MBD Platform

Software Coding Guidelines

How to Contact MathWorks

Latest news: [www.mathworks.com/](https://www.mathworks.com/)

Sales and services: [www.mathworks.com/sales\_and\_services](https://www.mathworks.com/sales_and_services)

User community: [www.mathworks.com/matlabcentral](https://www.mathworks.com/matlabcentral)

Technical support: [www.mathworks.com/support/contact\_us](https://www.mathworks.com/support/contact_us)

Phone: 508-647-7000

The MathWorks, Inc.  
1 Apple Hill Drive  
Natick, MA 01760-2098

Software Coding Guidelines

© COPYRIGHT 2021 by The MathWorks, Inc.

The software described in this document is furnished under a license agreement. The software may be used or copied only under the terms of the license agreement. No part of this manual may be photocopied or reproduced in any form without prior written consent from The MathWorks, Inc.

FEDERAL ACQUISITION: This provision applies to all acquisitions of the Program and Documentation by, for, or through the federal government of the United States. By accepting delivery of the Program or Documentation, the government hereby agrees that this software or documentation qualifies as commercial computer software or commercial computer software documentation as such terms are used or defined in FAR 12.212, DFARS Part 227.72, and DFARS 252.227-7014. Accordingly, the terms and conditions of this Agreement and only those rights specified in this Agreement, shall pertain to and govern the use, modification, reproduction, release, performance, display, and disclosure of the Program and Documentation by the federal government (or other entity acquiring for or through the federal government) and shall supersede any conflicting contractual terms or conditions. If this License fails to meet the government’s needs or is inconsistent in any respect with federal procurement law, the government agrees to return the Program and Documentation, unused, to The MathWorks, Inc.

Trademarks

MATLAB and Simulink are registered trademarks of The MathWorks, Inc. See [www.mathworks.com/trademarks](http://www.mathworks.com/trademarks) for a list of additional trademarks. Other product or brand names may be trademarks or registered trademarks of their respective holders.

Patents

MathWorks products are protected by one or more U.S. patents. Please see [www.mathworks.com/patents](https://www.mathworks.com/patents) for more information.

Document Title: **Software Coding Guidelines for** *<Project>*

**Document Number:** *<Doc No>*

Revision: *<Revision>*

Project: *<Project>*

|  |  |  |
| --- | --- | --- |
| Approvals: |  |  |
|  |  |  |
|  |  |  |
| *<Name>*, Author |  | Date |
|  |  |  |
| *<Name>*, Project Management |  | Date |
|  |  |  |
| *<Name>*, Engineering |  | Date |
|  |  |  |
| *<Name>*, Quality Engineering |  | Date |

| Change History | | | | |
| --- | --- | --- | --- | --- |
| Revision | Modification/Description | Date | Author | Checked |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

Contents

[1 Introduction 1-1](#_Toc88563918)

[1.1 Purpose and Scope 1-1](#_Toc88563919)

[1.2 Applicable Documents 1-1](#_Toc88563920)

[1.3 Referenced Documents 1-2](#_Toc88563921)

[2 Programming Languages 2-1](#_Toc88563922)

[2.1 C Language 2-1](#_Toc88563923)

[3 Source Code Presentation Standards 3-1](#_Toc88563924)

[3.1 Coding Guidelines for Safety Critical Systems 3-1](#_Toc88563925)

[3.1.1 MISRA C:2012 Directives 3-1](#_Toc88563926)

[3.1.2 MISRA C:2012 Rules 3-2](#_Toc88563927)

[4 Naming Conventions 4-1](#_Toc88563928)

[4.1 Automatic Generated Identifiers 4-1](#_Toc88563929)

[4.2 User-Defined Identifiers 4-1](#_Toc88563930)

[5 Coding Conditions and Constraints 5-1](#_Toc88563931)

[5.1 Coding Conditions 5-1](#_Toc88563932)

[5.2 Coding Constraints 5-1](#_Toc88563933)

[6 Constraints on Coding Tools 6-1](#_Toc88563934)

[6.1 Coder Constraints 6-1](#_Toc88563935)

[6.2 Compiler and Linker Constraints 6-1](#_Toc88563936)

[7 Appendix A: Mapping of Coding Guidelines to ISO 26262-6 7-1](#_Toc88563937)

[7.1 Design Principles for Software Unit Design and Implementation 7-1](#_Toc88563938)

# Introduction

## Purpose and Scope

This document comprises the Software Coding Guidelines as referenced by the Documentation of the Software Development Environment for the project *<Project>*. The Software Coding Guidelines define the methods, rules, and tools to be used to code the software.

