PRQA
Programming Research

COMPONENT MANUAL

M2CM 3.3.0

March, 2016

IMPORTANT NOTICE

DISCLAIMER OF WARRANTY

The staff of Programming Research Ltd. have taken due care in preparing this document which is believed to be accurate at the time of printing. However, no liability can be accepted for errors or omissions nor should this document be considered as an expressed or implied warranty that the products described perform as specified within.

COPYRIGHT NOTICE

TRADEMARKS

PRQA, the PRQA logo , QA·C, QA·C++ and High Integrity C++ (HIC++) are trademarks of *Programming Research* Ltd.
"MISRA", "MISRA C" and "MISRA C++" are registered trademarks of MIRA Limited, held on behalf of the MISRA Consortium.
Yices is a registered trademark of SRI International.
Windows is a registered trademark of Microsoft Corporation.

CONTACTING PROGRAMMING RESEARCH LTD

For technical support, contact your nearest Programming Research Ltd authorized distributor or you can contact Programming Research's head office:

by telephone on +44 (0) 1932 888 080
by fax on +44 (0) 1932 888 081
or by webpage: www.programmingresearch.com/services/contact-support/

Contents

List of Figures

List of Tables

1 Introduction

1.1 MISRA C

The original MISRA C Guidelines were published in April 1998 by the Motor Industry Research Association in a document entitled "Guidelines For The Use Of The C Language In Vehicle Based Software". In 2004 an updated version of this document was published under the title "MISRA C:2004 Guidelines for the use of the C language in critical systems". In July 2007 a Technical Corrigendum was issued to address a number of issues which required correction or clarification. This was incorporated into the base document when MISRA C2 was reprinted in June 2008. Most recently in 2013, an updated version of the MISRA C Guidelines was published under the title "MISRA C:2012 Guidelines for the use of the C language in critical systems". There are therefore three versions of the guidelines and these are now referred to as MISRA C:1998 (MISRA C1), MISRA C:2004 (MISRA C2) and MISRA C:2012 (MISRA C3).

Although derived from MISRA C:1998, MISRA C:2004 contains a number of additional requirements and a few significant changes. Code that is compliant with MISRA C:1998 will not necessarily be compliant with MISRA C:2004 and code that is compliant with MISRA C:2004 will not necessarily be compliant with MISRA C:1998.

1.2 The MISRA C:2004 Compliance Module

This document provides an introduction to the MISRA C:2004 Compliance Module (M2CM). The M2CM module is designed to enforce compliance with the MISRA C:2004 programming standard in conjunction with QA·C.

QA·C is a software tool that performs static analysis on C source code. Invariably a process of configuration is required in order to introduce QA·C effectively to a particular software development environment. It is important to have QA·C configured so as to restrict the output of warning messages to those that are of particular concern. M2CM is one such configuration, it configures QA·C to enforce compliance with the MISRA C:2004 coding rules.

QA·C is able to detect most of the statically detectable conditions identified in the MISRA C:2004 rules (as well as many others). M2CM configures QA·C to identify issues which are specific to those rules, and provides a cross-reference between the standard QA·C warning message(s) and the corresponding MISRA C:2004 rule. Also available is an extended HTML description of each rule.

For some MISRA C:2004 coding standard guidelines which are not enforceable directly by QA·C, additional checks are provided through a secondary-analysis program. These

checks are programmed using the analysis output files (.met files) generated by the QA·C "primary analysis" process.

Some MISRA C:2004 rules, as noted in Paragraph 5.6 of the guidelines, impose constraints on the complete project rather than the code in a single translation unit. These requirements are enforced by the Cross-Module Analysis capability of QA·C.

1.3 Installation

For details of installation please refer to the PRQA Framework Installation Notes.

Please check the PRQA Framework Release Notes for the version of M2CM that is compatible with the installed version of QA·C.

2 M2CM Components

The M2CM compliance module introduces a number of additional components to your QA·C installation which provide the following features:

- A subset of the QA·C messages appropriate to enforcement of the MISRA C:2004 rules.
- A new message structure reflecting the MISRA C:2004 rules.
- Secondary-analysis checks to enforce a number of the MISRA C:2004 rules.

2.1 Dataflow

Dataflow analysis is required for the enforcement of a number of MISRA rules. An appropriate configuration for dataflow analysis will depend on factors such as the hardware environment and the complexity of the code. These issues are discussed in the dataflow reference manual. There are no configuration requirements specific to M2CM enforcement.

3 Using M2CM

3.1 Rules and Messages

MISRA C:2004 defines 142 rules grouped in 21 sections, for example Environment, Language Extensions, Documentation, Character Sets, Identifiers. Each rule is classified as either Required or Advisory.

QA·C messages are arranged in 10 levels. In M2CM the following levels are used:

Level 0: **Information**
Level 4: **MISRA Rules**

Level 0 contains informational messages, sub-messages and error recovery messages.

QA·C includes a large number of new messages which are designed to supersede existing messages particularly in the area of arithmetic type conversions. These new messages have been implemented to provide improved enforcement of MISRA C:2004 and to reduce the incidence of false positive messages. To provide continuity to users of previous versions of M2CM, this version does not make use of most of these new messages.

Level 4 contains a message group corresponding to each MISRA C:2004 rule. A rule may be enforced by QA·C (primary analysis), Dataflow analysis, CMA (Cross-Module Analysis), Secondary-Analysis or by a combination of these 4 elements.

3.2 Mapping Messages to MISRA Rules

Some analysis messages are associated with more than one MISRA Rule. This occurs when:

- The requirements of 2 MISRA rules overlap. For example, any violation of Rule 16.6 ("The number of arguments passed to a function shall match the number of parameters") will also violate either Rule 1.1 or Rule 1.2., or
- The granularity of the condition identified by an analysis message does not coincide uniquely with a single MISRA Rule. For example, message 2212 ("Body of control statement is not enclosed within braces.") is mapped to both Rule 14.8 (The statement forming the body of a switch, while, do ... while or for statement shall be a compound statement) and Rule 14.9 (An if (expression) construct shall be followed by a compound statement. The else keyword shall be followed by either a compound statement, or another if statement).

The rule or rules with which each message is associated will always be reported in the "reference text". However, in the situations where a message is associated with more than

one rule, it will only be specifically *mapped* to one of the rules - the primary rule. Where this situation occurs, any statistics relating to the number of violations of any particular rule may be slightly misleading.

It is also important to note that if a message which is mapped to more than one rule is suppressed, it will be suppressed for all the rules with which it is associated.

4 Global Checks

Cross-Module Analysis checks are required to fully implement a compliance modules rule set. This analysis is normally only of significant value when the code from a complete project is available for analysis.

For details of configuration and use of Cross-Module Analysis, please refer to the RCMA Manual.

