**MISRA Compliance Guidelines**

**VERSION: 1.0**

**DATE: 29-Jul-2014**

**Prepared By:**

**Nexteer Automotive,**

**Saginaw, MI, USA**

**Revision History**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Sl. No.** | **Description** | **Author** | **Version** | **Date** | **Approved By** |
| 1 | Initial Version | Spandana Balani | 1.0 | 28-Jul-2014 | SPRB |

**Table of Contents**

[1. Abbreviations And Acronyms 5](#_Toc394389369)

[2. References 6](#_Toc394389370)

[3. Introduction 7](#_Toc394389371)

[3.1. Purpose 7](#_Toc394389372)

[3.2. Scope 7](#_Toc394389373)

[3.3. Document Conventions and Definitions 7](#_Toc394389374)

[4. MISRA Guidelines 8](#_Toc394389375)

[4.1. Software modules classification 8](#_Toc394389376)

[4.2. MISRA C Compliance and Deviations 8](#_Toc394389377)

[4.3. Suppression Technique 8](#_Toc394389378)

[4.4. Single Line Comment 9](#_Toc394389379)

[4.5. Block Level 9](#_Toc394389380)

[4.6. File Level Comment 9](#_Toc394389381)

[5. ISO 26262 Compliance 10](#_Toc394389382)

[5.1. Coverage of ISO26262 Part 6 Table 1 — Topics to be covered by modelling and coding guidelines 10](#_Toc394389383)

[5.2. Coverage of ISO26262 Part 6 Table 8 -- Design principles for software unit design and implementation 10](#_Toc394389384)

[6. Appendix 11](#_Toc394389385)

[6.1. QAC Project Creation and Analysis for component 11](#_Toc394389386)

[6.2. Guidelines to create component wise QAC project with standard folder structure 11](#_Toc394389387)

[6.3. Manual creation of QAC project 11](#_Toc394389388)

[6.4. QAC Project Creation and Analysis for Integration project 17](#_Toc394389389)

[6.5. Instructions to change QAC personality files 18](#_Toc394389390)

# Abbreviations And Acronyms

|  |  |
| --- | --- |
| Abbreviation | Description |
| MISRA | Motor Industry Software Reliability Association |
| ISO | International Standards Organization |

# References

|  |  |  |
| --- | --- | --- |
| Sr. No. | Title | Version |
| 1 | MISRA-C:2004, *Guidelines for the use of the C language in critical systems*, ISBN 978-0-9524156-2-6, MIRA, October 2004 | 2004 |
| 2 | Software Design and Coding Standards | 2014 |
| 3 | Software Naming Conventions | 1.0 |
| 4 | ISO/IEC 9899:1999(E) – International Standard /Programming languages – C |  |
| 5 | ISO 26262 -6 Road vehicles - Functional safety - Part 6: Product development at the software level | 2011-11-15 |
| 6 | QAC-8.1 Users Guide | 2012 |

# Introduction

## Purpose

The purpose of this document is to identify Nexteer Automotive’s Electrical Steering Systems software compliance to the MISRA C 2004 standards.

## Scope

All production intent software developed by Nexteer Automotive’s Electrical Steering Systems product engineering group [ESG] shall comply with the MISRA C coding Standards and any exceptions should be documented.

As required by MISRA C: 2004 (section 4.4), this document covers the following:

* A compliance matrix that demonstrates how each rule is enforced
* Analysis to ensure all C code in the product is compliant with the MISRA C rules or deviations documented
* List all instances where rules are not being followed, including the deviation process
* Configuration and use of QAC version 8.x as the Static Code Analysis tool

## Document Conventions and Definitions

The following definitions are used throughout this document:

* **Shall**: indicates a mandatory requirement without exception in compliance.
* **Should**: indicates a mandatory requirement; exceptions allowed only with documented justification.
* **May**: indicates an optional action.

