MISRA C COMPLIANCE ENFORCEMENT

AUTOMATED SOURCE CODE ANALYSIS TO MAINTAIN COMPLIANCE

SIMPLIFY AND STREAMLINE MISRA C COMPLIANCE

The MISRA C compliance module reports on dataflow problems, software defects, language implementation errors, inconsistencies, dangerous usage and coding standard violations quickly and efficiently. The MISRA C compliance module is an optional add-on for the QA·C static analysis solution, providing enforcement of the MISRA C coding guidelines and to ensure security-related defects and violations to the MISRA C coding guidelines are detected. The MISRA C compliance module provides an extension to the analysis and reporting capabilities of QA·C to directly highlight violations of the MISRA C guidelines, and combines error detection and security best practice with full integration within the PRQA product suite.

The MISRA C compliance module provides an out-of-the-box configuration for QA·C, which eliminates the need to manually configure the tool to enforce MISRA C rules, and includes additional checks to supplement the already extensive suite of QA·C analysis checks. The existing QA·C report templates are also enhanced to allow generation of reports that specifically show the compliance of a code base to the MISRA C standard, to inform internal stakeholders or to use for audit purposes.

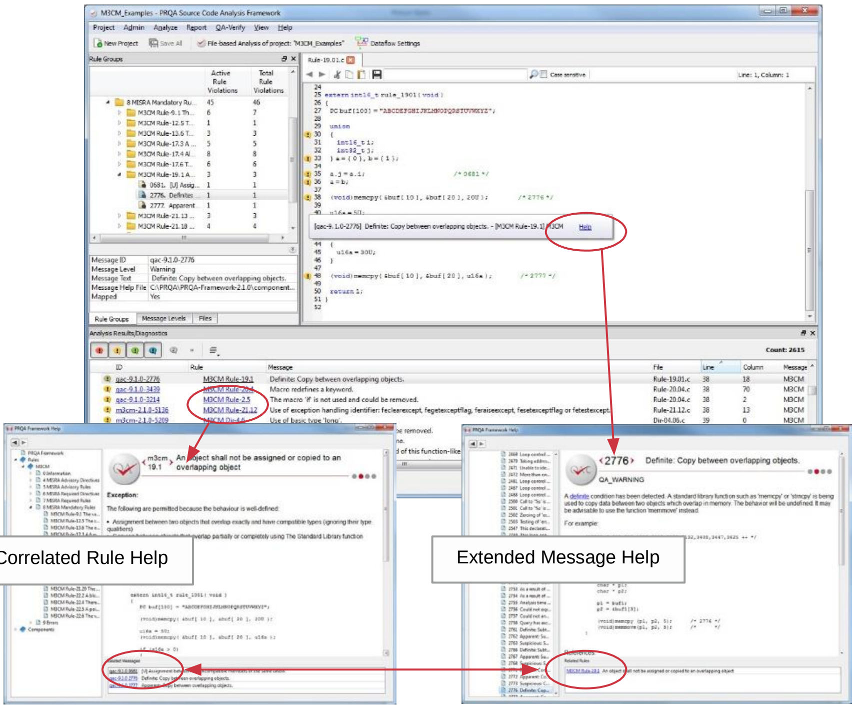
IDENTIFIES WHAT THE PROBLEM IS, EXPLAINS WHY IT'S A PROBLEM AND SHOWS HOW TO FIX IT

The QA·C static analyzer automatically performs in-depth analyses on your source code without executing programs. It checks your software for security vulnerabilities and conformance to MISRA C coding best practices and can be configured to run locally on either desktop or server. QA·C identifies issues which compilers and most developers miss. These include lesser-known issues explicitly stated in the ISO standards and language constructs that, while not classified as incorrect, may result in unpredictable behavior.

Unlike bug catchers or less sophisticated static analyzers, QA·C finds more issues while producing fewer false positives and negatives.