5 Rule Cross Reference

MISRA-C:2004 Rule 1.1 (Required)
All code shall conform to ISO/IEC 9899:1990 C programming language, ISO 9899, amended and corrected by ISO/IEC 9899/COR1:1995, ISO/IEC 9899/AMD1:1995, and ISO/IEC 9899/COR2: 1996192
QA·C Messages: 0180, 0232, 0233, 0240, 0241, 0244, 0246, 0261, 0320, 0321, 0322, 0338, 0410, 0422, 0423, 0426, 0427, 0429, 0430, 0431, 0432, 0435, 0436, 0437, 0446, 0447, 0448, 0449, 0450, 0451, 0452, 0453, 0454, 0456, 0457, 0458, 0466, 0467, 0468, 0469, 0476, 0477, 0478, 0481, 0482, 0483, 0484, 0485, 0486, 0487, 0493, 0494, 0495, 0496, 0513, 0514, 0515, 0536, 0537, 0540, 0541, 0542, 0546, 0547, 0550, 0551, 0554, 0555, 0556, 0557, 0558, 0559, 0560, 0561, 0562, 0563, 0564, 0565, 0580, 0588, 0589, 0590, 0591, 0601, 0604, 0605, 0609, 0611, 0612, 0614, 0616, 0617, 0618, 0619, 0620, 0621, 0622, 0627, 0628, 0629, 0631, 0633, 0635, 0638, 0639, 0640, 0641, 0642, 0643, 0644, 0645, 0646, 0647, 0649, 0650, 0651, 0653, 0655, 0656, 0657, 0659, 0660, 0662, 0664, 0665, 0671, 0673, 0674, 0675, 0677, 0682, 0683, 0684, 0685, 0690, 0698, 0699, 0708, 0709, 0715, 0736, 0737, 0738, 0739, 0746, 0747, 0755, 0756, 0757, 0758, 0766, 0767, 0768, 0774, 0775, 0801, 0802, 0803, 0804, 0810, 0811, 0812, 0821, 0828, 0830, 0831, 0834, 0835, 0844, 0845, 0850, 0851, 0852, 0856, 0857, 0858, 0859, 0866, 0873, 0875, 0877, 0899, 0930, 0940, 0941, 0943, 0944, 0945, 1001, 1002, 1003, 1006, 1008, 1011, 1012, 1014, 1015, 1018, 1019, 1020, 1021, 1022, 1023, 1024, 1025, 1026, 1027, 1028, 1029, 1030, 1031, 1033, 1034, 1035, 1036, 1037, 1038, 1041, 1042, 1043, 1044, 1045, 1046, 1047, 1048, 1051, 1052, 1053, 1054, 1055, 1056, 3236, 3237, 3238, 3244, 3664
Enforcement: QA·C

MISRA-C:2004 Rule 1.2 (Required)
No reliance shall be placed on undefined or unspecified behaviour.
QA·C Messages: 0160, 0161, 0162, 0163, 0164, 0165, 0166, 0167, 0168, 0169, 0170, 0171, 0172, 0173, 0174, 0175, 0176, 0177, 0178, 0179, 0184, 0185, 0186, 0190, 0191, 0192, 0193, 0194, 0195, 0196, 0197, 0198, 0199, 0200, 0201, 0203, 0204, 0206, 0207, 0208, 0235, 0275, 0301, 0302, 0304, 0307, 0309, 0337, 0400, 0401, 0402, 0403, 0475, 0543, 0544, 0545, 0602, 0623, 0625, 0626, 0630, 0632, 0636, 0654, 0658, 0661, 0667, 0668, 0672, 0676, 0678, 0680, 0706, 0745, 0777, 0779, 0809, 0813, 0814, 0836, 0837, 0848, 0853, 0854, 0864, 0865, 0867, 0872, 0874, 0885, 0887, 0888, 0914, 0915, 0942, 1509, 1510, 2800, 2810, 2820, 2830, 2840, 3113, 3114, 3239, 3311, 3312, 3319, 3437, 3438
Enforcement: QA·C and CMA

MISRA-C:2004 Rule 1.3 (Required)
Multiple compilers and/or languages shall only be used if there is a common defined interface standard for object code to which the languages/compilers/assemblers conform.
Enforcement: NSE (Not Statically Enforceable)

MISRA-C:2004 Rule 1.4 (Required)
The compiler/linker shall be checked to ensure that 31 character significance and case sensitivity are supported for external identifiers.
Enforcement: NSE

MISRA-C:2004 Rule 1.5 (Advisory)
Floating-point implementations should comply with a defined floating-point standard.
Enforcement: NSE

MISRA-C:2004 Rule 2.1 (Required)
Assembly language shall be encapsulated and isolated.
QA·C Messages: 3006
Enforcement: QA·C

MISRA-C:2004 Rule 2.2 (Required)
Source code shall only use C-style comments.
QA·C Messages: 1011
Enforcement: QA·C

MISRA-C:2004 Rule 2.3 (Required)
The character sequence /* shall not be used within a comment.
QA·C Messages: 3108
Enforcement: QA·C

MISRA-C:2004 Rule 2.4 (Advisory)
Sections of code should not be 'commented out'.
Enforcement: NSE

MISRA-C:2004 Rule 3.1 (Required)
All usage of implementation-defined behaviour shall be documented.
QA·C Messages: 0202, 0284, 0285, 0286, 0287, 0288, 0289, 0292, 0299, 0303, 0305, 0306, 0308, 0581, 0634, 0878, 2850, 2851, 2852, 2853, 2855, 2856, 2857, 2860, 2861, 2862, 2890, 2895
Enforcement: QA·C and Dataflow

MISRA-C:2004 Rule 3.2 (Required)
The character set and the corresponding encoding shall be documented.
Enforcement: NSE

MISRA-C:2004 Rule 3.3 (Advisory)
The implementation of integer division in the chosen compiler should be determined, documented and taken into account.
Enforcement: NSE

MISRA-C:2004 Rule 3.4 (Required)
All uses of the #pragma directive shall be documented and explained.
QA·C Messages: 3116

Enforcement: QA·C

MISRA-C:2004 Rule 3.5 (Required)
If it is being relied upon, the implementation-defined behaviour and packing of bitfields shall be documented.
Enforcement: NSE

MISRA-C:2004 Rule 3.6 (Required)
All libraries used in production code shall be written to comply with the provisions of this document, and shall have been subject to appropriate validation.
Enforcement: NSE

MISRA-C:2004 Rule 4.1 (Required)
Only those escape sequences that are defined in the ISO C standard shall be used.
QA·C Messages: 0235, 3610
Enforcement: QA·C

MISRA-C:2004 Rule 4.2 (Required)
Trigraphs shall not be used.
QA·C Messages: 3601
Enforcement: QA·C

MISRA-C:2004 Rule 5.1 (Required)
Identifiers (internal and external) shall not rely on the significance of more than 31 characters.
QA·C Messages: 0777, 0779
Enforcement: QA·C

MISRA-C:2004 Rule 5.2 (Required)
Identifiers in an inner scope shall not use the same name as an identifier in an outer scope, and therefore hide that identifier.
QA·C Messages: 2547, 3334
Enforcement: QA·C

MISRA-C:2004 Rule 5.3 (Required)
A typedef name shall be a unique identifier.
QA·C Messages: 1506, 1507, 1508, 3448
Enforcement: QA·C and CMA

MISRA-C:2004 Rule 5.4 (Required)
A tag name shall be a unique identifier.
QA·C Messages: 0547
Enforcement: QA·C

MISRA-C:2004 Rule 5.5 (Advisory)
No object or function identifier with static storage duration should be reused.
QA·C Messages: 1525, 1526, 1527, 1528, 1529

Enforcement: CMA

MISRA-C:2004 Rule 5.6 (Advisory)
No identifier in one name space should have the same spelling as an identifier in another name space, with the exception of structure member and union member names.
QA·C Messages: 0780, 0781
Enforcement: QA·C

MISRA-C:2004 Rule 5.7 (Advisory)
No identifier name should be reused.
Enforcement: Not enforced

MISRA-C:2004 Rule 6.1 (Required)
The plain char type shall be used only for the storage and use of character values.
QA·C Messages: 1810, 1811, 1812, 1813, 4401, 4421, 4431, 4441, 4451, 4510, 4511, 4512, 4513, 4514, 4517, 4518, 4519
Enforcement: QA·C

MISRA-C:2004 Rule 6.2 (Required)
Signed and unsigned char type shall be used only for the storage and use of numeric values.
QA·C Messages: 4413, 4414
Enforcement: QA·C

MISRA-C:2004 Rule 6.3 (Advisory)
Typedefs that indicate size and signedness should be used in place of the basic numerical types.
QA·C Messages: 5013
Enforcement: Secondary-analysis.

