What's New in QA-MISRA 26.04?

QA-MISRA 26.04, released in April 2026, focuses on improving analysis performance, deeper code intelligence, enhancing reporting, and increasing flexibility in everyday workflows.

This document provides a concise overview of the significant changes in QA-MISRA 26.04.

Contents

Introduction

QA-MISRA 26.04 is available from April 2026. This update introduces a set of performance, usability, and analysis improvements designed to support modern development environments and increasingly complex codebases.

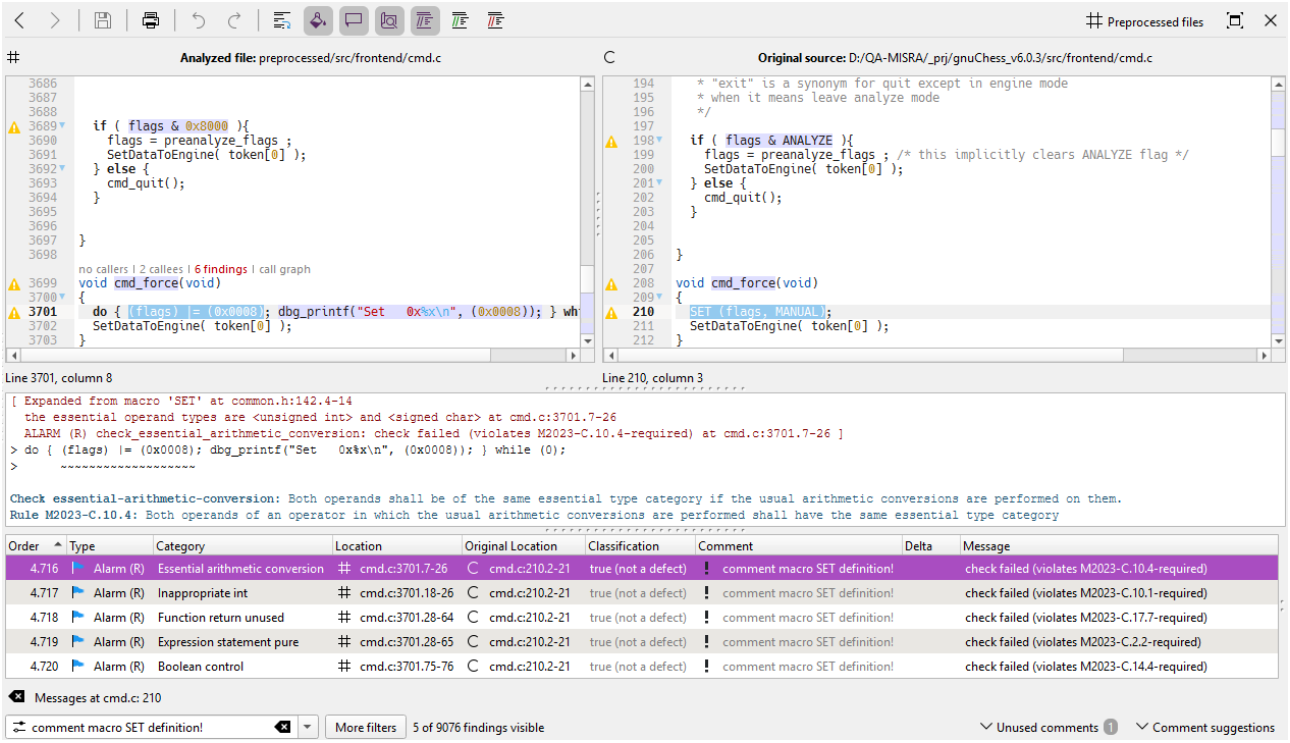
This release focuses on three key areas: faster analysis through parallelisation, improved accuracy in call graph and reporting, and greater flexibility in configuration and annotations. Alongside these, the release includes several optimisations and workflow enhancements to improve the overall user experience.

QA-MISRA 26.04 also contains many other productivity and flexibility enhancements as well as fixes. The full set of changes is documented in the Release Notes, which tracks all changes in QA-MISRA since version 22.04.

The most important changes in this release are highlighted in the sections below.