BENEFITS:

- Automatically track, report and demonstrate MISRA C Compliance.
- Continuously inspect source code for conformance to the MISRA C coding guidelines
- Scale to millions of lines of code
- Increase code portability and re-usability
- Give your developers contextual feedback that helps them correct and learn from mistakes
- Reduce bottlenecks caused by manual code review and slow analysis tools and methods
- Analyze your source code without executing programs


M3CM_Examples - PRQA Source Code Analysis Framework
Project Admin Analyze Report QA-Verify View Help
New Project
Save All
File-based Analysis of project: "M3CM_Examples"
Dataflow Settings
Rule Groups
Rule-19.01.c
Case sensitive
Line: 1, Column: 1
Rule Groups
Message Levels
Files
Analysis Results/Diagnostics
Count: 2615
Help

	Active Rule Violations	Total Rule Violations
8 MISRA Mandatory Ru...	45	46
M3CM Rule-9.1 Th...	6	7
M3CM Rule-12.5 T...	1	1
M3CM Rule-13.6 T...	3	3
M3CM Rule-17.3 A ...	5	5
M3CM Rule-17.4 Al...	8	8
M3CM Rule-17.6 T...	6	6
M3CM Rule-19.1 A...	3	3
0681. [U] Assig...	1	1
2776. Definite:	1	1
2777. Apparent	1	1
M3CM Rule-21.13 ...	3	3
M3CM Rule-21.18 ...	4	4

Message ID	qac-9.1.0-2776
Message Level	Warning
Message Text	Definite: Copy between overlapping objects.
Message Help File	C:\PRQA\PRQA-Framework-2.1.0\component...
Mapped	Yes

```
extern int16_t rule_1901( void )
{
  PC buf[100] = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";

  union
  {
    int16_t i;
    int32_t j;
  } a = { 0 }, b = { 1 };

  a.j = a.i;                  /* 0681 */
  a = b;

  (void)memcpy( &buf[ 10 ], &buf[ 20 ], 20U );        /* 2776 */
```

[qac-9.1.0-2776] Definite: Copy between overlapping objects. - [M3CM Rule-19.1] M3CM

```
  {
    u16a = 30U;
  }

  (void)memcpy( &buf[ 10 ], &buf[ 20 ], u16a );       /* 2777 */

  return 1;
}
```

ID	Rule	Message	File	Line	Column	Message
qac-9.1.0-2776	M3CM Rule-19.1	Definite: Copy between overlapping objects.	Rule-19.01.c	38	18	M3CM
qac-9.1.0-3439	M3CM Rule-20.4	Macro redefines a keyword.	Rule-20.04.c	38	70	M3CM
qac-9.1.0-3214	M3CM Rule-2.5	The macro 'if' is not used and could be removed.	Rule-20.04.c	38	2	M3CM
m3cm-2.1.0-5136	M3CM Rule-21.12	Use of exception handling identifier: feclearexcept, fegetexceptflag, feraiseexcept, fesetexceptflag or fetestexcept.	Rule-21.12.c	38	13	M3CM
m3cm-2.1.0-5209	M3CM Dir-4.6	Use of basic type 'long'.	Dir-04.06.c	39	0	M3CM

m3cm 19.1 An object shall not be assigned or copied to an overlapping object

Exception:

The following are permitted because the behaviour is well-defined:

- Assignment between two objects that overlap exactly and have compatible types (ignoring their type qualifiers)

Correlated Rule Help

2776 Definite: Copy between overlapping objects.

QA_WARNING

A definite condition has been detected. A standard library function such as 'memcpy' or 'strncpy' is being used to copy data between two objects which overlap in memory. The behavior will be undefined. It may be advisable to use the function 'memmove' instead.

For example:

Extended Message Help

> *Since we began using the QA·C MISRA Compliance Module, the quality and consistency of our first-generation code has skyrocketed, and our final products have been virtually error-free.*
>
> *Stuart Jobbins, Delphi Diesel*

DON’T JUST FIND BUGS - ENABLE BEST PRACTICE

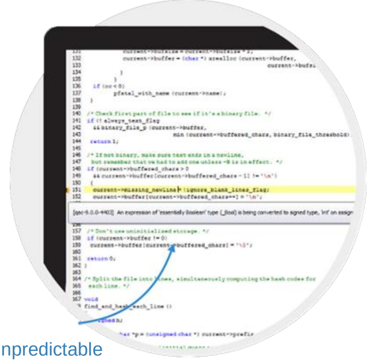
The use of MISRA has expanded well beyond automotive and is used in many industries including aerospace, telecom, medical devices, defense, and railway. The vision of MISRA is to define a subset of the C language in which the opportunity to make mistakes is either removed or reduced. This is a requirement of many standards for the development of safety-related software and can also be used to develop any application with high integrity or high reliability requirements. Developing safety critical and secure code is a significant challenge when it comes to the C languages and may not be a well-understood concept by many developers. With this in mind, the MISRA C coding guidelines attempts to educate developers and drive change rather than just document defective code.