MISRA-C:2004 Rule 6.4 (Required)
Bit fields shall only be defined to be of type unsigned int or signed int.
QA·C Messages: 0634, 0635
Enforcement: QA·C

MISRA-C:2004 Rule 6.5 (Required)
Bit fields of signed type shall be at least 2 bits long.
QA·C Messages: 3659, 3660, 3665
Enforcement: QA·C

MISRA-C:2004 Rule 7.1 (Required)
Octal constants (other than zero) and octal escape sequences shall not be used.
QA·C Messages: 0336, 0339, 3628
Enforcement: QA·C

MISRA-C:2004 Rule 8.1 (Required)
Functions shall have prototype declarations and the prototype shall be visible at both the

function definition and call.
QA·C Messages: 3002, 3335, 3450
Enforcement: QA·C

MISRA-C:2004 Rule 8.2 (Required)
Whenever an object or function is declared or defined, its type shall be explicitly stated.
QA·C Messages: 2050, 2051
Enforcement: QA·C

MISRA-C:2004 Rule 8.3 (Required)
For each function parameter the type given in the declaration and definition shall be identical, and the return types shall also be identical.
QA·C Messages: 0624, 1331, 1332, 1333, 3320, 3675
Enforcement: QA·C

MISRA-C:2004 Rule 8.4 (Required)
If objects or functions are declared more than once their types shall be compatible.
QA·C Messages: 0626, 0627, 0628, 1510
Enforcement: QA·C and CMA

MISRA-C:2004 Rule 8.5 (Required)
There shall be no definitions of objects or functions in a header file.
QA·C Messages: 3406, 3480
Enforcement: QA·C

MISRA-C:2004 Rule 8.6 (Required)
Functions shall be declared at file scope.
QA·C Messages: 3221
Enforcement: QA·C

MISRA-C:2004 Rule 8.7 (Required)
Objects shall be defined at block scope if they are only accessed from within a single function.
QA·C Messages: 1514, 3218
Enforcement: QA·C and CMA

MISRA-C:2004 Rule 8.8 (Required)
An external object or function shall be declared in one and only one file.
QA·C Messages: 1513, 3222, 3408, 3447, 3451
Enforcement: QA·C and CMA

MISRA-C:2004 Rule 8.9 (Required)
An identifier with external linkage shall have exactly one external definition.
QA·C Messages: 0630, 1509
Enforcement: QA·C and CMA

MISRA-C:2004 Rule 8.10 (Required)

All declarations and definitions of objects or functions at file scope shall have internal linkage unless external linkage is required.
QA·C Messages: 1504, 1505
Enforcement: CMA

MISRA-C:2004 Rule 8.11 (Required)
The static storage class specifier shall be used in definitions and declarations of objects and functions that have internal linkage.
QA·C Messages: 3224
Enforcement: QA·C

MISRA-C:2004 Rule 8.12 (Required)
When an array is declared with external linkage, its size shall be stated explicitly or defined implicitly by initialisation.
QA·C Messages: 3684
Enforcement: QA·C

MISRA-C:2004 Rule 9.1 (Required)
All automatic variables shall have been assigned a value before being used.
QA·C Messages: 2883, 2961, 2962, 2971, 2972
Enforcement: Dataflow

MISRA-C:2004 Rule 9.2 (Required)
Braces shall be used to indicate and match the structure in the non-zero initialisation of arrays and structures.
QA·C Messages: 0686, 0693, 0694
Enforcement: QA·C

MISRA-C:2004 Rule 9.3 (Required)
In an enumerator list, the '=' construct shall not be used to explicitly initialise members other than the first, unless all items are explicitly initialised.
QA·C Messages: 0723
Enforcement: QA·C

MISRA-C:2004 Rule 10.1 (Required)
The value of an expression of integer type shall not be implicitly converted to a different underlying type if: a) it is not a conversion to a wider integer type of the same signedness, or b) the expression is complex, or c) the expression is not constant and is a function argument, or d) the expression is not constant and is a return expression
QA·C Messages: 0570, 0572, 1256, 1257, 1290, 1291, 1292, 1293, 1294, 1295, 1296, 1297, 1298, 1299, 1317, 1461, 1800, 1802, 1803, 1804, 1810, 1811, 1812, 1813, 1820, 1821, 1822, 1823, 1824, 1830, 1831, 1832, 1833, 1834, 1840, 1841, 1842, 1843, 1844, 1850, 1851, 1852, 1853, 1854, 1860, 1861, 1862, 1863, 1864, 1880, 1881, 1882, 1890, 1891, 2900, 4117, 4401, 4402, 4403, 4404, 4405, 4410, 4412, 4413, 4414, 4415, 4420, 4421, 4422, 4423, 4424, 4425, 4430, 4431, 4432, 4434, 4435, 4436, 4437, 4440, 4441,

4442, 4443, 4445, 4446, 4447, 4451, 4460, 4461, 4463, 4464, 4470, 4471, 4480, 4481, 4490, 4491, 4500, 4501, 4502, 4503, 4504, 4505, 4507
Enforcement: QA·C

MISRA-C:2004 Rule 10.2 (Required)
The value of an expression of floating type shall not be implicitly converted to a different type if: a) it is not a conversion to a wider floating type, or b) the expression is complex, or c) the expression is a function argument, or d) the expression is a return expression
QA·C Messages: 1264, 1265, 1266, 1880, 1881, 1882, 1892, 1893, 1894, 1895, 4450, 4452, 4453, 4454, 4462, 4465, 4472, 4482, 4492
Enforcement: QA·C

MISRA-C:2004 Rule 10.3 (Required)
The value of a complex expression of integer type shall only be cast to a type of the same signedness that is no wider than the underlying type of the expression.
QA·C Messages: 4118, 4390, 4391, 4393, 4394
Enforcement: QA·C

MISRA-C:2004 Rule 10.4 (Required)
The value of a complex expression of floating type shall only be cast to a floating type that is narrower or of the same size.
QA·C Messages: 4392, 4395
Enforcement: QA·C

MISRA-C:2004 Rule 10.5 (Required)
If the bitwise operators and << are applied to an operand of underlying type unsigned char or unsigned short, the result shall be immediately cast to the underlying type of the operand.
QA·C Messages: 4397, 4398, 4399, 4498, 4499
Enforcement: QA·C

MISRA-C:2004 Rule 10.6 (Required)
A "U" suffix shall be applied to all constants of unsigned type.
QA·C Messages: 1281
Enforcement: QA·C

MISRA-C:2004 Rule 11.1 (Required)
Conversions shall not be performed between a pointer to a function and any type other than an integral type.
QA·C Messages: 0302, 0307, 0313
Enforcement: QA·C

MISRA-C:2004 Rule 11.2 (Required)
Conversions shall not be performed between a pointer to object and any type other than an integral type, another pointer to object type or a pointer to void.
QA·C Messages: 0301

Enforcement: QA·C

MISRA-C:2004 Rule 11.3 (Advisory)
A cast should not be performed between a pointer type and an integral type.
QA·C Messages: 0303, 0305, 0306, 0360, 0361, 0362
Enforcement: QA·C

MISRA-C:2004 Rule 11.4 (Advisory)
A cast should not be performed between a pointer to object type and a different pointer to object type.
QA·C Messages: 0310, 0316, 0317
Enforcement: QA·C

MISRA-C:2004 Rule 11.5 (Required)
A cast shall not be performed that removes any const or volatile qualification from the type addressed by a pointer.
QA·C Messages: 0311, 0312
Enforcement: QA·C

MISRA-C:2004 Rule 12.1 (Advisory)
Limited dependence should be placed on C's operator precedence rules in expressions.
QA·C Messages: 3389, 3391, 3392, 3393, 3394, 3395, 3396, 3397
Enforcement: QA·C

MISRA-C:2004 Rule 12.2 (Required)
The value of an expression shall be the same under any order of evaluation that the standard permits.
QA·C Messages: 0400, 0401, 0402, 0403
Enforcement: QA·C

MISRA-C:2004 Rule 12.3 (Required)
The sizeof operator shall not be used on expressions that contain side effects.
QA·C Messages: 3307
Enforcement: QA·C

