MISRA C++ 2023 GUIDELINES EXPLAINED

What are the MISRA C++ 2023 guidelines?

The MISRA C++ guidelines establish a subset of the C++ language that minimizes or completely removes opportunities for errors, making it suitable for use in safety-critical, mission-critical, and high-reliability contexts

Because of its maturity, versatility, and efficiency, C++ is frequently chosen for critical systems, both in embedded and hosted applications. However, no programming language can ensure that the final executable code behaves precisely as the developer intended. C++ has its limitations, including issues related to language definition, misuse, misunderstanding, and runtime error checking. Many safety-related standards, such as ISO 26262, mandate restricting the used programming language to a safe subset, without defining this subset. The safe subset defined by MISRA, incorporated into a robust software development process, mitigates many of these C++ language shortcomings. Additionally, even though a part of MISRA is specific to embedded or real-time software, most of it is relevant regardless of the end-use purpose of the code.

History and background

In 1998, MISRA introduced MISRA C, a comprehensive set of guidelines designed to facilitate the development of safety-related systems using the C programming language, primarily within the automotive industry. The need arose from the perception that C was a desirable programming language for the automotive industry but which raised some concerns as to its suitability for mission-critical applications.

Over time, MISRA C expanded its influence beyond the automotive sector, becoming the dominant international standard for coding guidelines when using C in critical systems. These guidelines gained widespread recognition for meeting the language subset requirements outlined in the 1994 MISRA Development guidelines for vehicle-based software and IEC 61508.

Fast forward ten years, and as C++ popularity increased, it found itself in a situation akin to what C faced years ago. Was it suitable for use within mission-critical applications? Recognizing this gap, MISRA developed a comprehensive set of guidelines specifically tailored for using C++ in critical systems. The result was a set of guidelines similar to those originally designed for C and termed MISRA C++.

This landmark document, titled “MISRA C++ Guidelines for the Use of the C++ Language in Critical Systems,” saw its official publication and launch on June 5, 2008. It targets the C++03 version of the language.

Since its publication in 2008 the MISRA C++ guidelines have expanded their reach beyond automotive into other sectors such as medical technology, transportation, and the finance industry - anywhere the integrity of the code within software is considered mission or safety-critical.

C++ and the industries intended to be served by the guidelines continued to evolve. Without an updated version of MISRA C++, a new set of guidelines emerged in 2017, as part of the AUTOSAR platform that added guidelines targeting C++14 through a development partnership of automotive interested parties. These guidelines were based on the 2008 MISRA C++ guidelines.

In 2019, the MISRA consortium announced the integration of AUTOSAR C++ coding guidelines into one single, updated industry-standard - MISRA C++ 2023.

What do the latest version of the MISRA C++ guidelines bring?

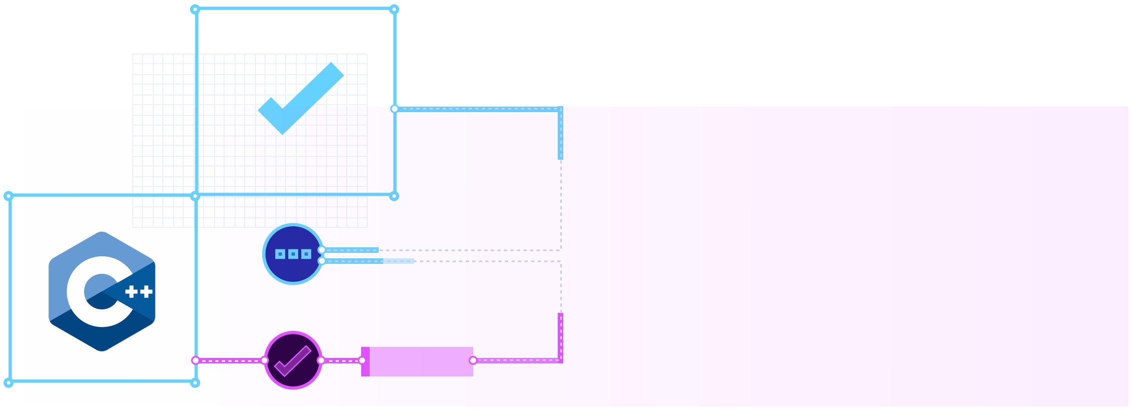
MISRA C++ 2023 builds upon the existing guidelines initially published in 2008 and also integrates the AUTOSAR C++ guidelines into one industry standard for best practice to:

- Ensure it is aligned with modern C++ (C++17) as it continues to evolve.
- Rectify known issues from previous editions.
- Introduce new guidelines supported by solid rationale.
- Remove guidelines lacking sufficient reasoning.
- Enhance specifications and rationale for existing guidelines when appropriate.
- Improve the decidability of guidelines to facilitate processing by static analysis tools.

Adopting the MISRA C++ 2023 guidelines

Developers should adopt MISRA C++ 2023 guidelines at the outset of a project.

However, in cases where a project builds upon existing, proven code, a careful judgment should be made by developers as to the benefits of moving to the new guidelines. Developers should consider the risks of introducing new defects through the process of making the code compliant.



Directives and rules

Each MISRA guideline falls into one of two categories: **directive** or **rule**.

A **directive** is a guideline requiring additional information, often found in design documents or requirements specifications, to check compliance. Static analysis tools may be able to assist with compliance; however, there is a risk of interpretation as to what constitutes non-compliance. There are three directives.

A **rule** is a guideline against which a complete description is provided. This is where static analysis tools can check compliance, as no additional information is required, and in theory, compliance with the rule should not be open to interpretation. There are 175 MISRA C++ 2023 rules.

Classification of rules

Rules are further categorized as **mandatory**, **required**, or **advisory**.

Any code within a project that claims to be MISRA compliant will comply with every **mandatory** guideline. No deviation is permitted. For **required** guidelines, the code must either comply or a justification for formal deviation be produced. **Advisory** guidelines are designed to help improve a project's code quality attributes. However, any code within a project that claims to be MISRA compliant cannot ignore advisory guidelines. Instead, any differences shall be documented with a justification.

The rules are then further classified as **decidable** or **undecidable**. This classification reflects a static analyzer's theoretical ability to find any issue related to that rule and, therefore, determine compliance. For a rule classified as **decidable**, a tool, such as SonarLint, can respond definitively yes or no to the question, "Does this code comply?"

A rule is typically classified as **undecidable** if the ability to detect a violation is dependent upon other factors such as:

- Object values
- Control flow

Undecidable means that no implementation of the rule will ever be certain of discovering all violations. However, depending on the quality of the implementation and tool used, many violations may be reported.

Achieving MISRA C++ 2023 compliance

Various steps will help prepare to achieve MISRA C++ 2023 compliance throughout the software development cycle. There also exist tools that handle the certification of the compliance itself.

Compliance cannot be added to the end of a project as an afterthought. The best practice is for developers to ensure all code is compliant as they create it.

Sonar tools focus on more than compliance with a specific set of guidelines. Instead, Sonar solutions ensure the creation of clean code at every step of the development process. The natural consequence of this is code best positioned for eventual compliance.

Using a tool such as SonarLint in the IDE, which dynamically checks code against specific MISRA C++ 2023 rules and also against many other aspects of code quality, best positions a project for successful MISRA C++ 2023 compliance.

Combined with a powerful solution such as SonarQube integrated into the CI/CD pipeline, this ensures that in-depth code analysis takes place with every change in the code. It aligns teams around a common standard of clean code, including MISRA C++ 2023 specific rules, and provides clear quality gates to best prepare a project for eventual certification.

Summary

MISRA C++ 2023 offers a set of guidelines designed to enhance the reliability and safety of C++ code. It addresses common language pitfalls and challenges, incorporates AUTOSAR C++ to become the de facto guideline, and builds upon the existing MISRA C++ guidelines, making it a valuable resource for developers working on safety-critical and high-reliability systems.

The home of MISRA C++ 2023 is https://misra.org.uk/