

HIGHLIGHTS

- Based on MISRA — “Guidelines for the Use of the C Language in Vehicle Based Software”
- Selectable restrictions to ISO/IEC 9899:1990 standard C programming language
- TASKING is the *only* toolset to include these *unique* code checking facilities in standard C compiler tools
- Supported on many TASKING compiler tools, including C166, ST10, TriCore, M16C, XA, and 8051
- Well suited for safety-related or mission-critical applications, or where general quality and robustness of code are important

SUMMARY

The C programming language is today's de facto standard for high-level language programming of embedded systems. Unfortunately, drawbacks in consistency and intrinsic quality make C somewhat unsuitable for programming safety-related applications. Through enhanced code checking and strict enforcement of best practice programming rules, TASKING's MISRA C code checking helps programmers produce more robust code.

Since February 1999, TASKING has been the only toolset to include unique facilities in standard C compiler tools that help programmers produce code that conforms to the guidelines defined by MISRA® (Motor Industry Software Reliability Association, www.misra.org.uk). These guidelines are a major benefit to programmers of embedded products in safety-related areas such as automotive, industrial control, communications, medical, and aerospace. But they are also valuable to any programmer of embedded applications concerned with the general quality and robustness of their work.

THE C LANGUAGE IN EMBEDDED SYSTEMS

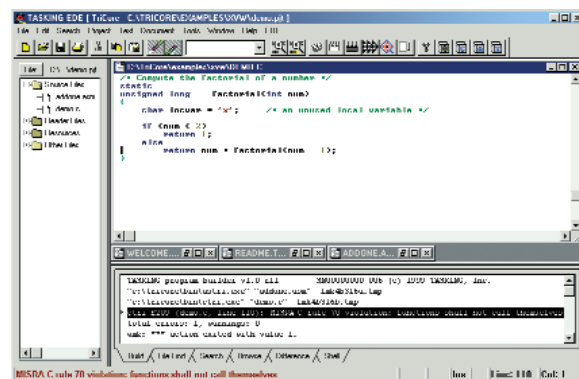
In the last decade, an increasing percentage of embedded microprocessor applications have been written in the programming language C. In fact, C has turned into the de facto standard for higher-level language programming of embedded applications.

A number of causes have led to the increased popularity of C in this area:

- The ever-increasing complexity of applications drives programmers from assembly to the high-level languages.
- The high-level programming language C offers good support for high-speed, low-level I/O operations. Programmers of embedded applications particularly appreciate this mixed high/low-level approach.
- In comparison to other high-level language compilers, C language compilers tend to deliver more condensed code size. With product costs in mind, this is generally considered a very important issue.
- Virtually all mathematical modelling tools generate C source code.
- C offers significant productivity gains with opportunities for code re-use, improved code maintenance, and ongoing developments over the life of the application.

However, using a high-level language such as C does not guarantee problem-free code.

- C can be written in a structured manner that reduces the chance of producing errors. But C can also be written in a very condensed manner, which is hard to comprehend and dramatically increases the likelihood of introducing errors.
- The compiler does not necessarily detect small typing errors. Consider the operators `&&`, `&`, `||`, `|`, `+=`, `=`, and `==`, and think of the ease with which a typo will still lead to perfectly valid C code.



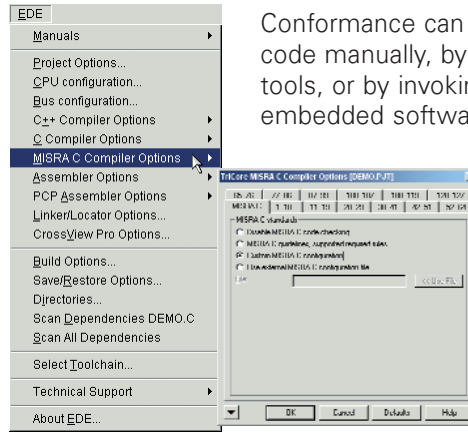
Wouldn't you like to be warned for a construct like this?

```
if ((x = y) != 0)
```



MISRA C rule 35 violation:
Assignment operators shall not be used in Boolean expressions

- Not every programmer is fully aware of the effects of all the possible constructs in the C language. For instance, casts (implicit or explicit) can cause both confusion and errors.
- A number of the features in C have not been well defined. Other definitions of C features differ from what the programmer would expect. Some compiler behavior is left undefined in the ISO C standard.
- One of the main reasons that C compilers do a great job of generating compact, efficient code is because of the limited run-time checking in C.



Conformance can be checked by reviewing the code manually, by using static code analysis tools, or by invoking integrated restrictions in the embedded software development tools.

The following list highlights some of the techniques that are contained in the MISRA C guidelines.

In the group "**Comments**," nested comments are prohibited and it is advised not to comment out sections of code.

In the group "**Identifiers**," a limit is defined of max 31-character significance, and the use of identical identifiers is discouraged.

In the group "**Types**," the basic types *char*, *int*, *short*, *long*, *float* and *double* should be replaced with typedefs indicating the specific length (e.g., *SI_16* for a 16 bit signed integer) and the type *char* shall always be declared as either unsigned *char* or signed *char*.