# MISRA Guidelines

## Software modules classification

Nexteer’s embedded projects contain software that is developed internally, generated through tools and 3rd party modules that should not require changes. Each type of software module, as defined in Ref [2], may require a different set of rules and thus needs to be categorized into appropriate group for reporting. Refer to the program’s Software Development Plan for the classification of steering application source modules.

For this revision of the document:

* Type 1A files comply according to section 5 (MISRA C Compliance and Deviations) of this document
* Type 1B files (the sections that are modified by Nexteer) are treated as Type 1A
* Type 1B files (the sections that are not modified by Nexteer) are not considered for the compliance
* Type 2 files are not considered for the compliance
* Type 3 files are not considered for the compliance

## MISRA C Compliance and Deviations

The embedded table “MISRA Compliance Guideline” associates MISRA C 2004 rules with QAC messages. The QAC warning messages are summarized in the table such that each warning is either turned ON or OFF at the tool level (Message personality). For the warning messages that are turned ON, this table lists which of those can be deviated. The table also explains the suppression strategy for the warnings that can be deviated.



MISRA C 2004 rules 1.3, 1.4, 1.5, 2.4, 3.2, 3.3, 3.5, 3.6, and 20.3 are not checked by QAC and require manual verification. Refer to the comments in the embedded table.

## Suppression Technique

MISRA compliance guideline lists the suppression strategy for the allowed deviations for all type 1A files. According to Ref [2] Rule [N45.1], for each MISRA rule violation where QAC warning suppression is required, the MISRA rule violation comment shall appear after the QAC suppression comment.

According to Ref [2] Rule [N44.1], for each MISRA rule violation, a comment at the corresponding code line shall give the justification for the deviation and shall be formatted as follows:

/\* MISRA-C:2004 Rule XX.X: reason why the rule could not be followed or why deviation is allowed in this instance \*/

Below are the allowed suppression strategies for QAC warning messages with allowed deviation:

## Single Line Comment

Format: /\* PRQA S <QAC warning #> \*/

Details: The rule is suppressed at the occurrence of the violation.

Example:

/\* Metrics start hook does nothing in production build. \*/

Metrics\_TaskStart(D\_SPINXT1\_CNT\_U08); /\* PRQA S 3112 \*/

/\*

\* MISRA-C: 2004 Rule 14.2: These deviations are required for metrics hooks. These statements are intended to have

\* no effect on production code.

\*/

## Block Level

Format: /\* PRQA S <QAC warning #> <Number of Lines> \*/

Details: The rule is suppressed for the block due to the frequency of usage throughout the block. Annotation at each deviation unnecessarily clutters the file.

Example:

/\* PRQA S 310 10

\* MISRA-C: 2004 Rule 11.4: ADD RATIONALE\*/

## File Level Comment

Format: /\* PRQA S <QAC warning #> EOF \*/

Details: When a rule is violated 3 or more times within a file for the same reason, e.g. pointer casts in files doing register access, one rule violation comment (refer section 6.4.8.1 of Ref [2] for placement – comment should be placed after File Comment Block and Include Statements) may be used to give justification for that deviation for the entire file.

Example:

/\* PRQA S 303 EOF

\* MISRA-C: 2004 Rule 11.3: This deviation is required for register access. The rule is suppressed for the entire file \*/

# ISO 26262 Compliance

## Coverage of ISO26262 Part 6 Table 1 — Topics to be covered by modelling and coding guidelines

| ISO26262-6 Table 1 Topic | | Coverage |
| --- | --- | --- |
| 1a | Enforcement of low complexity | Ref [2] Rule [N47], checked by QAC cyclomatic complexity metric |
| 1b | Use of language subsets | MISRA C 2004 Rule 1.1, 2.1, checked by QAC |
| 1c | Enforcement of strong typing | MISRA C 2004 Rule 6.1, 6.2, 6.3, 6.4 6.5, checked by QAC |
| 1d | Use of defensive implementation techniques | MISRA C 2004 Rule 3.1; Ref [2] Rule[N65], [N66], [N67], [N70] |
| 1e | Use of established design principles | Use of MISRA C as per Ref[6]; Ref [2] See section 7.2, 5.6 |
| 1f | Use of unambiguous graphical representation | Ref [2] section7 .1 |
| 1g | Use of style guides | Ref [2] Rules in sections 6.4.9, 6.4.11, and 6.4.12 |
| 1h | Use of naming conventions | MISRA C 2004 Rule 3.2, 5.1, 5.2, 5.3, 5.4, 5.5, 5.6, 5.7 Ref [3] and Ref [2] rules in section 6.4.1 |