The MISRA C coding guidelines consists of rules and directives, collectively referred to as guidelines. Rules are meant to provide normative requirements for code, whereas directives are meant to provide guidance that, when followed, will improve the safety, reliability, and security of software systems. These rules serve to define a safer subset of the language suitable for any development project where safety, quality, and reliability are a priority. As a result, the MISRA coding guidelines are now accepted worldwide as the bench- mark for developing safety-critical software in C and C++. The MISRA C module helps your organization make informed decisions by finding and reporting on violations of both rules and directives covered within the MISRA C Standard.

KEY FEATURES

ADVANCED DEFECT PREVENTION

Using a proprietary, high-performance C language parser combined with a Deep Flow Dataflow analysis engine, QA·C is able to build an accurate model of the behavior of the software and track the value of variables in the code as they would be at run time. This sophisticated analysis approach maximizes code coverage while minimizing false positives and false negatives and allows QA·C to detect critical defects not reported by compilers or other tools and recog- nize issues caused by dangerous, overly complex and non-portable language usage.



```
            current->buffer = (char *) xrealloc (current->buffer,
                                                 current->bufsi
          }
       }
   if (cc < 0)
         pfatal_with_name (current->name);
   }

  /* Check first part of file to see if it's a binary file. */
  if (! always_text_flag
    && binary_file_p (current->buffer,
                         min (current->buffered_chars, binary_file_threshold)
   return 1;

  /* If not binary, make sure text ends in a newline,
   but remember that we had to add one unless -B is in effect. */
  if (current->buffered_chars > 0
    && current->buffer[current->buffered_chars - 1] != '\n')
   {
    current->missing_newline = !ignore_blank_lines_flag;
    current->buffer[current->buffered_chars++] = '\n';
```

[qac-9.0.0-4403] An expression of 'essentially Boolean' type (_Bool) is being converted to signed type, 'int' on assign

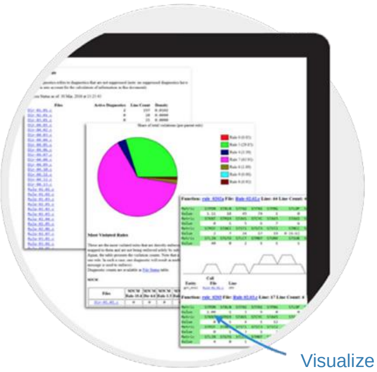
```
  /* Don't use uninitialized storage. */
  if (current->buffer != 0)
   current->buffer[current->buffered_chars] = '\0';

  return 0;
}

/* Split the file into lines, simultaneously computing the hash codes for
 each line. */

void
find_and_hash_each_line ()
```

Identify unpredictable
behaviors others miss

Clearly identify errors
without executing code

ACTIONABLE RESULTS TO COMPLY TO THE MISRA C STANDARD

The MISRA C module clearly identifies must-fix defects and includes a comprehensive knowl- edge base help system that provides detailed guidance with examples to support developers in fixing the issues found in the source code. Because developers get immediate and contex- tual feedback within their development environment, they can make the required changes as they are creating new code or reviewing existing code. In this way, developers build aware- ness of best practice approaches and can quickly form coding habits that are aligned with your organization’s expectations.



Visualize what parts of the
code need the most attention

MONITOR AND CONTINUALLY IMPROVE YOUR CODEBASE WITH CONFIGURABLE REPORTS

The compliance report helps you visualize which areas of your codebase require the most attention to reach a higher level compliance.

The code review report refocuses peer review on discussing design, optimization, and meeting requirements rather than costly manual investigation of code conformance and correctness.

The suppression report provides information on message diagnostics that have been suppressed during analysis.

ANALYSIS OF INDUSTRIAL-SCALE CODE

Automated static analysis using QA·C assists in identifying defects, vulnerabilities, and compliance issues early in the development cycle where they can be fixed faster and at lower cost. QA·C is fast, non-disruptive, easy-to-use, and scales to any size of development environment. As a result, organizations whose products need to perform securely and reliably in mission critical and safety critical environments trust in QA·C to help lower the risk of software failures, improve quality and reduce time-to-market.

EASY TO LEARN AND EASY TO USE

The MISRA C module functions as a plug-in within QA·C's powerful GUI and delivers a contextual drill-down environment linked to a deep knowledge base. QA·C explains why problems it discovers need to be corrected and then provides guidance to help in fixing them.