By definition, ANSI/ISO C compilers interpret decimal integer constants with leading zeros as Octal. To prevent confusing Octal and Integer constants, the use of Octal constants is prohibited.

In the group "**Conversions**," the use of implicit type conversions as well as redundant explicit casts are prohibited.

In the group "**Expressions**," a rule describes that the value of an expression should be the same under any permissible order of evaluation, and floating-point variables are not to be tested for exact equality or inequality.

In the group "**Control Flow**," the use of *goto*, *break* and *continue* is prohibited. Also a number of constraints on the use of the *if*, *else*, *switch*, and *case* constructs is defined.

The group "**Functions**" defines a large number of required rules on the declaration and use of functions.

The group "**Pointers and Arrays**" prohibits the use of non-constant pointers to functions and discourages the use of pointer arithmetic at all.

The group "**Structures and Unions**" requires that all structure/union members are named and referred to by name only.

Wouldn't you like to be warned for a construct like this?

```
I = (I == 2) && (J <= 1)
```

↑

MISRA C rule 35 violation:
Assignment operators shall not be used in Boolean expressions

There are no provisions in C that would prevent arithmetic exceptions such as divide by zero, overflow, validity of addresses or pointers, or surpassing array boundaries from causing a runtime software failure.

It is therefore easy to understand that programmers with a special interest in writing robust, consistent code have a concern with the programming language C.

MISRA C CONCEPT

C is accepted as the standard for higher-level language programming in embedded systems, yet it is considered unsuitable for programming safety-related applications. If a safety-related application needs to be programmed in a higher-level language, and C is the language of choice, then there are ways to avoid the drawbacks of C. Many of the companies developing safety-related embedded applications have written guidelines to restrict the use of error-prone C constructs with the intention of reducing the probability of errors.

Based on experience and expertise in automotive applications, as well as information from a number of public-domain sources, the Motor Industry Software Reliability Association (MISRA) defined a total of 127 programming rules applicable when developing safety-related applications in C. In

Guidelines for the Use of the C Language in Vehicle Based Software (MISRA C guidelines), MISRA has defined a core list of required rules and supplementary set of advisory rules.

Wouldn't you like to be warned for a construct like this?

```
I = (C << 8)
```

↑

MISRA C rule 38 violation:
A shift count shall be between 0 and the operand width minus 1

Although the MISRA C guidelines prescribe a wide range of programming techniques, they can also be interpreted as applying a subset of the C programming language that is intended to be suitable for safety-related applications.

Wouldn't you like to be warned for a construct like this?

```
A = 111;
B = 101;
C = 011;
```

↑

MISRA C rule 19 violation:
Octal Constants (other than zero) shall not be used

Wouldn't you like to be warned for a construct like this?

```
if (si++ || (ei == 11))
```

↑

MISRA C rule 34 violation:
The operands of a logical && or || shall be primary expressions

REFERENCES FOR FURTHER READING

MISRA, *Guidelines for the Use of C Language in Vehicle Based Software*, Motor Industry Research Association, 1998, ISBN 0952415690. <http://www.misra.org.uk>

Kernighan B.W., Ritchie D.M., *The C Programming Language*, second edition, Prentice Hall, 1988, ISBN 0131103628.

INTERNET

Web site: www.tasking.com
Developers forum: www.yahogroups.com/group/TASKINGforum
MISRA: www.misra.org.uk

DISTRIBUTOR

TASKING, the TASKING logo, Altium and the Altium logos are trademarks or registered trademarks of Altium Limited or its subsidiaries. All other registered and unregistered trademarks referenced herein are the property of their respective owners and no trademark rights to the same is claimed. Altium assumes no responsibility for any errors that may appear in this document.

ALTIUM SALES OFFICES**NORTH AMERICA**

333 Elm Street
Dedham MA 02026-4530 USA
Telephone: +1 781 320 9400
Facsimile: +1 781 320 9212
Email: tasking.sales.na@altium.com

THE NETHERLANDS

Plotterweg 31
3821 BB Amersfoort
The Netherlands
Telephone: +31 33 455 85 84
Facsimile: +31 33 455 00 33
Email: tasking.sales.nl@altium.com

GERMANY

Eltinger Straße 61
D-71229 Leonberg
Germany
Telephone: +49 71 52 979 910
Facsimile: +49 71 52 979 9120
Email: tasking.sales.de@altium.com

FRANCE

21 Avenue du Québec
91951 Les Ulis Cedex
France
Telephone: +33 1 69 59 26 10
Facsimile: +33 1 69 59 26 11
Email: tasking.sales.fr@altium.com

SWITZERLAND

Unterdorfstrasse 1
CH-4334 Sisseln
Switzerland
Telephone: +41 62 866 41 11
Facsimile: +41 62 866 41 10
Email: tasking.sales.ch@altium.com

JAPAN

ASAHI-GIN Gotanda Building 7F
23-9, Nishi-Gotanda 1-chome
Shinagawa-ku Tokyo 141-0031
Japan
Telephone: +81 3 5436 2501
Facsimile: +81 3 5436 2505
Email: tasking.sales.jp@altium.com