## Coverage of ISO26262 Part 6 Table 8 -- Design principles for software unit design and implementation

| ISO26262-6 Table 8 Method | | Coverage |
| --- | --- | --- |
| 1a | One entry and one exit point in subprograms and functions | MISRA C 2004 14.7, 16.8, checked by QAC |
| 1b | No dynamic objects or variables, or else online test during their creation | MISRA C 2004 21.1, checked by QAC |
| 1c | Initialization of variables | For temporary variables, MISRA C 2004 1.1, checked by QAC. RTE globals initialized by RTE. Declared globals and module-specific variables covered by rules in section 6.4.6 of Ref [2]. |
| 1d | No multiple use of variable names | Ref [2] rules in section 6.4.1; also MISRA C 2004 5.1, 5.2, 5.3, 5.4, 5.5, checked by QAC. |
| 1e | Avoid global variables or else justify their usage | Ref [2] Rule [N9] and [N10] |
| 1f | Limited use of pointers | MISRA c 2004 Rules under Pointers 17.1, 17.2, 17.3; Ref [2] Rules in section 6.4.13 |
| 1g | No implicit type conversions | MISRA C 2004 Rules under Arithmetic type conversion 10, checked by QAC |
| 1h | No hidden data flow or control flow | Ref [2] Rules in section 6.4.13, and naming conventions for function-like macros in Ref [3]. |
| 1i | No unconditional jumps | MISRA C 2004, 15.2 checked by QAC |
| 1j | No recursions | MISRA C 2004, 16.2 checked by QAC |

# Appendix

## QAC Project Creation and Analysis for component

A QAC project should exist for every SWC. It should be placed in the “tools” folder of the SWC. Note that the QAC project relies on the component having subprojects. A template for the QAC project file is attached below. To adapt the QAC project file to a specific SWC, the file must be edited in a text editor and “##SWC\_NAME##” needs to be replaced with the name of the c file that is generated from Davinci Developer. The projects shall reference all files through relative paths to avoid any project changes when a developer starts a working area on their computer. Refer section 8.2 for the Guidelines to create QAC project.



## Guidelines to create component wise QAC project with standard folder structure

1. The QAC project file should be placed in the component folder with a name QAC.prj” as shown below.

..\<Root folder>\<Component Name>\tools\QAC\QAC.Prj

1. QAC project should use the message personality and compiler personality files from the existing QAC project.

..\<Root folder>\QAC\m2cmMessage.p\_s

..\<Root folder>\QAC\m2cm.p\_c

1. A dedicated analyzer personality file should be created for each individual QAC project and should be placed in:

..\<Root folder>\<Component Name>\tools\QAC\m2cmAnalyser.p\_a

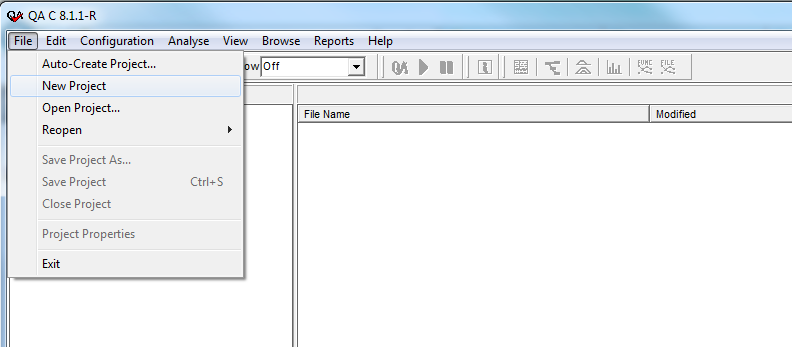
(The analyzer personality file shall reference all includes through relative paths to avoid any project changes when a developer starts a working area on their computer)