MISRA-C:2004 Rule 12.4 (Required)
The right hand operand of a logical && or || operator shall not contain side effects.
QA·C Messages: 3415
Enforcement: QA·C

MISRA-C:2004 Rule 12.5 (Required)
The operands of a logical && or || shall be primary-expressions.
QA·C Messages: 3398, 3399, 3400
Enforcement: QA·C

MISRA-C:2004 Rule 12.6 (Advisory)
The operands of logical operators (&&, || and !) should be effectively Boolean. Expres-

sions that are effectively Boolean should not be used as operands to operators other than (&&, ||, !, =, ==, != and ?:).
QA·C Messages: 3322, 3377, 4101, 4102, 4103, 4104, 4105, 4106, 4107, 4108, 4109, 4110, 4111, 4112, 4113, 4114, 4115, 4116
Enforcement: QA·C

MISRA-C:2004 Rule 12.7 (Required)
Bitwise operators shall not be applied to operands whose underlying type is signed.
QA·C Messages: 4532, 4533
Enforcement: QA·C

MISRA-C:2004 Rule 12.8 (Required)
The right hand operand of a shift operator shall lie between zero and one less than the width in bits of the underlying type of the left hand operand.
QA·C Messages: 0499, 2790
Enforcement: QA·C

MISRA-C:2004 Rule 12.9 (Required)
The unary minus operator shall not be applied to an expression whose underlying type is unsigned.
QA·C Messages: 3101, 3102
Enforcement: QA·C

MISRA-C:2004 Rule 12.10 (Required)
The comma operator shall not be used.
QA·C Messages: 3417, 3418
Enforcement: QA·C

MISRA-C:2004 Rule 12.11 (Advisory)
Evaluation of constant unsigned integer expressions should not lead to wrap-around.
QA·C Messages: 2910
Enforcement: QA·C

MISRA-C:2004 Rule 12.12 (Required)
The underlying bit representations of floating-point values shall not be used.
QA·C Messages: 3629
Enforcement: QA·C

MISRA-C:2004 Rule 12.13 (Advisory)
The increment (++) and decrement (–) operators should not be mixed with other operators in an expression.
QA·C Messages: 3440
Enforcement: QA·C

MISRA-C:2004 Rule 13.1 (Required)
Assignment operators shall not be used in expressions that yield a Boolean value.

QA·C Messages: 3326
Enforcement: QA·C

MISRA-C:2004 Rule 13.2 (Advisory)
Tests of a value against zero should be made explicit, unless the operand is effectively Boolean.
QA·C Messages: 3344, 4528, 4529, 4538, 4539, 4548, 4549, 4558, 4559, 4568, 4569
Enforcement: QA·C

MISRA-C:2004 Rule 13.3 (Required)
Floating-point expressions shall not be tested for equality or inequality.
QA·C Messages: 3341
Enforcement: QA·C

MISRA-C:2004 Rule 13.4 (Required)
The controlling expression of a for statement shall not contain any objects of floating type.
QA·C Messages: 3340, 3342
Enforcement: QA·C

MISRA-C:2004 Rule 13.5 (Required)
The three expressions of a for statement shall be concerned only with loop control.
QA·C Messages: 2462, 2463
Enforcement: QA·C

MISRA-C:2004 Rule 13.6 (Required)
Numeric variables being used within a for loop for iteration counting shall not be modified in the body of the loop.
QA·C Messages: 2469
Enforcement: QA·C

MISRA-C:2004 Rule 13.7 (Required)
Boolean operations whose results are invariant shall not be permitted.
QA·C Messages: 2990, 2991, 2992, 2993, 2994, 2995, 2996
Enforcement: Dataflow

MISRA-C:2004 Rule 14.1 (Required)
There shall be no unreachable code.
QA·C Messages: 0594, 1460, 1503, 2008, 2742, 2744, 2880, 2882, 3219
Enforcement: QA·C and Dataflow

MISRA-C:2004 Rule 14.2 (Required)
All non-null statements shall either (i) have at least one side effect however executed, or (ii) cause control flow to change.
QA·C Messages: 3110, 3112, 3425, 3426, 3427
Enforcement: QA·C

MISRA-C:2004 Rule 14.3 (Required)

Before preprocessing, a null statement shall only occur on a line by itself; it may be followed by a comment provided that the first character following the null statement is a white-space character.
QA·C Messages: 3138
Enforcement: QA·C

MISRA-C:2004 Rule 14.4 (Required)
The goto statement shall not be used.
QA·C Messages: 2001
Enforcement: QA·C

MISRA-C:2004 Rule 14.5 (Required)
The continue statement shall not be used.
QA·C Messages: 0770
Enforcement: QA·C

MISRA-C:2004 Rule 14.6 (Required)
For any iteration statement there shall be at most one break statement used for loop termination.
QA·C Messages: 0771
Enforcement: QA·C

MISRA-C:2004 Rule 14.7 (Required)
A function shall have a single point of exit at the end of the function.
QA·C Messages: 2889
Enforcement: QA·C

MISRA-C:2004 Rule 14.8 (Required)
The statement forming the body of a switch, while, do ... while or for statement shall be a compound statement.
QA·C Messages: 2212, 2214
Enforcement: QA·C

MISRA-C:2004 Rule 14.9 (Required)
An if (expression) construct shall be followed by a compound statement. The else keyword shall be followed by either a compound statement, or another if statement.
QA·C Messages: 2212, 2214, 3402
Enforcement: QA·C

MISRA-C:2004 Rule 14.10 (Required)
All if ... else if constructs shall be terminated with an else clause.
QA·C Messages: 2004
Enforcement: QA·C

MISRA-C:2004 Rule 15.0 (Required)
The MISRA C switch syntax shall be used.

QA·C Messages: 3234
Enforcement: QA·C

MISRA-C:2004 Rule 15.1 (Required)
A switch label shall only be used when the most closely-enclosing compound statement is the body of a switch statement.
QA·C Messages: 2019
Enforcement: QA·C

MISRA-C:2004 Rule 15.2 (Required)
An unconditional break statement shall terminate every non-empty switch clause.
QA·C Messages: 2003, 2020
Enforcement: QA·C

MISRA-C:2004 Rule 15.3 (Required)
The final clause of a switch statement shall be the default clause.
QA·C Messages: 2002, 2009
Enforcement: QA·C

MISRA-C:2004 Rule 15.4 (Required)
A switch expression shall not represent a value that is effectively Boolean.
QA·C Messages: 0735
Enforcement: QA·C

MISRA-C:2004 Rule 15.5 (Required)
Every switch statement shall have at least one case clause.
QA·C Messages: 3315
Enforcement: QA·C

MISRA-C:2004 Rule 16.1 (Required)
Functions shall not be defined with a variable number of arguments.
QA·C Messages: 1337
Enforcement: QA·C

MISRA-C:2004 Rule 16.2 (Required)
Functions shall not call themselves, either directly or indirectly.
QA·C Messages: 1520, 3670
Enforcement: QA·C and CMA

MISRA-C:2004 Rule 16.3 (Required)
Identifiers shall be given for all of the parameters in a function prototype declaration.
QA·C Messages: 1335, 1336
Enforcement: QA·C

MISRA-C:2004 Rule 16.4 (Required)
The identifiers used in the declaration and definition of a function shall be identical.
QA·C Messages: 1330

Enforcement: QA·C

MISRA-C:2004 Rule 16.5 (Required)
Functions with no parameters shall be declared and defined with the parameter list void.
QA·C Messages: 3001, 3007
Enforcement: QA·C

MISRA-C:2004 Rule 16.6 (Required)
The number of arguments passed to a function shall match the number of parameters.
QA·C Messages: 0422, 0423, 3319
Enforcement: QA·C