This document defines all of the following:

* Programming languages.
* Source code presentation standards.
* Naming conventions.
* Conditions and constraints on coding conventions.
* Constraints on coding tools.

Within the project *<Project>*, this document is applicable to source code.

You can use this template as a resource when creating a Software Coding Guidelines document. If you are updating an existing Software Coding Guidelines document to support Model-Based Design (MBD), you can use this template as a reference.

## Applicable Documents

Table . Regulations and Standards

| **ID** | **Document Title** |
| --- | --- |
| ISO 26262-6 | Road Vehicles – Functional Safety – Part 6: Product Development at the Software Level. ISO, 2018 |
| ISO 26262-8 | Road Vehicles – Functional Safety – Part 8: Supporting Processes. ISO, 2018 |
| ISO 26262-9 | Road Vehicles – Functional Safety – Part 9: Automotive Safety Integrity Level (ASIL)-Oriented and Safety-Oriented Analyses. ISO, 2018 |
|  | *<List additional documents here, e.g. Advisory Circulars, EASA Certification Memos, etc.>* |

Table . Organizational & Project Plans, Standards, and Documents

| ID | Document Title |
| --- | --- |
| SDE | **Documentation of the Software Development Environment for** *<Project>* |
| SVP | **Software Verification Plan for** *<Project>* |
| SCMP | **Software Configuration Management Plan for** *<Project>* |
| SCCP | **Software Change Management (Control) Plan for** *<Project>* |
| SMG | **Software Modeling Guidelines for** *<Project>* |
| SCG | **Software Coding Guidelines for** *<Project>* |
|  | *<List additional documents here.>* |

This initial release will identify the versions of completed documents, versions of the tools used, and the initial software configuration.

If any of the plans are revised during the project, the reasons for the changes must be captured and documented.

## Referenced Documents

Table 3. Reference Materials

| ID | Document Title |
| --- | --- |
| MISRA | Guidelines for the Use of the C Language in Critical Systems (MISRA C:2012) |
| MISRA AMD-1 | Additional Security Guidelines for MISRA C:2012 (MISRA C:2012 Amendment 1) |
| MISRA AMD-2 | Updates for ISO/IEC 9899:2011 Core Functionality (MISRA C:2012 Amendment 2) |
|  | *<List additional documents here.>* |

# Programming Languages

This section specifies the programming languages for the development of source code.

## C Language

C is a high-level computer programming language used for imperative, structured programming. C must be used in conjunction with a compiler to translate the source code into the executable object code.

*<Indicate which C standard, i.e. C90 or C99, you are adopting here.>*

# Source Code Presentation Standards

This section specifies the rules and style guidelines for the use of C as a programming language for source code development.

*<*The following guidelines provide a starting point for defining constraints on the use of coding tools within your project. The latest revision of MISRA C:2012 is available for purchase from[*The MISRA Consortium Limited*](https://www.misra.org.uk/shop)*.* Tailor the following set of guidelines based on your planned use of the language and applicability to your project.>

## Coding Guidelines for Safety Critical Systems

The Guidelines for the Use of the C Language in Critical Systems contain guidelines for developing code for high-integrity systems using structured programming in C. The guidelines provide guidance on identifiers, expressions, initialization, control flow, pointers, preprocessors, etc. to promote best practices for developing safety-related embedded software.

*<Use the Guidelines for the Use of the C Language in Critical Systems when defining your own subset of guidelines.* The latest revision of MISRA C:2012 is available for purchase from[*The MISRA Consortium Limited*](https://www.misra.org.uk/shop)*. For this chapter, consider listing MISRA C:2012 directives and rules that are essential to the quality of automatically generated code.>*

The Software Coding Guidelines given in this document are made up of a subset of MISRA C:2012 directives and rules that are deemed most critical to the reliability of automatically generated code.

|  |  |
| --- | --- |
| **Note** | The category of each MISRA directive or rule inserted in this document is based on its applicability to automatically generated code as explained in Appendix E of MISRA C:2012. |

### MISRA C:2012 Directives

The Implementation

* 1.1: Any implementation-defined behavior on which the output of the program depends shall be documented and understood. [Category: Required]

Compilation and Build

* 2.1: All source files shall compile without any compilation errors. [Category: Required]