1. QAC Output should be placed in the “QAC Output” folder

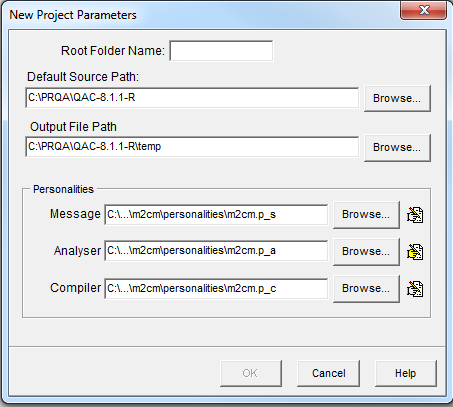
..\<Root folder>\<Component Name>\doc\QAC\_Results

## Manual creation of QAC project

1. Get the appropriate component project from Synergy to your local machine (example below uses DigHwTrqSENT)
2. Open QAC and click on File>>New Project



1. After clicking the ‘New Project’ a New Project Parameters window will pop up as shown

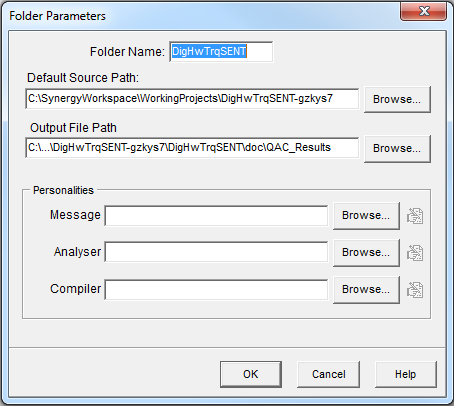


1. Fill in the details as mentioned below

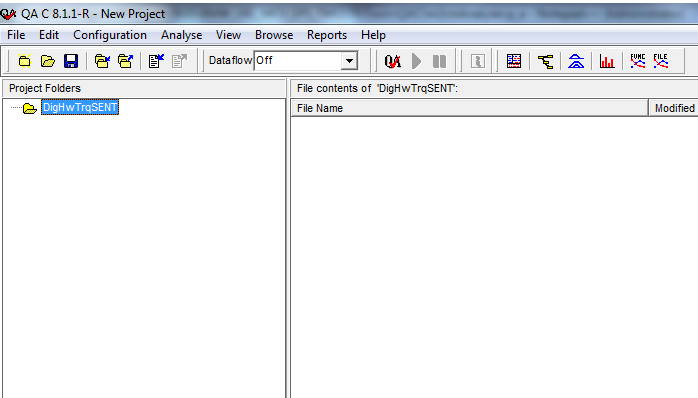
Root Folder Name: <Component Name>

Default Source Path: Point to local work area of the synergy project downloaded from step 1.