ADAPTABLE TO FIT EXISTING DEVELOPMENT ENVIRONMENTS

The MISRA C module plugs into QA·C and is easily integrated into existing build systems and continuous integration environments to provide a means to enhance "early and often" testing with automated code analysis that helps to avoid errors that are expensive to fix late in the development cycle. This allows existing code review processes to be accelerated and refocused, thereby helping to increase overall productivity while also improving quality and security of the software. Additionally, the MISRA C module and QA·C can be configured for incremental analysis to ensure that only new changes are analyzed and feedback can be provided quickly.

ROBUST AND FLEXIBLE CODING STANDARD ENFORCEMENT

The MISRA C module is based on the MISRA C coding guidelines , to automate compliance checks for the MISRA C coding guidelines and the generation of the reports and audit documentation required to demonstrate compliance. Separate modules are available to enforce the MISRA C:2004 and MISRA C:2012 (including Amendment 1 security guidelines). QA·C functionality also allows messages to be suppressed at targeted source code locations and these suppressions can be audited to report deviations to enforcement of a standard for compliance purposes.

KEY CHECKS

The MISRA C compliance module helps to avoid constructs in the C language that can reduce code reusability and lead to product failures, functional safety issues and vulnerabilities that attackers can exploit. Each compliance module applies the extensive QA·C message set supplemented by some additional MISRA-specific checks to enforce the coding rules. Documentation is provided describing rule enforcement and message interpretation, and an extensive set of example code is included to aid understanding.

MISRA C places particular focus on defining a safer subset of the C language for any development project where safety, quality, reliability or security are issues of concern.

The MISRA C categories of rules and directives include:

- Undefined and unspecified behavior
- Implementation defined behavior
- Code design
- Language extensions
- Unused code
- Comments
- Character sets and lexical conventions
- Types
- Literals and constants
- Declarations and definitions
- Initialization
- The essential type model
- Identifiers
- Pointer type conversions
- Expressions
- Side effects
- Control statement expressions
- Control flow
- Switch statements
- Functions
- Pointers and arrays
- Overlapping storage
- Preprocessing directives
- Standard Libraries
- Resources

GENERAL FEATURES

- Command line interface (CLI)
- Interactive GUI with message browser
- Online help & knowledge base
  - Usage & implementation contextual message
  - C language
  - MISRA C coding guidelines
- Summary & detailed reports
- IDE integrations

CODE ANALYSIS FEATURES

- 1,700+ selectable messages
- C language-specific parsing engine
- Parses code of any size & complexity
- Handles common language extensions
- Cross module analysis (link time checking)
- Semantic error detection
- Dataflow error detection
- Close name analysis

MESSAGE OUTPUT CONTROL

- Comment based suppression
- Baselining

RESULTS OUTPUT

- Configurable HTML reports
- Standard report types
  - Compliance
  - Code review
  - Suppression
  - Metric data

CODING STANDARD ENFORCEMENT

- Enforces MISRA C rules and directives
  - MISRA C:2004
  - MISRA C:2012
  - MISRA C:2012 Amendment 1
- Rule subsets for legacy code
- Best practice issues
- Naming convention checker
- Layout checker
- Defensive programming - defect avoidance
- Extensible rule base
- Customizable message text
- Deviation support

SGS-TÜV SAAR CERTIFIED

SGS-TÜV Saar has certified QA·C and QA·C++ as "usable in the development of safety related software" for the key safety critical standards, IEC 61508, ISO 26262, EN 50128, IEC 60880 and IEC 62304, enabling our customers to achieve product certifications to these standards more easily and in less time.

SGS TÜV SAAR
FUNKTIONALE SICHERHEIT GEPRÜFT
FUNCTIONAL SAFETY APPROVED

QA Systems and Programming Research Ltd

QA Systems is an authorised reseller of the QA·C / QA·C++, QA·Verify static testing tools and their compliance module add-ons, which are owned by Programming Research Ltd.
QA·C ®, QA·C++ ® and QA·Verify ® are registered trademarks of Programming Research Ltd, These tools and this document are the copyright © 2016 of Programming Research Ltd.
Third party trademarks, logos and trade names appearing in this document are the trademarks and property of their respective owners.

QA·C, QA·C++ and QA·Verify, offer the closest possible examination of C and C++ code. All contain powerful, proprietary parsing engines combined with deep accurate dataflow which deliver high fidelity language analysis and comprehension. They identify problems caused by language usage that is dangerous, overly complex, non-portable or difficult to maintain. Plus, they provide a mechanism for coding standard enforcement.

Contact Us

For further information regarding QA·C, QA·C++ and QA·Verify and compliance module add-ons, please contact QA Systems at **info@qa-systems.com** where appropriate QA Systems will re-direct you to Programming Research Ltd.