MISRA-C:2004 Rule 16.7 (Advisory)
A pointer parameter in a function prototype should be declared as pointer to const if the pointer is not used to modify the addressed object.
QA·C Messages: 3673
Enforcement: QA·C

MISRA-C:2004 Rule 16.8 (Required)
All exit paths from a function with non-void return type shall have an explicit return statement with an expression.
QA·C Messages: 0745, 2887, 2888, 3113, 3114
Enforcement: QA·C

MISRA-C:2004 Rule 16.9 (Required)
A function identifier shall only be used with either a preceding &, or with a parenthesised parameter list, which may be empty.
QA·C Messages: 3635
Enforcement: QA·C

MISRA-C:2004 Rule 16.10 (Required)
If a function returns error information, then that error information shall be tested.
QA·C Messages: 3200
Enforcement: QA·C

MISRA-C:2004 Rule 17.1 (Required)
Pointer arithmetic shall only be applied to pointers that address an array or array element.
QA·C Messages: 2930, 2931, 2932
Enforcement: QA·C and Dataflow

MISRA-C:2004 Rule 17.2 (Required)
Pointer subtraction shall only be applied to pointers that address elements of the same array.
QA·C Messages: 2771, 2772
Enforcement: Dataflow

MISRA-C:2004 Rule 17.3 (Required)

>, >=, <, <= shall not be applied to pointer types except where they point to the same array.
QA·C Messages: 2771, 2772
Enforcement: Dataflow

MISRA-C:2004 Rule 17.4 (Required)
Array indexing shall be the only allowed form of pointer arithmetic.
QA·C Messages: 0488, 0489, 0491, 0492
Enforcement: QA·C

MISRA-C:2004 Rule 17.5 (Advisory)
The declaration of objects should contain no more than 2 levels of pointer indirection.
QA·C Messages: 3260, 3261, 3262, 3263
Enforcement: QA·C

MISRA-C:2004 Rule 17.6 (Required)
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.
QA·C Messages: 3217, 3225, 3230, 4140
Enforcement: QA·C

MISRA-C:2004 Rule 18.1 (Required)
All structure and union types shall be complete at the end of a translation unit.
QA·C Messages: 0544, 0545, 0623, 0636
Enforcement: QA·C

MISRA-C:2004 Rule 18.2 (Required)
An object shall not be assigned to an overlapping object.
QA·C Messages: 2776, 2777
Enforcement: Dataflow

MISRA-C:2004 Rule 18.3 (Required)
An area of memory shall not be reused for unrelated purposes.
Enforcement: NSE

MISRA-C:2004 Rule 18.4 (Required)
Unions shall not be used.
QA·C Messages: 0750, 0759
Enforcement: QA·C

MISRA-C:2004 Rule 19.1 (Advisory)
#include statements in a file should only be preceded by other preprocessor directives or comments.
QA·C Messages: 5087
Enforcement: Secondary-analysis

MISRA-C:2004 Rule 19.2 (Advisory)

Non-standard characters should not occur in header file names in #include directives.
QA·C Messages: 0813, 0814, 0831
Enforcement: QA·C

MISRA-C:2004 Rule 19.3 (Required)
The #include directive shall be followed by either a <filename> or “filename” sequence.
QA·C Messages: 0809
Enforcement: QA·C

MISRA-C:2004 Rule 19.4 (Required)
C macros shall only expand to a braced initialiser, a constant, a string literal, a parenthesised expression, a type qualifier, a storage class specifier, or a do-while-zero construct.
QA·C Messages: 3409, 3411, 3412, 3413, 3431, 3452, 3458, 3460, 3461
Enforcement: QA·C

MISRA-C:2004 Rule 19.5 (Required)
Macros shall not be #define’d or #undef’d within a block.
QA·C Messages: 0842
Enforcement: QA·C

MISRA-C:2004 Rule 19.6 (Required)
#undef shall not be used.
QA·C Messages: 0841
Enforcement: QA·C

MISRA-C:2004 Rule 19.7 (Advisory)
A function should be used in preference to a function-like macro.
QA·C Messages: 3453
Enforcement: QA·C

MISRA-C:2004 Rule 19.8 (Required)
A function-like macro shall not be invoked without all of its arguments.
QA·C Messages: 0850, 0856
Enforcement: QA·C

MISRA-C:2004 Rule 19.9 (Required)
Arguments to a function-like macro shall not contain tokens that look like preprocessing directives.
QA·C Messages: 0853
Enforcement: QA·C

MISRA-C:2004 Rule 19.10 (Required)
In the definition of a function-like macro each instance of a parameter shall be enclosed in parentheses unless it is used as the operand of # or ##.
QA·C Messages: 3410
Enforcement: QA·C

MISRA-C:2004 Rule 19.11 (Required)
All macro identifiers in preprocessor directives shall be defined before use, except in #ifdef and #ifndef preprocessor directives and the defined() operator.
QA·C Messages: 3332
Enforcement: QA·C

MISRA-C:2004 Rule 19.12 (Required)
There shall be at most one occurrence of the # or ## preprocessor operators in a single macro definition.
QA·C Messages: 0880, 0881, 0884
Enforcement: QA·C

MISRA-C:2004 Rule 19.13 (Advisory)
The # and ## preprocessor operators should not be used.
QA·C Messages: 0341, 0342
Enforcement: QA·C

MISRA-C:2004 Rule 19.14 (Required)
The defined preprocessor operator shall only be used in one of the two standard forms.
QA·C Messages: 0885, 0887, 0888
Enforcement: QA·C

MISRA-C:2004 Rule 19.15 (Required)
Precautions shall be taken in order to prevent the contents of a header file being included twice.
QA·C Messages: 0883
Enforcement: QA·C

MISRA-C:2004 Rule 19.16 (Required)
Preprocessing directives shall be syntactically meaningful even when excluded by the preprocessor.
QA·C Messages: 3115
Enforcement: QA·C

MISRA-C:2004 Rule 19.17 (Required)
All #else, #elif and #endif preprocessor directives shall reside in the same file as the #if or #ifdef directive to which they are related.
QA·C Messages: 3317, 3318
Enforcement: QA·C

MISRA-C:2004 Rule 20.1 (Required)
Reserved identifiers, macros and functions in the standard library, shall not be defined, redefined or undefined.
QA·C Messages: 0836, 0848, 0854, 3439, 4600, 4601
Enforcement: QA·C

MISRA-C:2004 Rule 20.2 (Required)
The names of standard library macros, objects and functions shall not be reused.
QA·C Messages: 0602, 4602, 4603, 4604, 4605, 4606, 4607, 4608
Enforcement: QA·C

MISRA-C:2004 Rule 20.3 (Required)
The validity of values passed to library functions shall be checked.
QA·C Messages: 2810, 2811, 2812, 2840, 2841, 2842
Enforcement: QA·C and Dataflow

MISRA-C:2004 Rule 20.4 (Required)
Dynamic heap memory allocation shall not be used.
QA·C Messages: 5118
Enforcement: Secondary-analysis

MISRA-C:2004 Rule 20.5 (Required)
The error indicator errno shall not be used.
QA·C Messages: 5119
Enforcement: Secondary-analysis

MISRA-C:2004 Rule 20.6 (Required)
The macro offsetof, in library <stddef.h>, shall not be used.
QA·C Messages: 5120
Enforcement: Secondary-analysis

MISRA-C:2004 Rule 20.7 (Required)
The setjmp macro and the longjmp function shall not be used.
QA·C Messages: 5122
Enforcement: Secondary-analysis

MISRA-C:2004 Rule 20.8 (Required)
The signal handling facilities of <signal.h> shall not be used.
QA·C Messages: 5123
Enforcement: Secondary-analysis

MISRA-C:2004 Rule 20.9 (Required)
The input/output library <stdio.h> shall not be used in production code.
QA·C Messages: 5124
Enforcement: Secondary-analysis

MISRA-C:2004 Rule 20.10 (Required)
The library functions atof, atoi and atol from library <stdlib.h> shall not be used.
QA·C Messages: 5125
Enforcement: Secondary-analysis

MISRA-C:2004 Rule 20.11 (Required)
The library functions abort, exit, getenv and system from library <stdlib.h> shall not be

used.
QA·C Messages: 5126
Enforcement: Secondary-analysis

MISRA-C:2004 Rule 20.12 (Required)
The time handling functions of library <time.h> shall not be used.
QA·C Messages: 5127
Enforcement: Secondary-analysis.