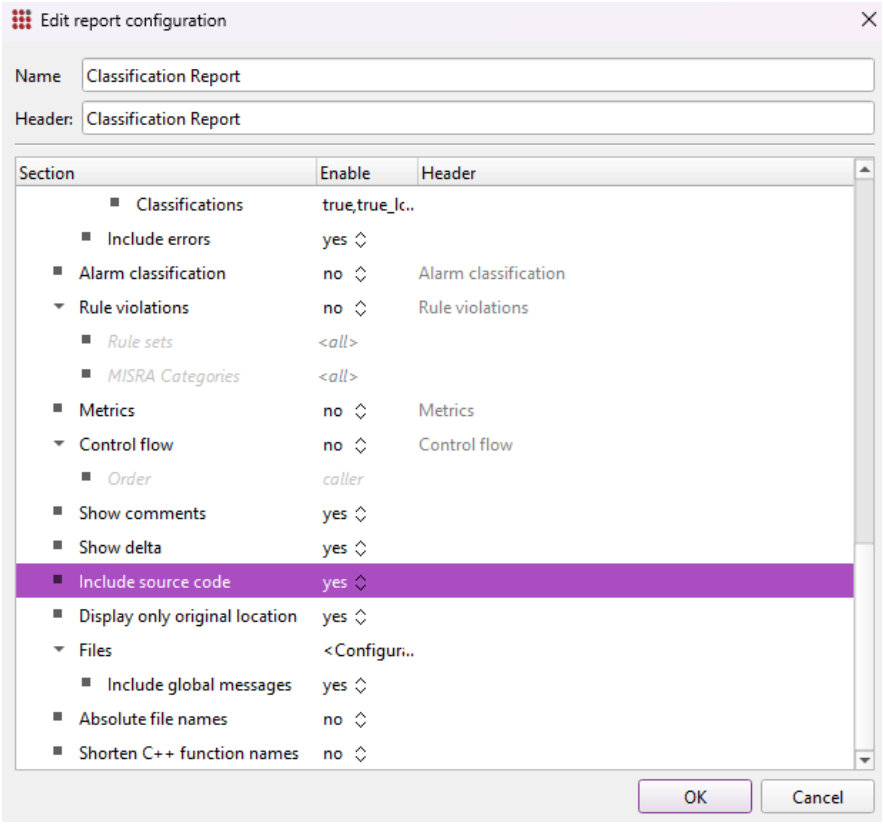
Control Rule Violations Once – Across All Macro Expansions

Managing rule violations in complex codebases just got easier. With the new 'with_expansions' argument, your directives can now extend to all macro expansions. Instead of manually addressing individual findings across every instance a macro is used, you can comment or suppress alarms once – directly at the macro definition level. Combined with a simplified, intuitive format for finding-keys specifications, this update drastically reduces noise and saves valuable development time during QA-MISRA compliance checks.



Improved Source Code Mapping and Reporting

Pinpointing and fixing code violations just got faster. Instead of navigating complex preprocessed files, report files now map findings directly to their original source code locations. By embedding the exact, relevant source code snippets right next to the reported findings, you get instant context without toggling between tools. This dramatically reduces debugging time, simplifies compliance reviews, and helps development teams resolve issues with greater precision.