Code Design

* 4.1: Run-time failures shall be minimized. [Category: Required]
* 4.3: Assembly language shall be encapsulated and isolated. [Category: Required]
* 4.7: If a function returns error information, then that error information shall be tested. [Category: Required]
* 4.10: Precautions shall be taken in order to prevent the contents of a header file being included more than once. [Category: Required]
* 4.11: The validity of values passed to library functions shall be checked. [Category: Required]
* 4.12: Dynamic memory allocation shall not be used. [Category: Required]
* 4.14: The validity of values received from external sources shall be checked. [Category: Required]

### MISRA C:2012 Rules

A standard C Environment

* 1.1: The program shall contain no violations of the standard C syntax and constraints, and shall not exceed the implementation’s translation limits. [Category: Required]
* 1.3: There shall be no occurrence of undefined or critical unspecified behavior. [Category: Required]
* 1.4: Emergent language features shall not be used. [Category: Required]

Unused Code

* 2.1: A project shall not contain unreachable code. [Category: Required]
* 2.2: There shall be no dead code. [Category: Required]

Comments

* 3.1: The character sequences /\* and // shall not be used within a comment. [Category: Required]
* 3.2: Line-splicing shall not be used in // comments. [Category: Required]

Character Sets and Lexical Conventions

* 4.1: Octal and hexadecimal escape sequences shall be terminated. [Category: Required]

Identifiers

* 5.1: External identifiers shall be distinct. [Category: Required]
* 5.2: Identifiers declared in the same scope and name space shall be distinct. [Category: Required]
* 5.4: Macro identifiers shall be distinct. [Category: Required]
* 5.5: Identifiers shall be distinct from macro names. [Category: Required]
* 5.6: A typedef name shall be a unique identifier. [Category: Required]
* 5.7: A tag name shall be a unique identifier. [Category: Required]
* 5.8: Identifiers that define objects or functions with external linkage shall be unique. [Category: Required]

Types

* 6.1: Bit-fields shall only be declared with an appropriate type. [Category: Required]
* 6.2: Single-bit named bit fields shall not be of a signed type. [Category: Required]

Literals and Constants

* 7.4: A string literal shall not be assigned to an object unless the object’s type is "pointer to const-qualified char". [Category: Required]

Declarations and Definitions

* 8.1: Types shall be explicitly specified. [Category: Required]
* 8.2: Function types shall be in prototype form with named parameters. [Category: Required]
* 8.3: All declarations of an object or function shall use the same names and type qualifiers. [Category: Required]
* 8.6: An identifier with external linkage shall have exactly one external definition. [Category: Required]
* 8.8: The static storage class specifier shall be used in all declarations of objects and functions that have internal linkage. [Category: Required]
* 8.10: An inline function shall be declared with the static storage class. [Category: Required]
* 8.12: Within an enumerator list, the value of an implicitly-specified enumeration constant shall be unique. [Category: Required]

Initialization

* 9.1: The value of an object with automatic storage duration shall not be read before it has been set. [Category: Mandatory]
* 9.4: An element of an object shall not be initialized more than once. [Category: Required]

Pointer Type Conversions

* 11.1: Conversions shall not be performed between a pointer to a function and any other type. [Category: Required]
* 11.2: Conversions shall not be performed between a pointer to an incomplete type and any other type. [Category: Required]
* 11.3: A cast shall not be performed between a pointer to object type and a pointer to a different object type. [Category: Required]
* 11.6: A cast shall not be performed between pointer to void and an arithmetic type. [Category: Required]
* 11.7: A cast shall not be performed between pointer to object and a non-integer arithmetic type. [Category: Required]
* 11.8: A cast shall not remove any const or volatile qualification from the type pointed to by a pointer. [Category: Required]

Expressions

* 12.2: The right hand operand of a shift operator shall lie in the range zero to one less than the width in bits of the essential type of the left hand operand. [Category: Required]
* 12.5: The sizeof operator shall not have an operand which is a function parameter declared as "array of type". [Category: Mandatory]

Side Effects

* 13.1: Initializer lists shall not contain persistent side effects. [Category: Required]
* 13.2: The value of an expression and its persistent side effects shall be the same under all permitted evaluation orders. [Category: Required]
* 13.5: The right hand operand of a logical && or || operator shall not contain persistent side effects. [Category: Required]
* 13.6: The operand of the sizeof operator shall not contain any expression which has potential side effects. [Category: Mandatory]

Control Statement Expressions

* 14.3: Controlling expressions shall not be invariant. [Category: Required]

Control Flow

* 15.6: The body of an iteration-statement or a selection-statement shall be a compound-statement. [Category: Required]