MISRA-C:2004 Rule 21.1 (Required)
Minimisation of run-time failures shall be ensured by the use of at least one of (a) static analysis tools/techniques; (b) dynamic analysis tools/techniques; (c) explicit coding of checks to handle run-time faults
QA·C Messages: 2791, 2792, 2801, 2802, 2811, 2812, 2821, 2822, 2831, 2832, 2841, 2842, 2845, 2846, 2847, 2871, 2872, 2877, 2891, 2892, 2896, 2897, 2901, 2902, 2911, 2912, 2980, 2981, 2982, 2983, 2984, 2985, 2986
Enforcement: Dataflow

6 Rule Enforcement

This report lists all rules, grouping by "Advisory" or "Required" and whether enforced.

6.1 Enforced Rules - Advisory

Table 6.1: M2CM Enforced Rules - Advisory

Rule Id	Rule	Enforcement
5.5	No object or function identifier with static storage duration should be reused.	1525, 1526, 1527, 1528, 1529
5.6	No identifier in one name space should have the same spelling as an identifier in another name space, with the exception of structure member and union member names.	0780, 0781
6.3	Typedefs that indicate size and signedness should be used in place of the basic numerical types.	5013
11.3	A cast should not be performed between a pointer type and an integral type.	0303, 0305, 0306, 0360, 0361, 0362
11.4	A cast should not be performed between a pointer to object type and a different pointer to object type.	0310, 0316, 0317
12.1	Limited dependence should be placed on C's operator precedence rules in expressions.	3389, 3391, 3392, 3393, 3394, 3395, 3396, 3397
12.6	The operands of logical operators (&&, || and !) should be effectively Boolean. Expressions that are effectively Boolean should not be used as operands to operators other than (&&, ||, !, =, ==, != and ?:).	3322, 3377, 4101, 4102, 4103, 4104, 4105, 4106, 4107, 4108, 4109, 4110, 4111, 4112, 4113, 4114, 4115, 4116
12.11	Evaluation of constant unsigned integer expressions should not lead to wrap-around.	2910
12.13	The increment (++) and decrement (–) operators should not be mixed with other operators in an expression.	3440
13.2	Tests of a value against zero should be made explicit, unless the operand is effectively Boolean.	3344, 4528, 4529, 4538, 4539, 4548, 4549, 4558, 4559, 4568, 4569
16.7	A pointer parameter in a function prototype should be declared as pointer to const if the pointer is not used to modify the addressed object.	3673
17.5	The declaration of objects should contain no more than 2 levels of pointer indirection.	3260, 3261, 3262, 3263

continued on next page

Table 6.1 – continued from previous page

Rule Id	Rule	Enforcement
19.1	#include statements in a file should only be preceded by other preprocessor directives or comments.	5087
19.2	Non-standard characters should not occur in header file names in #include directives.	0813, 0814, 0831
19.7	A function should be used in preference to a function-like macro.	3453
19.13	The # and ## preprocessor operators should not be used.	0341, 0342

Count of Enforced Advisory Rules = 16

6.2 Unenforced Rules - Advisory

Table 6.2: M2CM Unenforced Rules - Advisory

Rule Id	Rule
5.7	No identifier name should be reused.

Count of Unenforced Advisory Rules = 1

6.3 Not Statically Enforceable Rules - Advisory

Table 6.3: M2CM Not Statically Enforceable Rules - Advisory

Rule Id	Rule
1.5	Floating-point implementations should comply with a defined floating-point standard.
2.4	Sections of code should not be 'commented out'.
3.3	The implementation of integer division in the chosen compiler should be determined, documented and taken into account.

Count of Not Statically Enforceable Advisory Rules = 3

6.4 Enforced Rules - Required

Table 6.4: M2CM Enforced Rules - Required

Rule Id	Rule	Enforcement
1.1	All code shall conform to ISO/IEC 9899:1990 C programming language, ISO 9899, amended and corrected by ISO/IEC 9899/COR1:1995, ISO/IEC 9899/AMD1:1995, and ISO/IEC 9899/COR2: 1996192	0180, 0232, 0233, 0240, 0241, 0244, 0246, 0261, 0320, 0321, 0322, 0338, 0410, 0422, 0423, 0426, 0427, 0429, 0430, 0431, 0432, 0435, 0436, 0437, 0446, 0447, 0448, 0449, 0450, 0451, 0452, 0453, 0454, 0456, 0457, 0458, 0466, 0467, 0468, 0469, 0476, 0477, 0478, 0481, 0482, 0483, 0484, 0485
1.1	Continued...	0486, 0487, 0493, 0494, 0495, 0496, 0513, 0514, 0515, 0536, 0537, 0540, 0541, 0542, 0546, 0547, 0550, 0551, 0554, 0555, 0556, 0557, 0558, 0559, 0560, 0561, 0562, 0563, 0564, 0565, 0580, 0588, 0589, 0590, 0591, 0601, 0604, 0605, 0609, 0611, 0612, 0614, 0616, 0617, 0618, 0619, 0620, 0621

continued on next page

Table 6.4 – continued from previous page

Rule Id	Rule	Enforcement
1.1	Continued...	0622, 0627, 0628, 0629, 0631, 0633, 0635, 0638, 0639, 0640, 0641, 0642, 0643, 0644, 0645, 0646, 0647, 0649, 0650, 0651, 0653, 0655, 0656, 0657, 0659, 0660, 0662, 0664, 0665, 0671, 0673, 0674, 0675, 0677, 0682, 0683, 0684, 0685, 0690, 0698, 0699, 0708, 0709, 0715, 0736, 0737, 0738, 0739
1.1	Continued...	0746, 0747, 0755, 0756, 0757, 0758, 0766, 0767, 0768, 0774, 0775, 0801, 0802, 0803, 0804, 0810, 0811, 0812, 0821, 0828, 0830, 0831, 0834, 0835, 0844, 0845, 0850, 0851, 0852, 0856, 0857, 0858, 0859, 0866, 0873, 0875, 0877, 0899, 0930, 0940, 0941, 0943, 0944, 0945, 1001, 1002, 1003, 1006

continued on next page

Table 6.4 – continued from previous page

Rule Id	Rule	Enforcement
1.1	Continued...	1008, 1011, 1012, 1014, 1015, 1018, 1019, 1020, 1021, 1022, 1023, 1024, 1025, 1026, 1027, 1028, 1029, 1030, 1031, 1033, 1034, 1035, 1036, 1037, 1038, 1041, 1042, 1043, 1044, 1045, 1046, 1047, 1048, 1051, 1052, 1053, 1054, 1055, 1056, 3236, 3237, 3238, 3244, 3664
1.2	No reliance shall be placed on undefined or unspecified behaviour.	0160, 0161, 0162, 0163, 0164, 0165, 0166, 0167, 0168, 0169, 0170, 0171, 0172, 0173, 0174, 0175, 0176, 0177, 0178, 0179, 0184, 0185, 0186, 0190, 0191, 0192, 0193, 0194, 0195, 0196, 0197, 0198, 0199, 0200, 0201, 0203, 0204, 0206