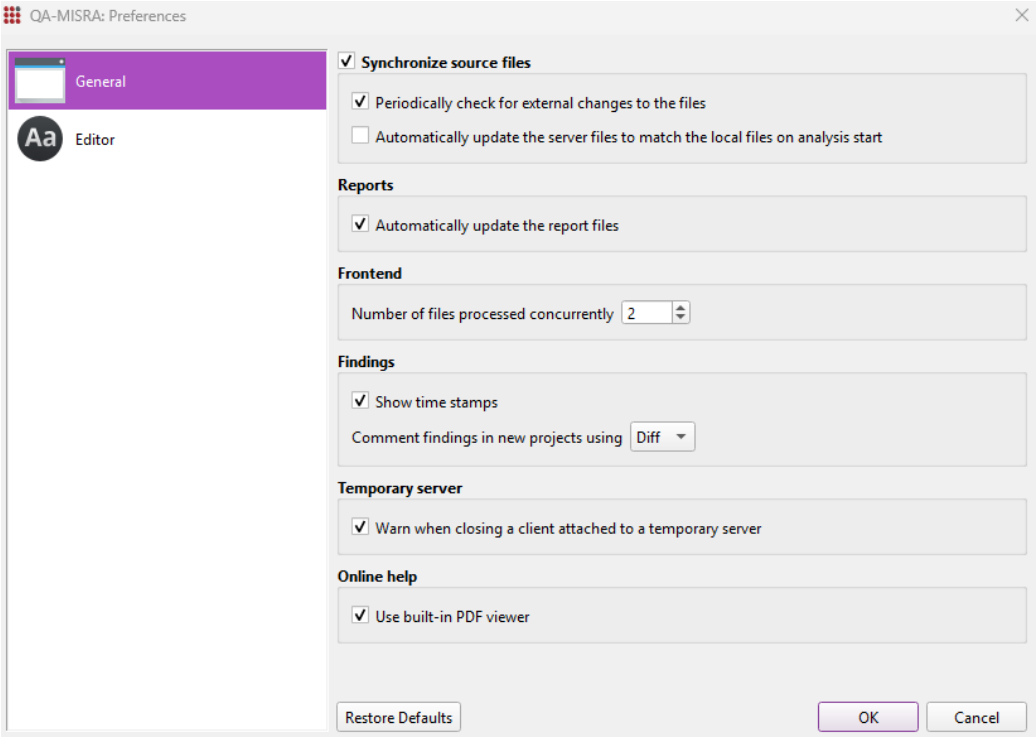
5. Results

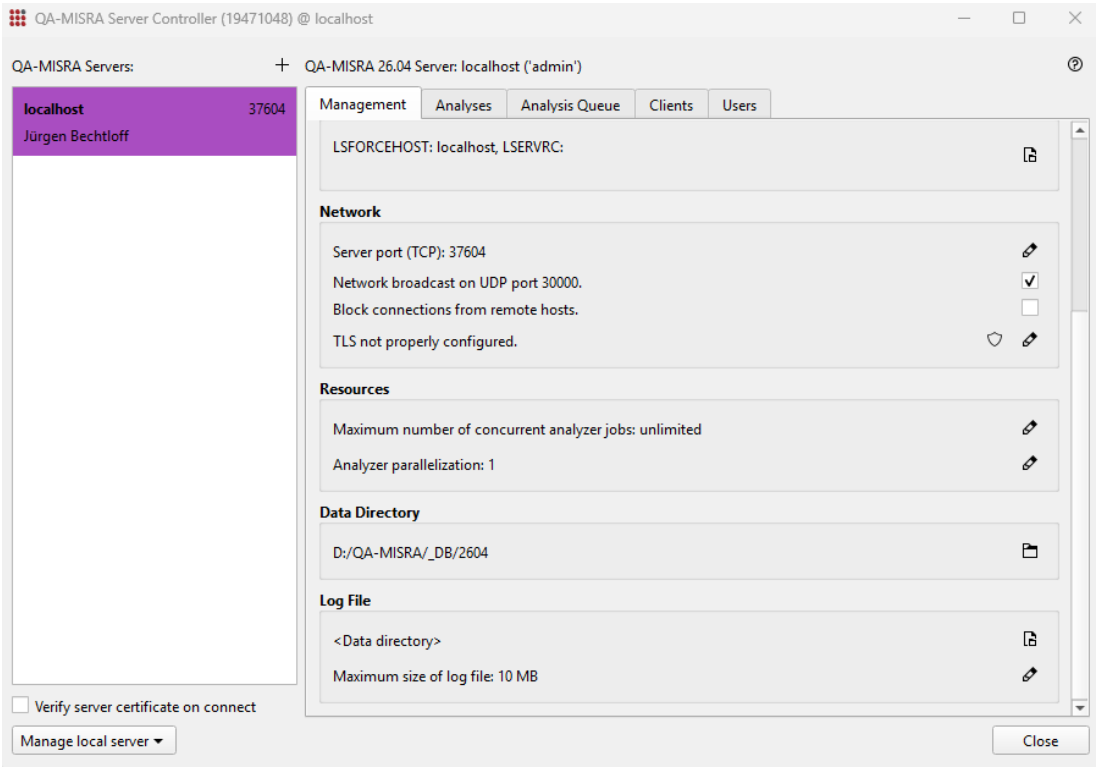
5.1. Classified Findings

Type	Category	Original location	Message
1. Alarm (R)	Function return unused	main.cc:337.1-20	ALARM (R) check_function_return_unused: check failed (violates M2023-CPP.0.1.2-required) > SET (flags, XBOARD); > ~~~~~~~~~~~~~~~~~~~~~
2. Alarm (R)	Function return unused	main.cc:339.1-18	ALARM (R) check_function_return_unused: check failed (violates M2023-CPP.0.1.2-required) > SET (flags, POST); > ~~~~~~~~~~~~~~~~~~
3. Alarm (R)	Function return unused	main.cc:344.4-23	ALARM (R) check_function_return_unused: check failed (violates M2023-CPP.0.1.2-required) > SET (flags, XBOARD); > ~~~~~~~~~~~~~~~~~~~~~
4. Alarm (R)	Function return unused	main.cc:346.4-20	ALARM (R) check_function_return_unused: check failed (violates M2023-CPP.0.1.2-required) > SET (flags, UCI); > ~~~~~~~~~~~~~~~~~
5. Alarm (R)	Function return unused	main.cc:348.4-21	ALARM (R) check_function_return_unused: check failed (violates M2023-CPP.0.1.2-required) > SET (flags, POST); > ~~~~~~~~~~~~~~~~~~
6. Alarm (R)	Function return unused	main.cc:351.4-23	ALARM (R) check_function_return_unused: check failed (violates M2023-CPP.0.1.2-required) > SET (flags, MANUAL); > ~~~~~~~~~~~~~~~~~~~~~