Select Output file path: ..\<Root folder>\<Component Name>\doc\QAC\_Results



1. After clicking okay, a <Component Name> folder will be visible in the Project Folders pane as shown below.



1. When you right click on the folder and click Add Folder, New Folder Parameters windows pop up.

Fill in the details as mentioned below

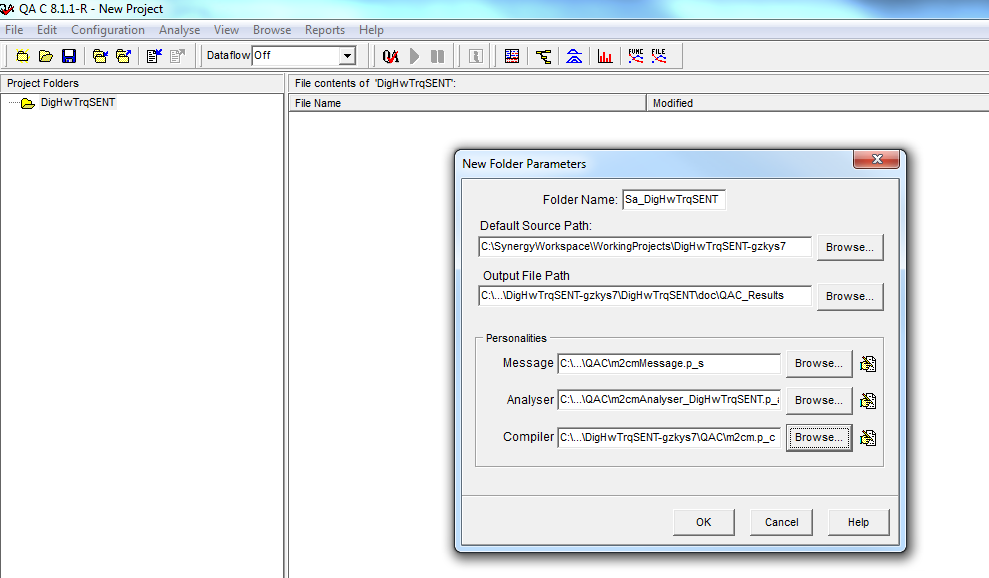
Root Folder Name: <Component Name>

Default Source Path: Point to local work area of the synergy project downloaded from step 1.

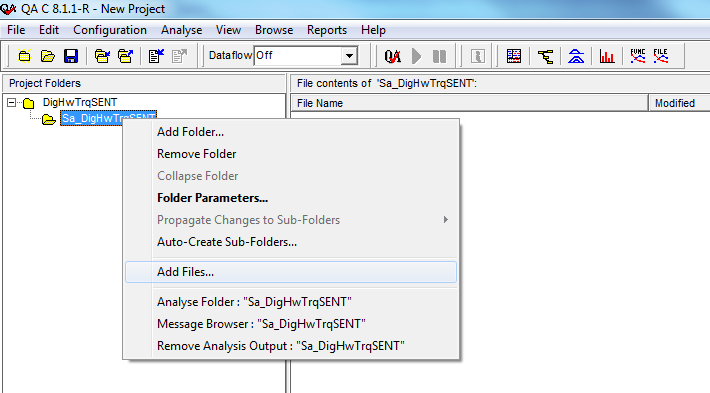
Select Output file path: ..\<Root folder>\<Component Name>\doc\QAC\_Results

Personalities

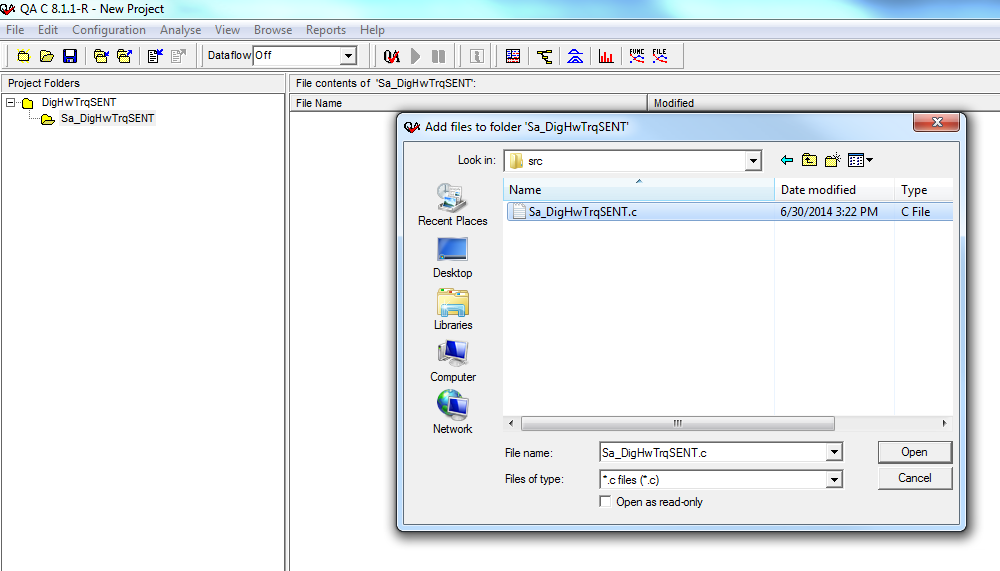
1. Message: ..\<Root folder>\QAC\m2cmMessage.p\_s
2. Analyzer: ..\<Root folder>\<Component Name>\tools\QAC\m2cmAnalyser.p\_a
3. Compiler: ..\<Root folder>\QAC\m2cm.p\_c