continued on next page

Table 6.4 – continued from previous page

Rule Id	Rule	Enforcement
1.2	Continued...	0207, 0208, 0235, 0275, 0301, 0302, 0304, 0307, 0309, 0337, 0400, 0401, 0402, 0403, 0475, 0543, 0544, 0545, 0602, 0623, 0625, 0626, 0630, 0632, 0636, 0654, 0658, 0661, 0667, 0668, 0672, 0676, 0678, 0680, 0706, 0745, 0777, 0779, 0809, 0813, 0814, 0836, 0837, 0848, 0853, 0854, 0864, 0865
1.2	Continued...	0867, 0872, 0874, 0885, 0887, 0888, 0914, 0915, 0942, 1509, 1510, 2800, 2810, 2820, 2830, 2840, 3113, 3114, 3239, 3311, 3312, 3319, 3437, 3438
2.1	Assembly language shall be encapsulated and isolated.	3006
2.2	Source code shall only use C-style comments.	1011
2.3	The character sequence /* shall not be used within a comment.	3108
3.1	All usage of implementation-defined behaviour shall be documented.	0202, 0284, 0285, 0286, 0287, 0288, 0289, 0292, 0299, 0303, 0305, 0306, 0308, 0581, 0634, 0878, 2850, 2851, 2852, 2853, 2855, 2856, 2857, 2860, 2861, 2862, 2890, 2895
3.4	All uses of the #pragma directive shall be documented and explained.	3116
4.1	Only those escape sequences that are defined in the ISO C standard shall be used.	0235, 3610

continued on next page

Table 6.4 – continued from previous page

Rule Id	Rule	Enforcement
4.2	Trigraphs shall not be used.	3601
5.1	Identifiers (internal and external) shall not rely on the significance of more than 31 characters.	0777, 0779
5.2	Identifiers in an inner scope shall not use the same name as an identifier in an outer scope, and therefore hide that identifier.	2547, 3334
5.3	A typedef name shall be a unique identifier.	1506, 1507, 1508, 3448
5.4	A tag name shall be a unique identifier.	0547
6.1	The plain char type shall be used only for the storage and use of character values.	1810, 1811, 1812, 1813, 4401, 4421, 4431, 4441, 4451, 4510, 4511, 4512, 4513, 4514, 4517, 4518, 4519
6.2	Signed and unsigned char type shall be used only for the storage and use of numeric values.	4413, 4414
6.4	Bit fields shall only be defined to be of type unsigned int or signed int.	0634, 0635
6.5	Bit fields of signed type shall be at least 2 bits long.	3659, 3660, 3665
7.1	Octal constants (other than zero) and octal escape sequences shall not be used.	0336, 0339, 3628
8.1	Functions shall have prototype declarations and the prototype shall be visible at both the function definition and call.	3002, 3335, 3450
8.2	Whenever an object or function is declared or defined, its type shall be explicitly stated.	2050, 2051
8.3	For each function parameter the type given in the declaration and definition shall be identical, and the return types shall also be identical.	0624, 1331, 1332, 1333, 3320, 3675
8.4	If objects or functions are declared more than once their types shall be compatible.	0626, 0627, 0628, 1510
8.5	There shall be no definitions of objects or functions in a header file.	3406, 3480
8.6	Functions shall be declared at file scope.	3221
8.7	Objects shall be defined at block scope if they are only accessed from within a single function.	1514, 3218
8.8	An external object or function shall be declared in one and only one file.	1513, 3222, 3408, 3447, 3451
8.9	An identifier with external linkage shall have exactly one external definition.	0630, 1509

continued on next page

Table 6.4 – continued from previous page

Rule Id	Rule	Enforcement
8.10	All declarations and definitions of objects or functions at file scope shall have internal linkage unless external linkage is required.	1504, 1505
8.11	The static storage class specifier shall be used in definitions and declarations of objects and functions that have internal linkage.	3224
8.12	When an array is declared with external linkage, its size shall be stated explicitly or defined implicitly by initialisation.	3684
9.1	All automatic variables shall have been assigned a value before being used.	2883, 2961, 2962, 2971, 2972
9.2	Braces shall be used to indicate and match the structure in the non-zero initialisation of arrays and structures.	0686, 0693, 0694
9.3	In an enumerator list, the '=' construct shall not be used to explicitly initialise members other than the first, unless all items are explicitly initialised.	0723
10.1	The value of an expression of integer type shall not be implicitly converted to a different underlying type if: a) it is not a conversion to a wider integer type of the same signedness, or b) the expression is complex, or c) the expression is not constant and is a function argument, or d) the expression is not constant and is a return expression	0570, 0572, 1256, 1257, 1290, 1291, 1292, 1293, 1294, 1295, 1296, 1297, 1298, 1299, 1317, 1461, 1800, 1802, 1803, 1804, 1810, 1811, 1812, 1813, 1820, 1821, 1822, 1823, 1824, 1830, 1831, 1832, 1833, 1834, 1840, 1841, 1842, 1843, 1844, 1850, 1851, 1852, 1853, 1854, 1860, 1861, 1862, 1863

continued on next page

Table 6.4 – continued from previous page

Rule Id	Rule	Enforcement
10.1	Continued...	1864, 1880, 1881, 1882, 1890, 1891, 2900, 4117, 4401, 4402, 4403, 4404, 4405, 4410, 4412, 4413, 4414, 4415, 4420, 4421, 4422, 4423, 4424, 4425, 4430, 4431, 4432, 4434, 4435, 4436, 4437, 4440, 4441, 4442, 4443, 4445, 4446, 4447, 4451, 4460, 4461, 4463, 4464, 4470, 4471, 4480, 4481, 4490
10.1	Continued...	4491, 4500, 4501, 4502, 4503, 4504, 4505, 4507
10.2	The value of an expression of floating type shall not be implicitly converted to a different type if: a) it is not a conversion to a wider floating type, or b) the expression is complex, or c) the expression is a function argument, or d) the expression is a return expression	1264, 1265, 1266, 1880, 1881, 1882, 1892, 1893, 1894, 1895, 4450, 4452, 4453, 4454, 4462, 4465, 4472, 4482, 4492
10.3	The value of a complex expression of integer type shall only be cast to a type of the same signedness that is no wider than the underlying type of the expression.	4118, 4390, 4391, 4393, 4394
10.4	The value of a complex expression of floating type shall only be cast to a floating type that is narrower or of the same size.	4392, 4395
10.5	If the bitwise operators and << are applied to an operand of underlying type unsigned char or unsigned short, the result shall be immediately cast to the underlying type of the operand.	4397, 4398, 4399, 4498, 4499
10.6	A “U” suffix shall be applied to all constants of unsigned type.	1281
11.1	Conversions shall not be performed between a pointer to a function and any type other than an integral type.	0302, 0307, 0313

continued on next page

Table 6.4 – continued from previous page

Rule Id	Rule	Enforcement
11.2	Conversions shall not be performed between a pointer to object and any type other than an integral type, another pointer to object type or a pointer to void.	0301
11.5	A cast shall not be performed that removes any const or volatile qualification from the type addressed by a pointer.	0311, 0312
12.2	The value of an expression shall be the same under any order of evaluation that the standard permits.	0400, 0401, 0402, 0403
12.3	The sizeof operator shall not be used on expressions that contain side effects.	3307
12.4	The right hand operand of a logical && or || operator shall not contain side effects.	3415
12.5	The operands of a logical && or || shall be primary-expressions.	3398, 3399, 3400
12.7	Bitwise operators shall not be applied to operands whose underlying type is signed.	4532, 4533
12.8	The right hand operand of a shift operator shall lie between zero and one less than the width in bits of the underlying type of the left hand operand.	0499, 2790
12.9	The unary minus operator shall not be applied to an expression whose underlying type is unsigned.	3101, 3102
12.10	The comma operator shall not be used.	3417, 3418
12.12	The underlying bit representations of floating-point values shall not be used.	3629
13.1	Assignment operators shall not be used in expressions that yield a Boolean value.	3326
13.3	Floating-point expressions shall not be tested for equality or inequality.	3341
13.4	The controlling expression of a for statement shall not contain any objects of floating type.	3340, 3342
13.5	The three expressions of a for statement shall be concerned only with loop control.	2462, 2463
13.6	Numeric variables being used within a for loop for iteration counting shall not be modified in the body of the loop.	2469
13.7	Boolean operations whose results are invariant shall not be permitted.	2990, 2991, 2992, 2993, 2994, 2995, 2996
14.1	There shall be no unreachable code.	0594, 1460, 1503, 2008, 2742, 2744, 2880, 2882, 3219
14.2	All non-null statements shall either (i) have at least one side effect however executed, or (ii) cause control flow to change.	3110, 3112, 3425, 3426, 3427