Faster Analysis with Parallelization of rule checking

A major enhancement in this release is the ability to parallelise server-side parsing and rule checking for frontend-level C rules. You can now distribute rule checking across multiple threads of execution for drastically faster analysis times.

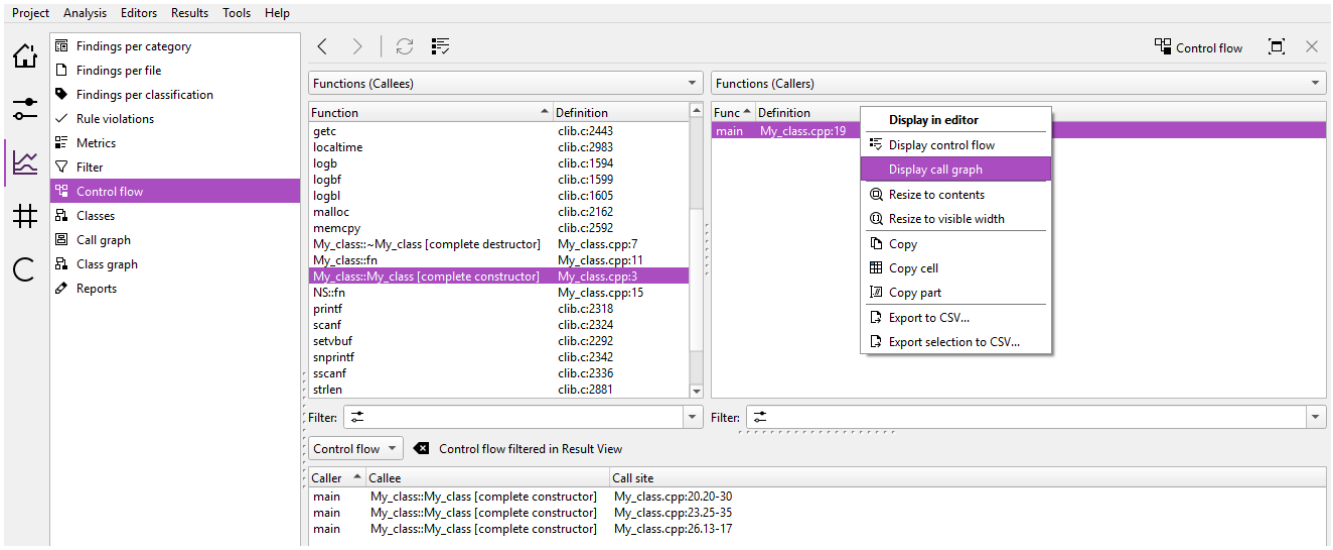


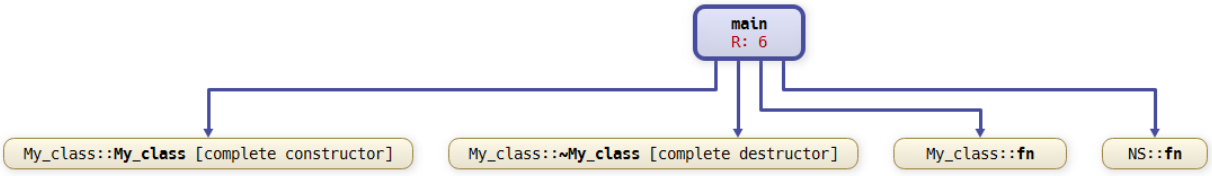


Deeper Code Intelligence

Constructor and Destructor Calls are now included in the call graphs and metric calculations

Constructor and destructor calls are now taken into account for control flow information, static call graph visualization, and threshold checks for relevant metrics such as CALLS, CALLING, and RPATH. This leads to more reliable metrics, better-informed decisions, and improved detection of complex dependencies—helping you optimize code quality with confidence.

