1-e, MISRAC2012-RULE_10_1-f, MISRAC2012-RULE_11_1-a, MISRAC2012-RULE_11_1-b, MISRAC2012-RULE_12_2-a, MISRAC2012-RULE_13_2-b, MISRAC2012-RULE_13_2-f, MISRAC2012-RULE_17_3-a, MISRAC2012-RULE_1_3-g, MISRAC2012-RULE_1_3-k, MISRAC2012-RULE_21_17-a, MISRAC2012-RULE_22_10-a, MISRAC2012-RULE_22_8-a, MISRAC2012-RULE_22_9-a, MISRAC2012-RULE_2_1-b, MISRAC2012-RULE_5_3-b
Object Oriented	OOP-48
Optimization	OPT-02, OPT-03, OPT-05, OPT-06, OPT-14, OPT-31
Possible Bugs	PB-33_b, PB-45, PB-46, PB-47, PB-48, PB-49, PB-50
Template	TEMPL-10

Removed Rules

The following rules have been removed and replaced with new ones to ensure compliance with the new AUTOSAR 18.10 standard:

- AUTOSAR-A14_8_1-a
- AUTOSAR-A18_1_5-a
- AUTOSAR-A2_11_1-b
- AUTOSAR-A2_11_1-c
- AUTOSAR-A2_11_1-d
- AUTOSAR-A2_11_1-e
- AUTOSAR-A2_11_2-a

- AUTOSAR-A2_11_3-a
- AUTOSAR-A2_11_3-b
- AUTOSAR-A2_11_5-a
- AUTOSAR-A2_11_5-b
- AUTOSAR-A2_14_1-a
- AUTOSAR-A2_14_2-a
- AUTOSAR-A2_2_1-a
- AUTOSAR-A2_6_1-a
- AUTOSAR-A2_8_2-a
- AUTOSAR-A2_8_4-a
- AUTOSAR-A2_9_1-a
- AUTOSAR-A7_1_1-b
- AUTOSAR-M0_1_5-a
- AUTOSAR-M14_5_2-a
- AUTOSAR-M14_7_3-a
- AUTOSAR-M14_8_1-a
- AUTOSAR-M2_10_3-a
- AUTOSAR-M2_10_6-a
- AUTOSAR-M2_10_6-b
- AUTOSAR-M2_10_6-c
- AUTOSAR-M2_10_6-d
- AUTOSAR-M5_2_1-a
- AUTOSAR-M7_3_5-a
- AUTOSAR-M8_5_1-a

The following rules that were mapped to PREPROC-24 rule were removed to better adhere to the industry standards:

- AUTOSAR-M16_0_7-b
- MISRA2004-19_11
- MISRA2008-16_0_7
- MISRA2012-RULE-20_9_a
- MISRAC2012-RULE_20_9-a

You can manually enable the PREPROC-24 rule in your test configuration to report the same violations.