continued on next page

Table 6.4 – continued from previous page

Rule Id	Rule	Enforcement
14.3	Before preprocessing, a null statement shall only occur on a line by itself; it may be followed by a comment provided that the first character following the null statement is a white-space character.	3138
14.4	The goto statement shall not be used.	2001
14.5	The continue statement shall not be used.	0770
14.6	For any iteration statement there shall be at most one break statement used for loop termination.	0771
14.7	A function shall have a single point of exit at the end of the function.	2889
14.8	The statement forming the body of a switch, while, do ... while or for statement shall be a compound statement.	2212, 2214
14.9	An if (expression) construct shall be followed by a compound statement. The else keyword shall be followed by either a compound statement, or another if statement.	2212, 2214, 3402
14.10	All if ... else if constructs shall be terminated with an else clause.	2004
15.0	The MISRA C switch syntax shall be used.	3234
15.1	A switch label shall only be used when the most closely-enclosing compound statement is the body of a switch statement.	2019
15.2	An unconditional break statement shall terminate every non-empty switch clause.	2003, 2020
15.3	The final clause of a switch statement shall be the default clause.	2002, 2009
15.4	A switch expression shall not represent a value that is effectively Boolean.	0735
15.5	Every switch statement shall have at least one case clause.	3315
16.1	Functions shall not be defined with a variable number of arguments.	1337
16.2	Functions shall not call themselves, either directly or indirectly.	1520, 3670
16.3	Identifiers shall be given for all of the parameters in a function prototype declaration.	1335, 1336
16.4	The identifiers used in the declaration and definition of a function shall be identical.	1330
16.5	Functions with no parameters shall be declared and defined with the parameter list void.	3001, 3007
16.6	The number of arguments passed to a function shall match the number of parameters.	0422, 0423, 3319
16.8	All exit paths from a function with non-void return type shall have an explicit return statement with an expression.	0745, 2887, 2888, 3113, 3114

continued on next page

Table 6.4 – continued from previous page

Rule Id	Rule	Enforcement
16.9	A function identifier shall only be used with either a preceding &, or with a parenthesised parameter list, which may be empty.	3635
16.10	If a function returns error information, then that error information shall be tested.	3200
17.1	Pointer arithmetic shall only be applied to pointers that address an array or array element.	2930, 2931, 2932
17.2	Pointer subtraction shall only be applied to pointers that address elements of the same array.	2771, 2772
17.3	>, >=, <, <= shall not be applied to pointer types except where they point to the same array.	2771, 2772
17.4	Array indexing shall be the only allowed form of pointer arithmetic.	0488, 0489, 0491, 0492
17.6	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.	3217, 3225, 3230, 4140
18.1	All structure and union types shall be complete at the end of a translation unit.	0544, 0545, 0623, 0636
18.2	An object shall not be assigned to an overlapping object.	2776, 2777
18.4	Unions shall not be used.	0750, 0759
19.3	The #include directive shall be followed by either a <filename> or “filename” sequence.	0809
19.4	C macros shall only expand to a braced initialiser, a constant, a string literal, a parenthesised expression, a type qualifier, a storage class specifier, or a do-while-zero construct.	3409, 3411, 3412, 3413, 3431, 3452, 3458, 3460, 3461
19.5	Macros shall not be #define’d or #undef’d within a block.	0842
19.6	#undef shall not be used.	0841
19.8	A function-like macro shall not be invoked without all of its arguments.	0850, 0856
19.9	Arguments to a function-like macro shall not contain tokens that look like preprocessing directives.	0853
19.10	In the definition of a function-like macro each instance of a parameter shall be enclosed in parentheses unless it is used as the operand of # or ##.	3410
19.11	All macro identifiers in preprocessor directives shall be defined before use, except in #ifdef and #ifndef preprocessor directives and the defined() operator.	3332
19.12	There shall be at most one occurrence of the # or ## preprocessor operators in a single macro definition.	0880, 0881, 0884
19.14	The defined preprocessor operator shall only be used in one of the two standard forms.	0885, 0887, 0888

continued on next page

Table 6.4 – continued from previous page

Rule Id	Rule	Enforcement
19.15	Precautions shall be taken in order to prevent the contents of a header file being included twice.	0883
19.16	Preprocessing directives shall be syntactically meaningful even when excluded by the preprocessor.	3115
19.17	All #else, #elif and #endif preprocessor directives shall reside in the same file as the #if or #ifdef directive to which they are related.	3317, 3318
20.1	Reserved identifiers, macros and functions in the standard library, shall not be defined, redefined or undefined.	0836, 0848, 0854, 3439, 4600, 4601
20.2	The names of standard library macros, objects and functions shall not be reused.	0602, 4602, 4603, 4604, 4605, 4606, 4607, 4608
20.3	The validity of values passed to library functions shall be checked.	2810, 2811, 2812, 2840, 2841, 2842
20.4	Dynamic heap memory allocation shall not be used.	5118
20.5	The error indicator errno shall not be used.	5119
20.6	The macro offsetof, in library <stddef.h>, shall not be used.	5120
20.7	The setjmp macro and the longjmp function shall not be used.	5122
20.8	The signal handling facilities of <signal.h> shall not be used.	5123
20.9	The input/output library <stdio.h> shall not be used in production code.	5124
20.10	The library functions atof, atoi and atol from library <stdlib.h> shall not be used.	5125
20.11	The library functions abort, exit, getenv and system from library <stdlib.h> shall not be used.	5126
20.12	The time handling functions of library <time.h> shall not be used.	5127
21.1	Minimisation of run-time failures shall be ensured by the use of at least one of (a) static analysis tools/techniques; (b) dynamic analysis tools/techniques; (c) explicit coding of checks to handle run-time faults	2791, 2792, 2801, 2802, 2811, 2812, 2821, 2822, 2831, 2832, 2841, 2842, 2845, 2846, 2847, 2871, 2872, 2877, 2891, 2892, 2896, 2897, 2901, 2902, 2911, 2912, 2980, 2981, 2982, 2983, 2984, 2985, 2986

Count of Enforced Required Rules = 116

6.5 Not Statically Enforceable Rules - Required

Table 6.5: M2CM Not Statically Enforceable Rules - Required

Rule Id	Rule
1.3	Multiple compilers and/or languages shall only be used if there is a common defined interface standard for object code to which the languages/compilers/assemblers conform.
1.4	The compiler/linker shall be checked to ensure that 31 character significance and case sensitivity are supported for external identifiers.
3.2	The character set and the corresponding encoding shall be documented.
3.5	If it is being relied upon, the implementation-defined behaviour and packing of bit-fields shall be documented.
3.6	All libraries used in production code shall be written to comply with the provisions of this document, and shall have been subject to appropriate validation.
18.3	An area of memory shall not be reused for unrelated purposes.

Count of Not Statically Enforceable Required Rules = 6

6.6 Rule Enforcement Summary

Table 6.6: M2CM Rule Enforcement Summary

Rule	Number
(a) Total Number of Rules	142
(b) Total Number of 'Not Statically Enforceable' Rules	9
(c) Total Number of Enforceable Rules (a-b)	133
(d) Total Number of Enforced Rules	132
(e) Total Number of Unenforced Rules (c-d)	1
(f) Enforced Rules Percentage (d/c)	99%
(g) Unenforced Rules Percentage (e/c)	1%