Functions

* 17.1: The features of <stdarg.h> shall not be used. [Category: Required]
* 17.2: Functions shall not call themselves, either directly or indirectly. [Category: Required]
* 17.3: A function shall not be declared implicitly. [Category: Mandatory]
* 17.4: All exit paths from a function with non-void return type shall have an explicit return statement with an expression. [Category: Mandatory]
* 17.6: The declaration of an array parameter shall not contain the static keyword between the [ ]. [Category: Mandatory]

Pointers and Arrays

* 18.1: A pointer resulting from arithmetic on a pointer operand shall address an element of the same array as that pointer operand. [Category: Required]
* 18.2: Subtraction between pointers shall only be applied to pointers that address elements of the same array. [Category: Required]
* 18.3: The relational operators >, >=, <, and <= shall not be applied to objects of pointer type except where they point into the same object. [Category: Required]
* 18.6: The address of an object with automatic storage shall not be copied to another object that persists after the first object has ceased to exist. [Category: Required]
* 18.7: Flexible array members shall not be declared. [Category: Required]
* 18.8: Variable-length array types shall not be used. [Category: Required]

Overlapping Storage

* 19.1: An object shall not be assigned or copied to an overlapping object. [Category: Mandatory]

Preprocessing Directives

* 20.2: The ', ", or \ characters and the /\* or // character sequences shall not occur in a header file name. [Category: Required]
* 20.3: The #include directive shall be followed by either a <filename> or "filename" sequence. [Category: Required]
* 20.4: A macro shall not be defined with the same name as a keyword. [Category: Required]
* 20.6: Tokens that look like a preprocessing directive shall not occur within a macro argument. [Category: Required]
* 20.7: Expressions resulting from the expansion of macro parameters shall be enclosed in parentheses. [Category: Required]
* 20.9: All identifiers used in the controlling expression of #if or #elif preprocessing directives shall be #define’d before evaluation. [Category: Required]
* 20.11: A macro parameter immediately following a # operator shall not immediately be followed by a ## operator. [Category: Required]
* 20.12: A macro parameter used as an operand to the # or ## operators, which is itself subject to further macro replacement, shall only be used as an operand to these operators. [Category: Required]
* 20.13: A line whose first token is # shall be a valid preprocessing directive. [Category: Required]
* 20.14: All #else, #elif, and #endif preprocessor directives shall reside in the same file as the #if, #ifdef, or #ifndef directive to which they are related. [Category: Required]

Standard Libraries

* 21.1: #define and #undef shall not be used on a reserved identifier or reserved macro name. [Category: Required]
* 21.2: A reserved identifier or macro name shall not be declared. [Category: Required]
* 21.3: The memory allocation and deallocation functions of <stdlib.h> shall not be used. [Category: Required]
* 21.4: The standard header file <setjmp.h> shall not be used. [Category: Required]
* 21.5: The standard header file <signal.h> shall not be used. [Category: Required]
* 21.6: The Standard Library input/output functions shall not be used. [Category: Required]
* 21.7: The atof, atoi, atol, and atoll functions of <stdlib.h> shall not be used. [Category: Required]
* 21.8: The library functions abort, exit, getenv, and system of <stdlib.h> shall not be used. [Category: Required]
* 21.9: The library functions bsearch and qsort of <stdlib.h> shall not be used. [Category: Required]
* 21.10: The Standard Library time and date functions shall not be used. [Category: Required]
* 21.11: The standard header file <tgmath.h> shall not be used. [Category: Required]
* 21.13: Any value passed to a function in <ctype.h> shall be representable as an unsigned char or be the value EOF. [Category: Mandatory]
* 21.14: The Standard Library function memcmp shall not be used to compare null terminated strings. [Category: Required]
* 21.15: The pointer arguments to the Standard Library function memcpy, memmove, and memcmp shall be pointers to qualified or unqualified versions of compatible types. [Category: Required]
* 21.16: The pointer arguments to the Standard Library function memcpy shall point to either a pointer type, an essentially signed type, an essentially unsigned type, an essentially Boolean type or an essentially enum type. [Category: Required]
* 21.17: Use of the string handling functions from <string.h> shall not result in accesses beyond the bounds of the objects referenced by their pointer parameters. [Category: Mandatory]
* 21.18: The size\_t argument passed to any function in <string.h> shall have an appropriate value. [Category: Mandatory]
* 21.19: The pointers returned by the Standard Library functions localeconv, getenv, setlocale, or strerror shall only be used as if they have pointer to const-qualified type. [Category: Mandatory]
* 21.20: The pointers returned by the Standard Library functions asctime, ctime, gmtime, localtime, localeconv, getenv, setlocale, or strerror shall not be used following a subsequent call to the same function. [Category: Mandatory]
* 21.21: The Standard Library function system of <stdlib.h> shall not be used. [Category: Required]