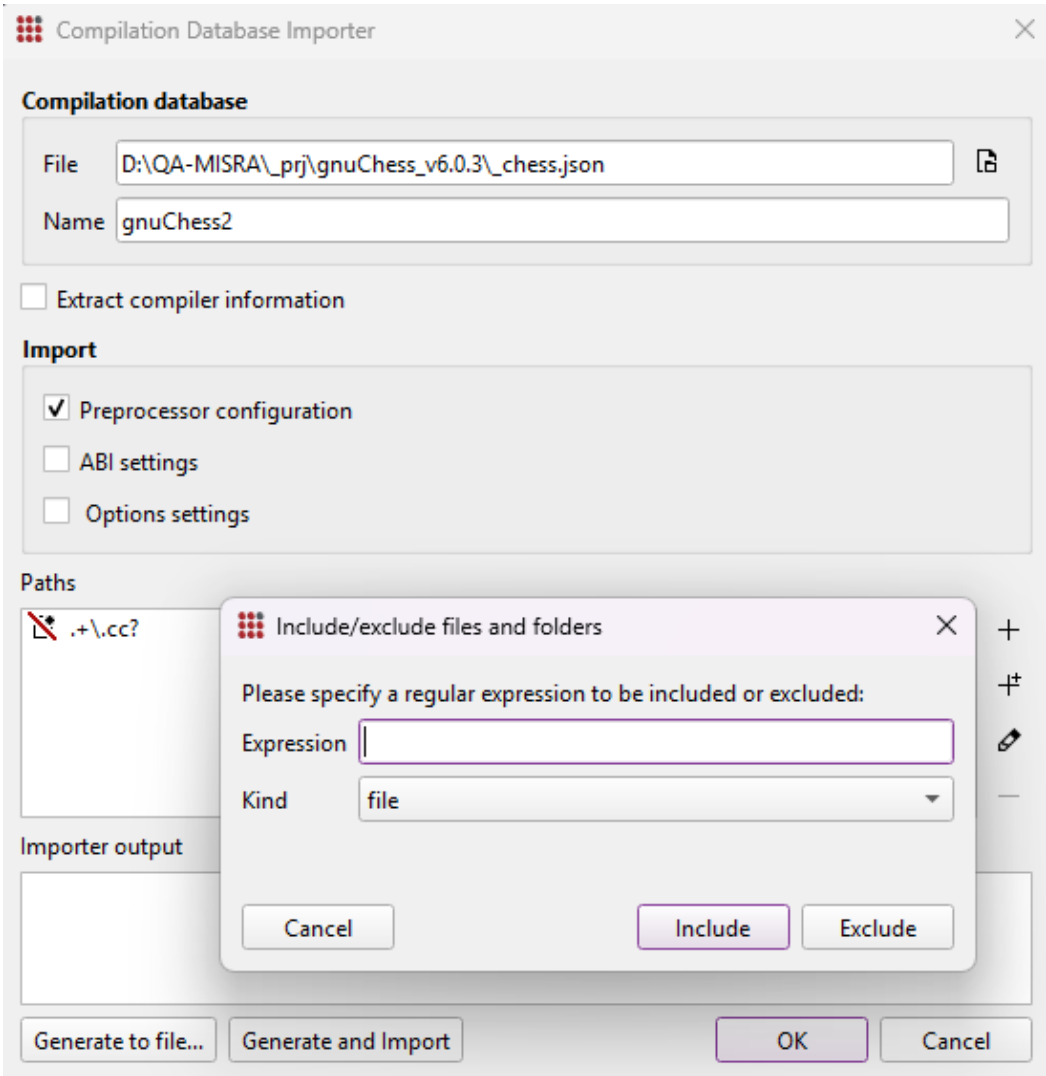




Streamline Your Analysis: Enhanced JSON Compilation Database Import

DAX takes JSON compilation database integration to the next level with powerful, user-defined filtering capabilities. Easily control your analysis scope by including or excluding specific files and directories through intuitive blacklists, whitelists, or flexible regular expressions. Focus only on the data that matters while keeping your workflow clean and efficient.

Seamless integration allows imported data to be enriched with additional attributes directly within the same DAX file, while still supporting standalone JSON import steps when needed. With the enhanced --compilation-database-to-dax option now accepting DAX files as arguments, you gain unmatched flexibility to tailor your import and analysis process to your exact requirements.



New integrations with GitLab CI/CD and Arm Keil Studio

The GitLab integration is available as a component in the GitLab CI/CD catalogue (https://gitlab.com/absint/components), enabling teams to include QA-MISRA static analysis directly in their GitLab pipelines with minimal configuration.

The Arm Keil Studio integration extends QA-MISRA's existing Keil support to the modern Keil Studio environment, making it easy to run compliance checks from within the IDE used by many embedded developers.