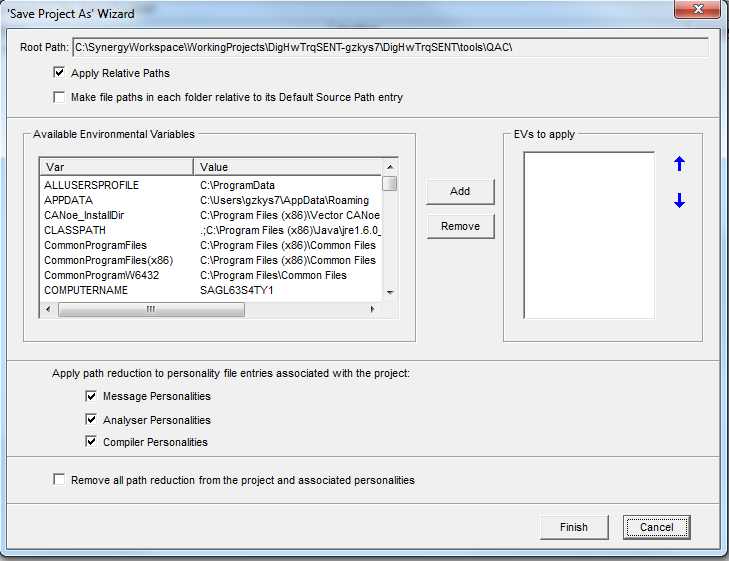
1. Now right click on the sub folder and click on Add Files.



1. Select the C files from <Component Name>\src and click **‘Open’**.



1. Now save the project at <Component Name>\tools\QAC with a name as QAC.prj.
2. After clicking **‘Save’** a new ‘Save Project As’ Wizard will pop up



Select ‘Make file paths in each folder relative to its Default Source Path entry’ and click **‘Finish’** in your new QAC project in set up.

## QAC Project Creation and Analysis for Integration project

1. The QAC Project file for the customers is stored in respective project folders – Application – Version – Source Files - zip folder at [go up a level](http://eroom1.nexteer.com/eRoom/erooms8/EPSSecureLibrary)http://eroom1.Nexteer.com/eRoomData/dot.gif[My eRooms](http://eroom1.Nexteer.com/eRoom) > [EPS Secure Library](http://eroom1.Nexteer.com/eRoom/erooms8/EPSSecureLibrary) > EA 3.x.
2. Copy the project for analysis to your local machine and open the QAC prj file in a text editor to add the source paths
3. Open the analyzer personality file in text editor to add include paths

(The analyzer personality file shall reference all includes through relative paths to avoid any project changes when a developer starts a working area on their computer)

1. Open the QAC.prj file and run analysis - all files should queue and say completed after analysis finishes
2. After the analysis is finished the warning messages can be viewed from Message Browser.

## Instructions to change QAC personality files

1. Copy the project for analysis to your local machine
2. Open QAC and project file
3. Right click on your project
4. Click on Folder Parameters
5. Click on Browse Button next to the personality files and choose the correct message, analyzer and compiler personalities. Note that the message and analyzer personality files are different for different customers but the compiler personality is common.
6. Right click on your top level project within QAC
7. Propagate changes to sub folders 🡪Message Personality
8. Propagate changes to sub folders 🡪Analyser Personality
9. Propagate changes to sub folders 🡪Compiler Personality

Note – It is important to propagate changes to subfolders every time any of the personality files get updated.