Resources

* 22.1: All resources obtained dynamically by means of Standard Library functions shall be explicitly released. [Category: Required]
* 22.2: A block of memory shall only be freed if it was allocated by means of a Standard Library function. [Category: Mandatory]
* 22.3: The same file shall not be open for read and write access at the same time on different streams. [Category: Required]
* 22.4: There shall be no attempt to write to a stream which has been opened as read-only. [Category: Mandatory]
* 22.5: A pointer to a FILE object shall not be dereferenced. [Category: Mandatory]
* 22.6: The value of a pointer to a FILE shall not be used after the associated stream has been closed. [Category: Mandatory]
* 22.7: The macro EOF shall only be compared with the unmodified return value from any Standard Library function capable of returning EOF. [Category: Required]
* 22.8: The value of errno shall be set to zero prior to a call to an errno-setting-function. [Category: Required]
* 22.9: The value of errno shall be tested against zero after calling an errno-setting-function. [Category: Required]
* 22.10: The value of errno shall only be tested when the last function to be called was an errno-setting-function. [Category: Required]

# Naming Conventions

This section specifies the naming conventions for functions, variables, and constants in the source code.

## Automatic Generated Identifiers

*<Describe the naming conventions for identifiers that are governed by the automatic code generator here.>*

## User-Defined Identifiers

*<Describe your naming conventions for identifiers that are user controlled here.>*

# Coding Conditions and Constraints

This section specifies the conditions and constraints on the allowable coding conventions.

## Coding Conditions

*<Describe your coding conditions here.>*

## Coding Constraints

*<Describe your coding constraints here.>*

# Constraints on Coding Tools

## Coder Constraints

*<Describe your constraints on the use of the coding tools, i.e. the automatic code generator, here.>*

## Compiler and Linker Constraints

*<Describe your constraints on the use of the compiler and linker here.>*

# Appendix A: Mapping of Coding Guidelines to ISO 26262-6

This section associates the Software Coding Guidelines to the applicable software design principles documented in ISO 26262-6:2018.

## Design Principles for Software Unit Design and Implementation

Table 6 of ISO 26262-6:2018 provides a list of design principles for software unit design and implementation. The MISRA C:2012 coding guidelines cover many of these principles. The table below identifies the coding guidelines that are relevant to each design principle.

Keep in mind that the collection of guidelines you select should address a combination of principles that are applicable for the ASIL under consideration.

Mapping of Guidelines to Design Principles for Software Unit Design and Implementation

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Design Principle** | **Guideline ID** | |
| **Applied Directive/Rule** | **Ignored Directive/Rule** |
| 1a | One entry and one exit point in subprograms and functions |  | Rule 15.5 [Advisory] |
| 1b | No dynamic objects or variables, or else online test during their creation | Dir 4.12 [Required] |  |
| 1c | Initialization of variables | Rule 9.1 [Mandatory] | Rule 9.2 [Readability] Rule 9.3 [Readability] |
| 1d | No multiple use of variable names | Rule 5.1 [Required] Rule 5.2 [Required] Rule 5.8 [Required] | Rule 5.3 [Advisory] Rule 5.9 [Readability] |
| 1e | Avoid global variables or else justify their usage |  |  |
| 1f | Restricted use of pointers | Rule 11.1 [Required] Rule 11.2 [Required] Rule 11.3 [Required] Rule 11.6 [Required] Rule 11.7 [Required] Rule 11.8 [Required] | Rule 11.4 [Advisory] Rule 11.5 [Advisory] Rule 11.9 [Readability] |
| 1g | No implicit type conversions |  | Rule 10.1 [Advisory] Rule 10.2 [Advisory] Rule 10.3 [Advisory] Rule 10.4 [Advisory] Rule 10.5 [Advisory] Rule 10.6 [Advisory] Rule 10.7 [Advisory] Rule 10.8 [Advisory] |
| 1h | No hidden data flow or control flow |  |  |
| 1i | No unconditional jumps |  | Rule 15.1 [Advisory] Rule 15.2 [Advisory] Rule 15.4 [Advisory] |
| 1j | No recursions | Rule 17.2 [Required] |  |